Réseaux
et systèmes
d'information

Bull

MANUEL DE RÉFÉRENCE

SOLAR

FMS16

Système de gestion de fichiers

LOGICIEL

LOGICIEL

LOGICIEL

LOGICIEL

LOGICIEL

1 164 226 00 036

Bull

Réseaux
et systèmes
d'information

Bull

Manuel de référence

Bull Sems

SOLAR

SYSTEME D'EXPLOITATION

FMS16 ADDENDUM A

Logiciel

SUJET : Système de gestion de fichiers

OBSERVATION : Première mise à jour du document 1 164 226 01 036 00 de Février 1984. Suivre les directives de mise à jour et placer la présente feuille directement après la couverture de votre manuel pour indiquer que celui-ci est à l'indice 01.

VERSION LOGICIEL :

DATE : DECEMBRE 1985

REF Bull-Sems : 1 164 226 00 036 01/FR

Réseaux
et systèmes
d'information

Bull

Bull
Sems

Imprimé en France

| Vos suggestions sur la forme et le fond de ce manuel seront les bienvenues. Une feuille destinée à recevoir vos remarques se trouve à la fin du présent manuel.

AVERTISSEMENT

Le lecteur trouvera dans cette notice :

- Une introduction technique et générale, au produit FMS (chapitre 1 et 2)
- La description fonctionnelle des requêtes offertes par FMS pour accéder aux fichiers, ainsi que la spécification précise, de leur interface de programmation. (chapitre 3 à 9).
- Une vue d'ensemble sera obtenue par la lecture de la présentation et des paragraphes 2.1, 2.3, 2.4 de l'introduction.

Le lecteur se reportera au Manuel d'Utilisation de FMS à fin d'y trouver, un ensemble de conseils concernant l'utilisation globale du produit :

- Mise en oeuvre pratique du produit ; c'est-à-dire comment intégrer FMS dans un système d'exploitation.
- Un ensemble d'informations permettant d'optimiser l'accès aux fichiers.
- Un ensemble d'informations décrivant la structure et le fonctionnement interne de FMS.

Dans la suite de ce manuel, la référence FMS16 sera remplacée par FMS.

SOMMAIRE GENERAL

1 — PRESENTATION

1 — PRESENTATION

FILE MANAGEMENT SYSTEM

1.1 - DEFINITIONS

Le système de fichiers :

FMS est un module système, permettant de gérer des informations organisées en fichiers, et implantées physiquement sur des mémoires secondaires.
FMS est un module système, c'est à dire un **composant** de différents systèmes d'Exploitation (BOS16, RTES16, ...) au même titre que IOCS, le moniteur d'entrées - sorties. Comme IOCS, il est activable par des requêtes programmées (SVC : appels superviseur).

Les mémoires secondaires :

Les mémoires secondaires sur lesquelles FMS est capable de gérer des fichiers ont toutes en **commun** la **propriété d'être** adressables par **secteurs.**
FMS gère des disques dont la taille secteur est égale à 128 mots de 16 bits.

Un fichier :

Le terme de fichier recouvre des notions **différentes** selon le contexte dans lequel il est employé.

Ainsi par exemple, lorsque l'on parle d'un Fichier Produits, on s'intéresse à l'**ensemble** des **informations** décrivant les produits fabriqués par une entreprise. Cet ensemble d'informations permet par exemple, de réaliser les factures correspondant aux commandes des Clients de l'entreprise.

Par contre, lorsque l'on dit que le fichier des produits est de type Séquentiel Indexé, le terme de fichier désigne le **récipient** dans lequel ont été placées les informations de l'utilisateur.
Ainsi d'un point de vue utilisateur, le terme de fichier désigne un **contenu**, c'est à dire un ensemble d'informations utilisateur, et du point de vue des Systèmes de Fichiers, il désigne un **contenant** d'informations.
Pour FMS un contenant est représenté par l'ensemble des informations, dites système, qui identifient et décrivent ce contenant.
Afin de comprendre le fonctionnement de FMS l'usager pourra employer une définition simple :

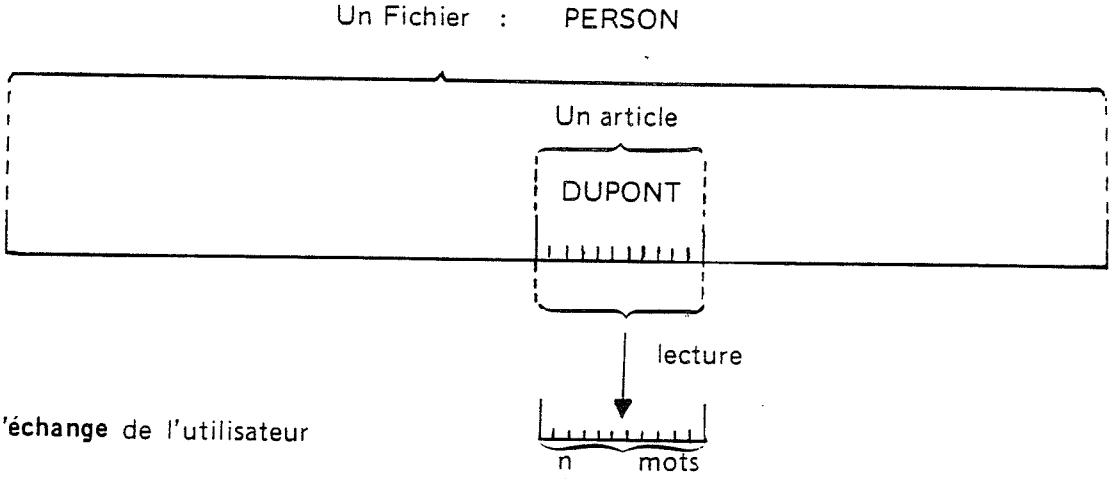
Un Fichier est un récipient d'informations, identifié par un nom.

Le transfert d'Informations :

Il semble dès maintenant important de décrire précisément, la fonction de base de l'outil FMS : le **transfert d'informations** entre une mémoire secondaire et l'utilisateur.

L'utilisateur ou **usager**, deux termes utilisés de façon équivalente dans la présente notice, représentent soit une **personne**, soit un **programme d'application** en cours d'exécution.
C'est l'utilisateur, au sens d'un programme d'application, qui adresse au système d'exploitation une **requête** (SVC) spécifiant un transfert d'informations.

Exemple de requête : LIRE **l'article de nom** : DUPONT appartenant au **fichier** de nom : PERSON



Une zone d'échange est une zone de mémoire centrale exploitable par un programme d'application.
Pour FMS un **article** est une partie d'un fichier. C'est un **contenant** d'informations, situé physiquement sur une mémoire secondaire et **identifié** dans cet exemple par un nom : DUPONT.
Le contenu d'un article est pour l'utilisateur un ensemble **logique** d'informations, décrivant par exemple la personne DUPONT. Le contenu d'un article est pour FMS un **vecteur de n mots** logiquement contigus.
Dans l'exemple ci-dessus l'utilisateur demande donc à FMS :

1) - De rechercher le contenant de nom : DUPONT, c'est à dire de savoir où il se trouve sur la mémoire secondaire.
2) - De placer le contenu correspondant dans la zone d'échange, spécifiée par l'utilisateur à l'aide des paramètres associés à sa requête.

A l'utilisateur incombe la gestion des contenus. A FMS incombe la gestion des contenants. Aux deux parties incombent la connaissance des noms des fichiers et des articles.

1.2 - LA GESTION DYNAMIQUE DES SUPPORTS PHYSIQUES

FMS se propose de fournir à l'utilisateur, des outils d'accès à l'information plus **évolués** que ceux offerts par IOCS. En effet lorsque le programmeur d'application ne dispose que d'un outil du type : Moniteur d'entrées - sorties (IOCS), c'est à lui qu'incombe la gestion des supports physiques selon leurs caractéristiques techniques.
Parmi les nombreux problèmes liés à l'utilisation des mémoires secondaires citons à titre d'exemple, les deux suivants :

- Stocker des informations dans une mémoire, suppose d'avoir résolu les problèmes d'**allocation** et de **désallocation** de mémoire.
- Réaliser des E/S IOCS sur un disque à bras mobiles par exemple, suppose d'avoir résolu les problèmes liés à l'existence des **secteurs** et **cylindres.**

Des applications évolutives

Il est fréquent de constater que les **applications** traitées à l'aide d'un système informatique **évoluent** dans le temps.

- De par leur nature propre. Par exemple, tout au long d'une année (d'un mois, d'un jour) d'exploitation, le volume d'information à traiter varie dans des proportions telles qu'il est impératif de **récupérer la place** libérée par une information éliminée du système.
- De par le fait qu'il est souvent souhaitable pour des raisons d'économie de réaliser des programmes **polyvalents** traitant une classe d'applications. Ces programmes sont alors paramétrables au sens large, et le sont d'autant plus facilement qu'ils ne contiennent pas d'algorithmes liés à la gestion physique des mémoires secondaires.

Quelques exemples :

- Sur certains systèmes d'exploitation le sous dimensionnement d'un récipient d'informations ou fichier, oblige l'utilisateur à régénérer son système.
- Sur d'autres systèmes à fin de récupérer la place libérée par la destruction d'un grand nombre de fichiers, il est nécessaire de :
  1) Eventuellement, arrêter le fonctionnement temps réel du système.
  2) Employer un utilitaire de retassage du disque dont le temps d'exécution est de l'ordre de plusieurs dizaines de minutes, voir plusieurs heures.

FMS se propose comme objectif de permettre la réalisation de programmes d'application **évolutifs** et facilement **paramétrables.**

Indépendance vis à vis du hardware :

Lorsque la **technologie** du domaine informatique **évolue** rapidement, il semble également souhaitable de rendre les programmes d'application **indépendants des caractéristiques techniques** d'une nouvelle mémoire secondaire.

- FMS permet de lire des **articles** de32 K mots. Il traite donc automatiquement les problèmes liés à l'existence des cylindres dans un disque à bras mobiles.
- FMS réalise pour l'utilisateur des entrées - sorties au niveau du **mot** et non au niveau du secteur. Il traite donc automatiquement le fait qu'une information logique ne soit pas alignée en tête de secteur et par conséquent **optimise** également le taux d'occupation d'un disque.

— FMS permet à l'utilisateur de mettre en service une nouvelle mémoire secondaire de **plus grande capacité** sans modifier les programmes d'application et tout en **utilisant la nouvelle place disponible.**

Des méthodes d'accès évoluées :

La **gestion dynamique** des mémoires secondaires réalisée par FMS permet de construire des méthodes d'accès évoluées et pourtant simples à utiliser. Par exemple : les outils de mise au point de programmes, que les systèmes mettent à la disposition de l'utilisateur, utilisent fréquemment des fichiers temporaires séquentiels, et sont en particulier pour cette raison, de manipulation très simple.
En effet, la requête de **création** d'un fichier temporaire séquentiel ne contient que 4 paramètres qui sont :

- Le **nom du fichier** (6 caractères ASCII)
- Le **numéro désignant le support** sur lequel FMS devra créer le fichier. (1 octet)
- Le **type du fichier :** séquentiel (1/2 octet)
- Le **numéro d'utilisation** du fichier (1 octet) (ce numéro sera rappelé par les requêtes d'entrées-sorties pendant la création du fichier)

Les requêtes **d'écritures séquentielles** ne contiennent que 3 paramètres qui sont :

- Le **numéro d'utilisation** du fichier (1 octet) (ce numéro est défini par l'utilisateur à la création du fichier temporaire)
- **L'adresse** de la zone d'échange (1 mot)
- La **longueur** de la zone d'échange (14 bits).

FMS réalise dynamiquement l'allocation de place pour un fichier séquentiel ceci en **temps réel** et **automatiquement**, lors des requêtes d'écritures successives dans le fichier.

FMS permet à plusieurs utilisateurs de stocker sur une mémoire secondaire des fichiers permanents avec le **minimum de concertation.** Il suffit qu'il y ait de la place sur le support et que les usagers s'entendent pour assurer la non homonymie des noms de fichiers.
Par le fait que, **seul** FMS connait effectivement l'implantation physique des informations sur un disque, un programme d'application en cours de mise au point, par exemple, peut difficilement détruire les fichiers des autres usagers.
Autrement dit, FMS assure une **protection** de l'information supérieure à celles des systèmes demandant à l'utilisateur de gérer tout ou partie de l'allocation de place sur les mémoires secondaires.

Des méthodes d'accès performantes

Pour des objectifs différents, **dynamisme** ou **performance,** FMS gère deux types d'organisation physique pour les fichiers. Soit :

- **Des fichiers dynamiques**
- **Des fichiers statiques**

Fichiers Dynamiques

Le Séquentiel, L'Indexé,
FMS · réalise l'allocation et la désallocation dynamique, pendant la vie du fichier à la demande des requêtes d'écritures (WRITE, IWRITE).

Objectifs :

- Volume d'informations illimité (Séquentiel)
- Articles de taille variable (Indexé).

Fichiers Statiques

Direct, Direct à Trous, Séquentiel Index, Séquentiel Chainé
FMS réalise l'allocation dynamique de toute la place nécessaire, à la création du fichier.

Objectif :

- Performance Temps d'Accès

Exemple :
Pour créer un fichier Direct, il est seulement demandé à l'utilisateur de fournir :

- **le nom** du fichier (8 caractères)
- **le numéro** du support sur lequel FMS devra créer le fichier
- **la taille** des articles ($\leq$ 32 767)
- **le nombre** d'articles ($\leq$ 65 535)

A la création FMS alloue **toute la place** nécessaire au fichier (Taille x nombre).
Pendant la vie du fichier, FMS est capable de réaliser l'accès direct aux nombres (DREAD, DWRITE) en 1 **seul accès disque** (Taille d'article multiple de secteurs) : Soit un transfert direct : disque — zone d'échange.

Remarque : lorsqu'après l'initialisation d'un support, l'utilisateur crée successivement plusieurs fichiers statiques, ceux-ci sont alloués par rapport au début du support de façon **continue** et **dans l'ordre** de leur création. Le **premier** fichier créé se trouve au **début** du support.
Donc au vu de l'état d'allocation d'un support (FUP2) l'utilisateur est capable de placer ses fichiers dans un disque pour **optimiser** les déplacements de bras.

Le principe de l'allocation dynamique

Le système de fichiers constitue donc, un **écran** entre les utilisateurs et les supports physiques.
Le programmeur utilise le système de fichiers, ce dernier gère les supports physiques.

Pour atteindre un tel **objectif** FMS gère **dynamiquement l'allocation** et la **désallocation** de place sur les mémoires secondaires adressables.

Le principe en est très simple.
Un espace de mémoire secondaire est divisé en **unités d'allocation** de taille fixe.
L'unité d'allocation de mémoire secondaire adressable est appelée un **granule.** FMS alloue les granules au niveau des fichiers.
Lorsque l'on crée ou agrandit un fichier FMS lui alloue le nombre de granules libres dont il a besoin.
Lorsque l'on détruit ou réduit un fichier FMS récupère les granules ainsi rendus libres pour les fichiers du même espace physique de mémoire secondaire.

1.3 - UN ENSEMBLE DE METHODES D'ACCES

FMS propose à l'utilisateur de classer ses informations dans des fichiers et des articles, selon des critères appartenant plus à la logique de son application, qu'aux caractéristiques des supports physiques utilisés.

Un fichier possède une organisation interne, plus précisément il est constitué d'articles dont le nombre et la taille (la même pour tous ou non) est fixé par l'utilisateur à la création du fichier ou des articles. La structure d'un fichier en articles est appelée **organisation logique.** Cette structure se distingue de l'**organisation physique,** qui est celle utilisée de façon interne par FMS , à fin de s'adapter aux particularités des supports physiques).

FMS propose donc plusieurs organisations logiques ayant chacune des objectifs fonctionnels différents :

- **Séquentiel**
- **Indexé**
- **Direct ou Direct à Trous**
- **Séquentiel indexé**
- **Séquentiel chainé.**
- **Direct longueur variable**

Séquentiel : L'organisation logique d'un fichier séquentiel est adaptée au stockage de :
- Programmes : symbolique, binaire «Link-éditable», binaire Translatable.
- Données dont seul l'ordre chronologique est utilisé et dont le volume ne sera pas limité.

Indexé : L'organisation logique d'un fichier indexé est adaptée au stockage de :
- Programmes : image mémoire segmentée ou non. Les branches d'un programme segmenté étant identifiées par un nom.
- Bibliothèque de programmes identifiés par un nom.
- Données subdivisées en articles de longueur variable et identifiés par un nom.

Direct : L'organisation logique d'un fichier direct est adaptée au stockage de :
- Données subdivisées en articles de longueur constante et identifiés par un numéro (nombre entier de 1 à N) sur lesquels un accès direct rapide (1 accès disque) à l'article est indispensable.
- Données à organisation logique séquentielle dont la taille sera bornée.

Une version **Direct à Trous** permet de placer en vrac des articles dans un fichier direct. FMS se charge alors de trouver une place libre pour un nouvel article, et donc de lui affecter un numéro d'identification qu'il rendra à l'utilisateur.

Séquentiel Indexé : L'organisation logique d'un fichier séquentiel indexé est adaptée au stockage de :
- Données contenant un grand nombre d'articles (100 000 par exemple) de taille constante et pouvant être identifiés par une chaîne de caractères.
- Exemples d'application pratique : un fichier clients, un fichier produits.

Séquentiel Chaîné : L'organisation logique d'un fichier séquentiel chaîné est adaptée au stockage de :
- Données constituees de N «fichiers séquentiels», ou chaînes d'articles. Une chaîne d'articles est une suite chronologique d'articles de taille constante.
- Exemples d'application pratique : un fichier de mouvements sur des comptes clients.

Direct Longueur Variable : Cette méthode d'accès est adaptée à la gestion de modules de bibliothèques identifiés par un numéro de 0 à 32 767.

1.4 - UN ACCES AUX FICHIERS DANS UN CONTEXTE TEMPS REEL

FMS offre à l'utilisateur, la possibilité d'accéder aux fichiers dans un contexte de multiprogrammation.
Le calculateur sur lequel fonctionne FMS , gère un contexte de multiprogrammation du type : **Multi-tâches Temps Réel.**
Plus précisément la machine est capable de gérer N tâches (1 à 128). A chacune de ces tâches est affecté, de façon bijective un numéro de priorité. La règle de «scheduling» utilisée dans le conflit d'accès aux différentes ressources d'une installation, a pour but d'activer la tâche la plus **prioritaire.**
FMS a donc comme objectif de fonctionner dans un contexte temps réel, en sachant que dans un système temps réel il est parfois admis de demander aux tâches d'application de se concerter afin de rendre possible ou performant le partage des ressources du système.

Un fichier : une ressource

Les différents outils d'accès aux fichiers, ne peuvent pas être utilisés par les usagers d'un système informatique de façon anarchique. Pour cela FMS propose une deuxième définition d'un fichier :

Un fichier est une ressource possédant 1 ou plusieurs points d'entrée.

FMS propose alors que les accès aux fichiers se fassent selon la même règle pour tous les usagers. Pour respecter la règle commune de partage des accès aux fichiers. FMS fourni des outils de contrôle, permettant de **synchroniser les demandes d'accès** aux ressources fichiers.
Tout accès à un fichier doit être précédé d'une **demande d'accès.** Soit pour un fichier permanent l'exemple suivant :

OPEN OLD	demande d'accès à la ressource fichier contrôlée par FMS
READ	Phase d'accès à la ressource fichier contrôlée par FMS
REWIND	
WRITE	
REWIND	
READ	
CLOSE	Libération de la ressource fichier.

Pour FMS c'est un **usager** qui demande l'accès à un fichier, à l'aide d'une requête OPEN OLD par exemple. Voici quelques exemples pratiques de la notion d'usager pour FMS :
- Un «JOB» en cours d'exécution sous BOS16 (RUN)
- N tâches exécutant toutes, un même travail dans un système Temps Réel tel que RTES16
- Un utilisateur d'un système multi-consoles.

On peut donc constater qu'en général les différents utilisateurs ne peuvent se concerter pour se partager correctement les ressources fichiers. En particulier, ils ne peuvent pas toujours prévoir qu'un fichier sera libre à l'instant où ils désireront y accéder.

FMS a donc comme objectif de fournir des outils permettant à des usagers différents d'un même système informatique de se partager les fichiers dans un contexte Temps Réel.

FMS gère des fichiers temporaires.

Un fichier temporaire est créé, utilisé, puis détruit pendant une seule unité d'exécution d'un usager. Il est **non partageable** entre des usagers. C'est à dire que seul l'usager créateur peut y accéder.
FMS assure automatiquement la non homonymie des noms des fichiers temporaires d'usagers différents en même temps présent dans le système.

FMS gère des fichiers Permanents

Un fichier permanent est d'abord créé, utilisé (initialiser son contenu), puis fermé (au sens libération de ressource) par un usager. Pendant cette phase de création il est non partageable entre des usagers. Une fois la phase de création terminée un fichier permanent est **partageable**. Ce qu'il faut comprendre de la façon suivante : Il est accessible à un quelconque usager. Aucun fichier permanent n'est privé ou réservé à certains usagers.
Une fois la phase de création terminée, un fichier permanent est accessible de la façon suivante :

OPEN OLD	(Usager, Numéro d'util., Nom, Fichier...)
READ	(Usager, Numéro d'util. ...)
etc . . .	
CLOSE	(Usager, Numéro d'util.)

FMS gère **deux types** de fichiers permanents.

— Un fichier permanent peut être créé de type : **partageable simultanément entre des usagers.**
C'est à dire que N usagers différents peuvent accéder **simultanément** au même fichier dans un but de consultation (lectures seulement)
Lorsqu'un usager demande l'accès à un fichier pour le modifier (lire et écrire) le fichier doit être libre pour que sa demande soit acceptée. Une fois sa demande acceptée, toute autre demande d'accès est refusée jusqu'à ce que le fichier soit de nouveau libre.
Un fichier «Permanent Simultané» est une ressource partageable :
- A 1 point d'entrée lorsqu'il est ouvert en lecture et écriture.
- A N points d'entrée lorsqu'il est ouvert en lecture seulement.

— Un fichier permanent peut être créé de type : **partageable Non Simultanément entre des usagers.**
C'est à dire que lorsqu'un usager a obtenu le droit d'accéder à un fichier toute demande d'accès au même fichier, provenant d'un autre usager, est refusée jusqu'à ce que le fichier soit libre.
De ce point de vue, un Fichier «Permanent Non Simultané» est une ressource partageable à 1 seul point d'entrée.
Cependant une autre demande d'accès provenant du même usager, est à priori acceptée. On appellera ce type de partage du multi-accès. Il permet en particulier de réaliser du multi-accès en écriture sur des fichiers directs.
En plus des possibilités de partage décrites ci-dessus, l'objectif visé par ces deux types de Permanent est d'accroître la **sécurité** de fonctionnement d'un système Temps Réel avec «Background».
La règle d'utilisation proposée est alors la suivante :
- Les fichiers permanents du «Foreground» seront de type : non simultané.
- Les fichiers permanents du «Background» seront de type : simultané.

FMS donne le droit d'acceder à un fichier permanent, que si l'utilisateur a spécifié le bon type de permanent en paramètre de sa demande d'accès (OPEN OLD).

De plus FMS fournira des outils pour **modifier** les caractéristiques d'un fichier. En particulier l'utilisateur pourra demander à ce qu'un fichier soit protégé en écriture ou au contraire accessible en écriture.
Un utilisateur pourra également demander à ce qu'un fichier permanent créé simultané soit **transformé en** non simultané.

1.5 - FMS EST UN SYSTEME DE FICHIERS A LA CARTE

1.5.1 - Présentation

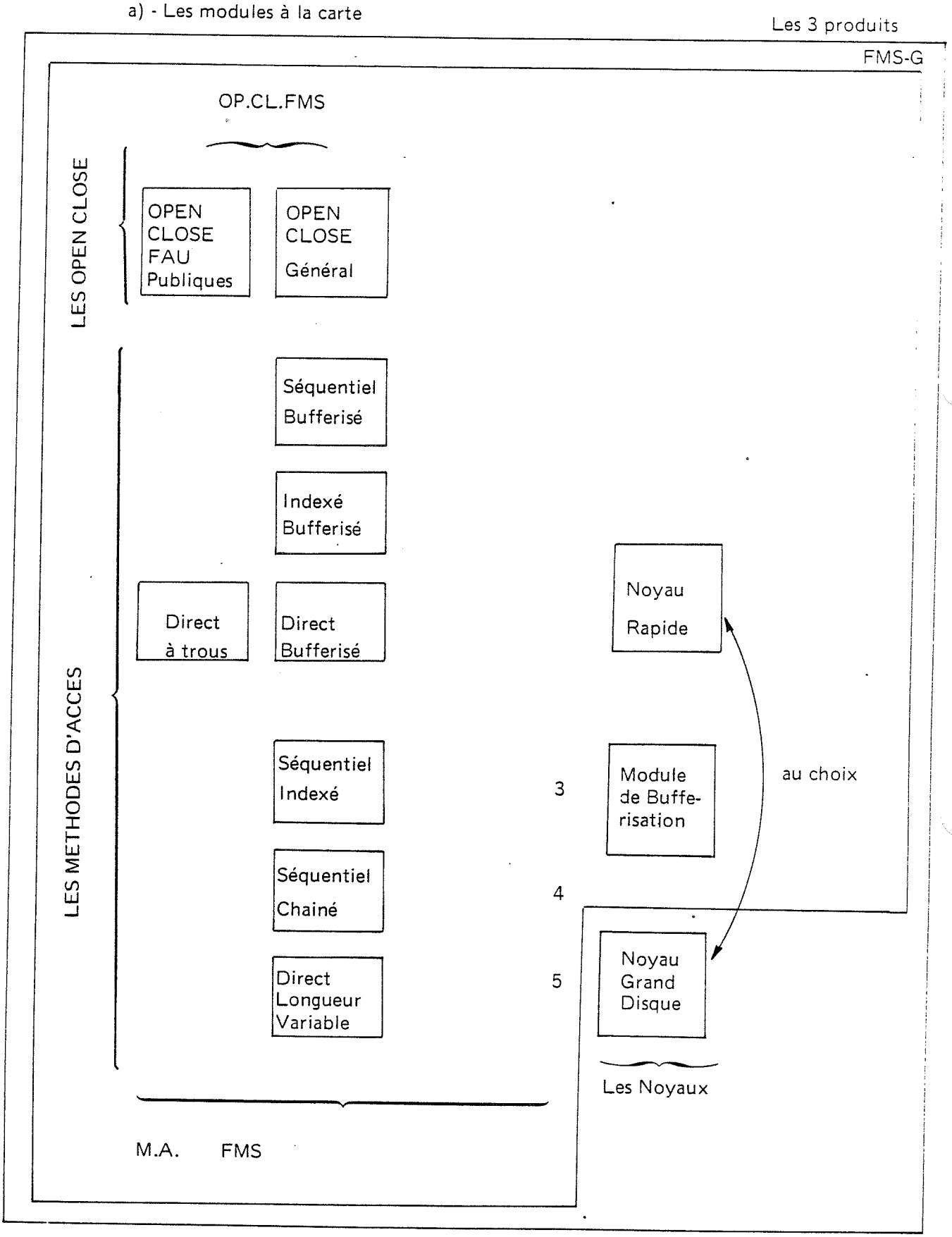
D'une part, FMS est un système de fichier appartenant au noyau résident du système d'exploitation. Il est donc totalement **résident** en mémoire centrale et pour cela répond à des objectifs de **performance.**
D'autre part, FMS possède une structure **modulaire** telle, que **l'utilisateur** peut en bénéficier. C'est-à-dire que FMS lui est proposé sous la forme d'un «jeux de construction» (des bibliothèques de modules), et qu'il peut donc choisir les modules, c'est-à-dire les outils dont il a effectivement besoin, pour construire une version opérationnelle de FMS. Les possibilités de **choix** se situent au niveau des outils suivants :

— Des **méthodes d'accès**
Parmi l'ensemble des méthodes d'accès décrites précédemment, l'utilisateur choisi celle (s) qu'il désire utiliser pour construire une version opérationnelle de FMS. Il est également possible à un utilisateur averti d'implémenter ses propres méthodes d'accès dans FMS sous condition d'avoir pris connaissance du fonctionnement et des interfaces internes du système de fichiers.

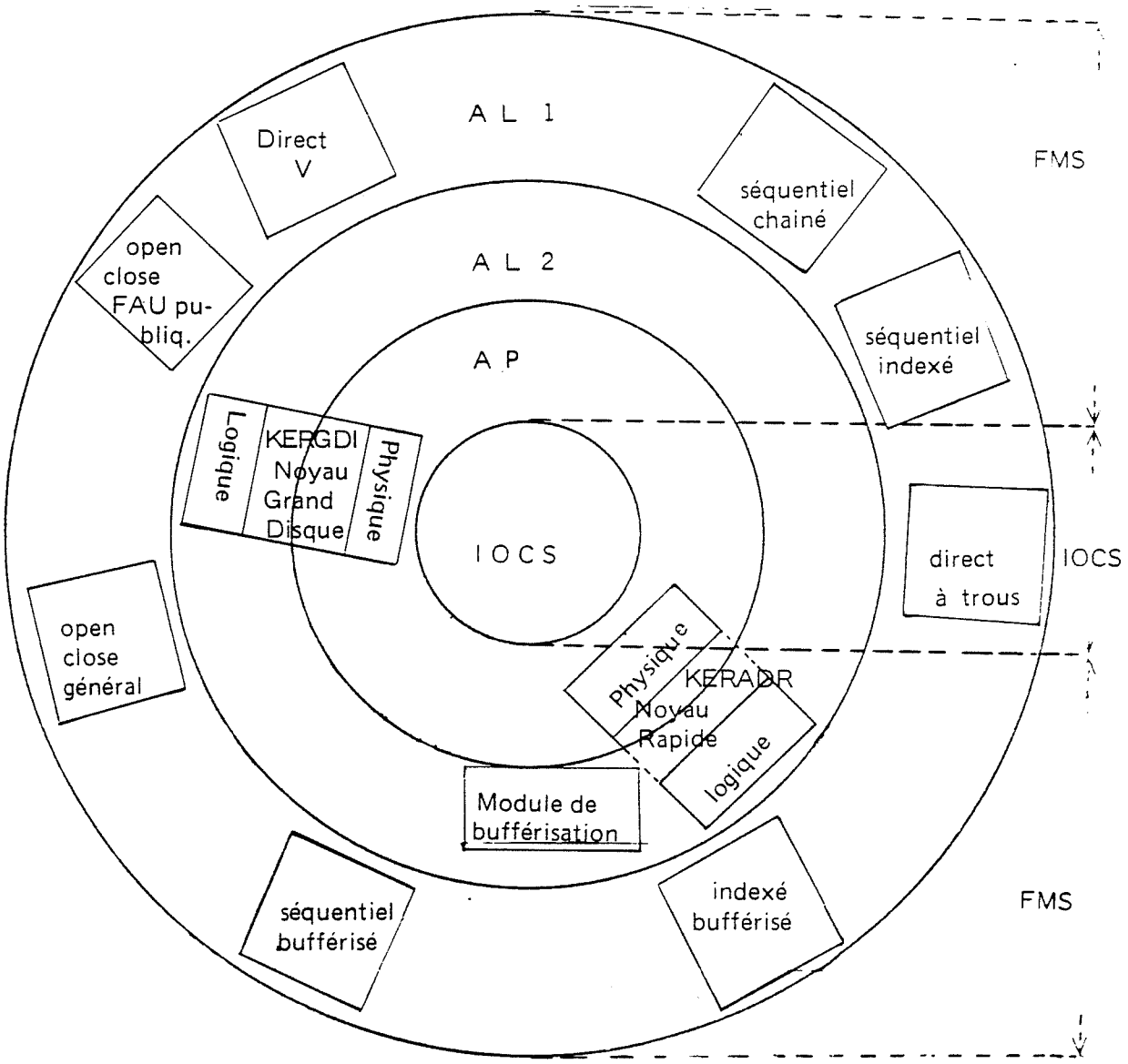
La génération de FMS c'est-à-dire la construction d'une version opérationnelle et son intégration dans un système d'exploitation est rendue fiable et très simple par l'utilisation d'un outil de génération automatique : **GFMS16.**
L'utilisateur paramètre son système de fichier à l'aide de quelques Macro définition, conversationnelles ou préparées dans un fichier.

1.5.2 - Structure du système de fichier

a) - Les modules à la carte



b) - Architecture générale du système de fichiers



Ce modèle situe les différents modules à la carte dans l'architecture générale de FMS

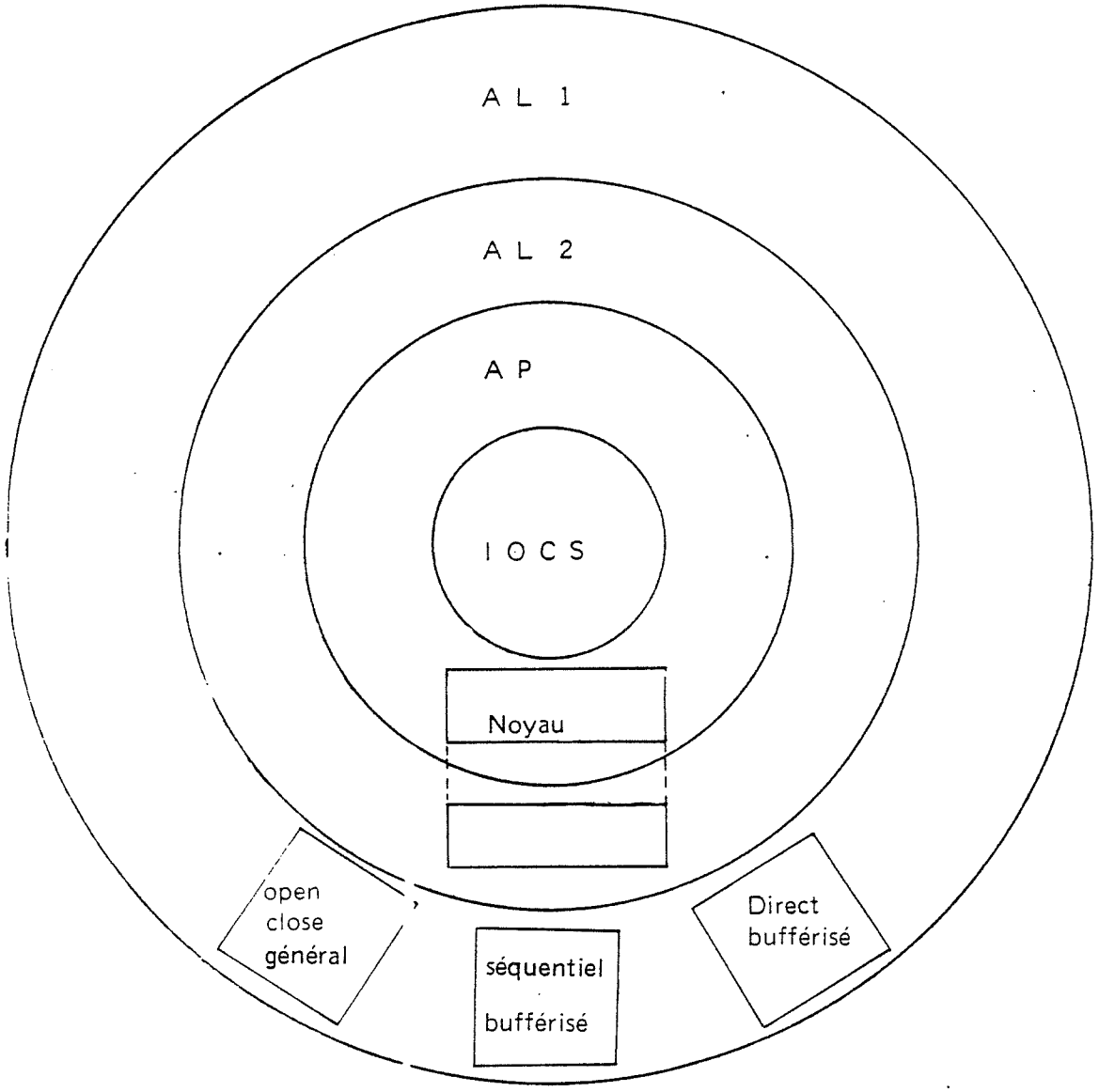
IOCS est le moniteur d'entrées-sorties.

- AP : un anneau gérant dynamiquement l'allocation, la désallocation et l'accès à l'espace disque physique.
- AL2 : est un anneau gérant l'accès logique à des fichiers non structurés.
- AL1 : est un anneau gérant des unités d'accès logique à des fichiers structurés en articles selon les méthodes d'accès.

La réalisation pratique d'une **nouvelle méthode d'accès** s'effectue en rajoutant **un module** dans l'anneau le plus externe : AL1.

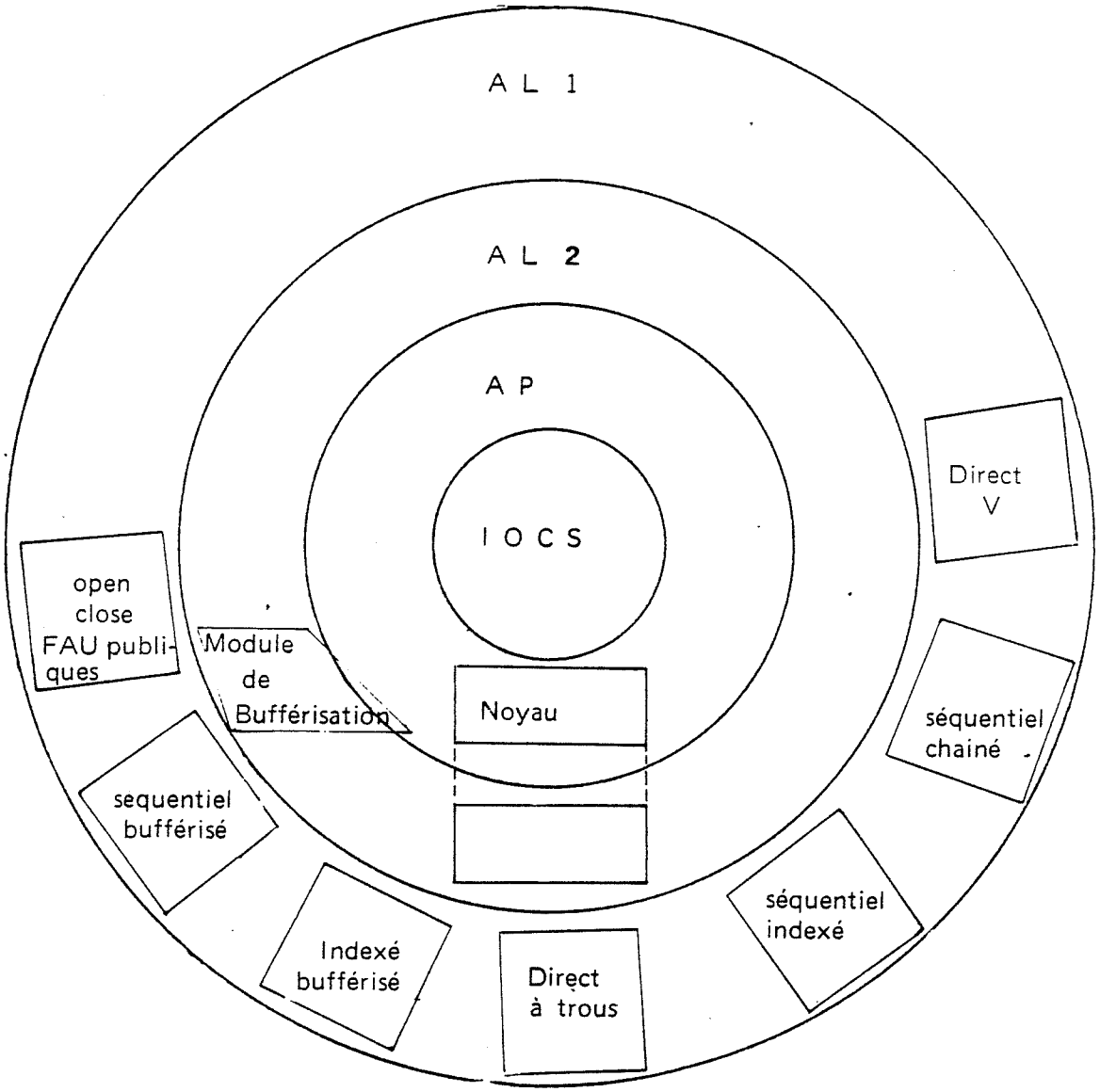
c) - Exemple de systèmes de fichiers opérationnels

1) - FMS configuration minimale utilisable



Le système de fichier possède 2 méthodes d'accès - Séquentiel - Direct.

2) - FMS configuration maximale actuelle.



Le système de fichier possède 5 méthodes d'accès :
Séquentiel - Indexé - Direct à trous - Séquentiel Indexé - Séquentiel chainé.

Il possède toutes les options de performance.

1.6 - FUP LE COMPLEMENT INDISPENSABLE DE FMS

FUP (File Utilities Programs) est un ensemble de processeurs fonctionnant dans un contexte de traitement par lots.
Les objectifs généraux de FUP peuvent se décrire de la façon suivante :

FUP complément indispensable de FMS

FUP a pour but de réaliser en différé les fonctions complémentaires et indispensables, de celles réalisées par FMS en tant que composant de la partie résidente d'un système d'exploitation.
Exemple de fonction complémentaire et indispensable : initialisation d'un support physique géré par FMS.

FUP outil de fiabilité

FUP a pour but de contrôler et faciliter l'utilisation des fichiers disque gérés par FMS. En particulier certains utilitaires permettent de «nettoyer» ou récupérer un support physique géré par FMS sur lequel est intervenu un incident.
D'autres utilitaires réalisent l'édition d'états fournissant :

- la liste des fichiers situés sur un support, ainsi que des indications sur leur organisation logique et leurs caractéristiques physiques.
- la description de l'état d'allocation d'un espace disque.

FUP outil d'archivage et de transfert d'informations

FUP permet d'archiver (et de restituer) un fichier sur (ou à partir d'une) bande magnétique ou bande papier.
FUP permet d'archiver (et de restituer) un support disque sur (ou à partir d'une)bande magnétique.
FUP est également un outil général de transfert d'informations. Il prend en compte les supports suivants : carte perforée, bande perforée, bande magnétique, support disque, imprimante, télé-imprimeur fichier disque, article d'un fichier disque.

2 — INTRODUCTION

2 — INTRODUCTION

Le lecteur trouvera dans ce chapitre la définition des principales notions utile à une connaissance technique du fonctionnement externe de FMS, ainsi que la terminologie spécifique du produit.

2.1 - STRUCTURE LOGIQUE DE L'INFORMATION

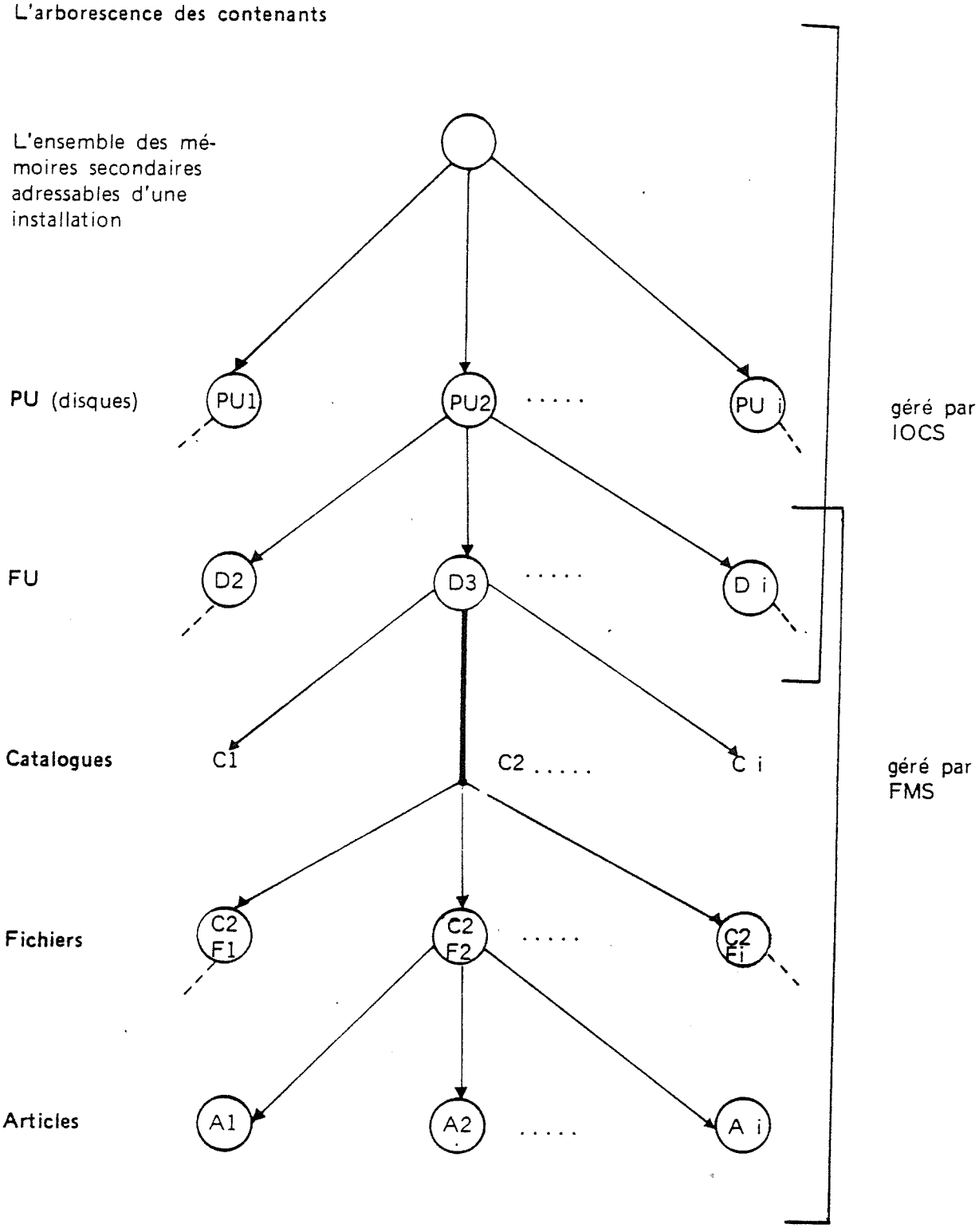
2.1.1 - Les classes d'informations

L'utilisateur traite des **informations** et FMS gère des **contenants,** tous les deux s'entendent pour **identifier** ce qui représente pour l'un des paquets d'informations et pour l'autre des boîtes. FMS propose à l'utilisateur de structurer ses informations de façon **arborescente.** En effet cette structure doit correspondre à celle des contenants physiques ou logiques, réels ou virtuels gérés par IOCS et FMS.

Les différents contenants. De l'élément de base à celui qui les contient tous :
- l'Article (enregistrement logique)
- le Fichier
- le Catalogue
- l'Unité Fonctionnelle (FU)
- l'Unité Physique (PU)
- l'ensemble des mémoires secondaires adressables d'une installation (Disques).

L'utilisateur doit donc structurer ses informations en articles qu'il regroupe en fichiers. Les noms des fichiers doivent comporter un nom de catalogue à fin de distinguer logiquement des groupes de fichiers entre eux. Ces fichiers sont stockés sur des Unités Fonctionnelles. Une ou plusieurs FU correspondent à une Unité Physique. Les PU constituent à leur tour, l'ensemble des mémoires secondaires adressables d'une installation.

L'arborescence des contenants



a) L'article

Un article est **l'unité d'information logique** (Enregistrement Logique). L'article est pour l'utilisateur le plus petit élément d'information identifiable et accessible directement. Pour FMS c'est le plus petit récipient, contenant un vecteur de n mots contigus et identifié par un nom. un numéro selon la méthode d'accès utilisée.
L'article est l'élément **générique** d'un fichier.
C'est à la création du fichier ou de l'article que **l'utilisateur** définit l'appartenance d'un article à un fichier.
Un article ou une portion d'article est l'unité logique de **transfert** d'informations entre la mémoire centrale et une mémoire secondaire.

b) **Le fichier**

Un **Fichier** est le plus grand contenant logique réel d'informations.
Il porte un nom de 6 caractères ASCII définit par l'utilisateur à la **création** du fichier. A ce nom de fichier est associé le nom d'un catalogue dont la définition est donnée au paragraphe suivant : c).
Un fichier est composé d'un ou plusieurs **articles** selon la méthode d'accès. Pour certains fichiers le nombre d'articles est **fixe** et défini à la création du fichier (Séquentiel, Direct), pour d'autres il est **variable** pendant la vie du fichier (Indexé, Séquentiel Indexé). FMS ne fournit aucune requête pour modifier l'appartenance d'un article à un fichier. L'utilisateur ne peut que lire le contenu d'un article et le mettre dans un autre article qu'il crée ou qui existe déjà.
C'est l'utilisateur qui définit à la **création** d'un fichier, sur quelle Unité Fonctionnelle FMS devra créer le fichier. FMS ne fournit pas de requête pour modifier l'appartenance d'un fichier à une FU, ce service est cependant rendu par **FUP**.
L'utilisateur regroupe dans une FU des fichiers liés entre eux par des critères, logiques ou quantitatifs (taille des fichiers ou performance de l'allocation dynamique) propres à une application.
FMS fournit des requêtes pour créer et détruire des fichiers, demander l'accès à un fichier permanent, modifier les attributs d'un fichier (ex : un nom de fichier).

c) **Le catalogue**

Un catalogue est un contenant logique "virtuel". C'est-à-dire qu'il n'existe pas de récipient contenant tous les fichiers d'un même catalogue.
Lorsque l'utilisateur crée un fichier **Permanent** il associe au nom du fichier un **Mot de passe PUBlic** (PUBW) de 2 caractères ASCII.

On appellera globalement nom de fichier la concaténation : FNAM (6 caract. — PUBW (2 carac.)

exemples	:	RBOSD	: S
		FMSSTD	: S
		FMSE	: S

L'objectif de ce mot de passe public est de permettre à des utilisateurs programmant indépendemment les uns des autres d'assurer la non homonymie des noms des fichiers permanents d'une installation. Exemples de mots de passe public les initiales des programmeurs ou les numéros des groupes de travail.
Lorsqu'un utilisateur stocke des fichiers permanents sur un support physique qu'il est seul à utiliser, le mot de passe public peut jouer le rôle d'un nom de catalogue.

Exemples :

" : S " le catalogue des fichiers contenant le software de base (systèmes d'exploitation, Processeurs...). Le nom de catalogue ' : S ' ne doit pas être utilisé pour des fichiers utilisateur)

" IM " Le catalogue des fichiers contenant des images mémoire.

FMS ne fournit pas de requête pour ne modifier que le nom d'un catalogue. Il permet cependant de renommer le nom d'un fichier permanent (FNAM-PUBW).

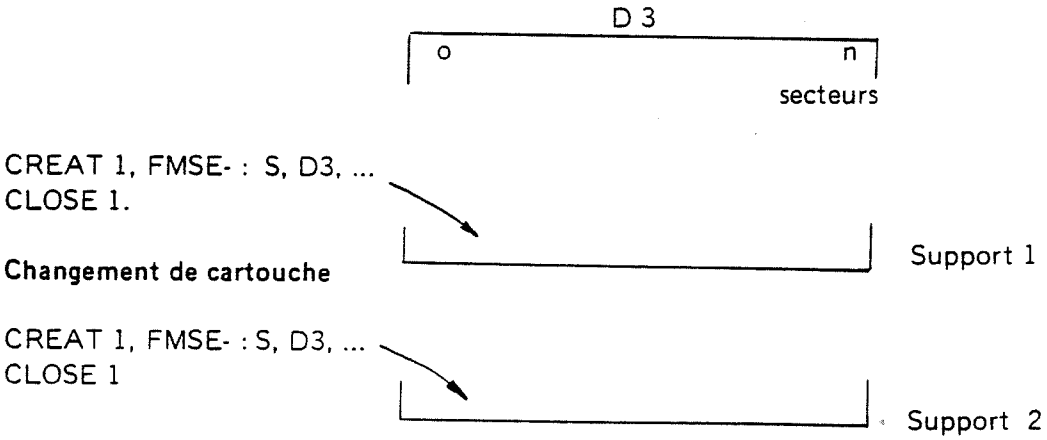
d) Unité Fonctionnelle (FU-Support) et Unité Physique

Les notions d'Unité Fonctionnelle (FU) et d'Unité Physique (PU) sont définies dans le manuel de références d'IOCS.
A la génération d'un système d'exploitation l'utilisateur définit, parmi l'ensemble des FU adressant les disques de l'installation, quel sous-ensemble sera géré par FMS.
Lorsqu'une FU correspond à un disque fixe on peut dire qu'une FU **est une partie** (tout ou partie) **d'un disque.** Lorsqu'une FU correspond à un support disque amovible, on dira qu'une FU est **un espace d'adressage** de **N supports** disque. Dans ce cas il faudra donc savoir distinguer si besoin est, les notions :

FU = espace d'adressage
Support = partie physique d'un disque

Un utilisateur pourra donc créer deux fichiers de même nom sur la même FU, mais à des instants différents et sur des supports différents.

Exemple :



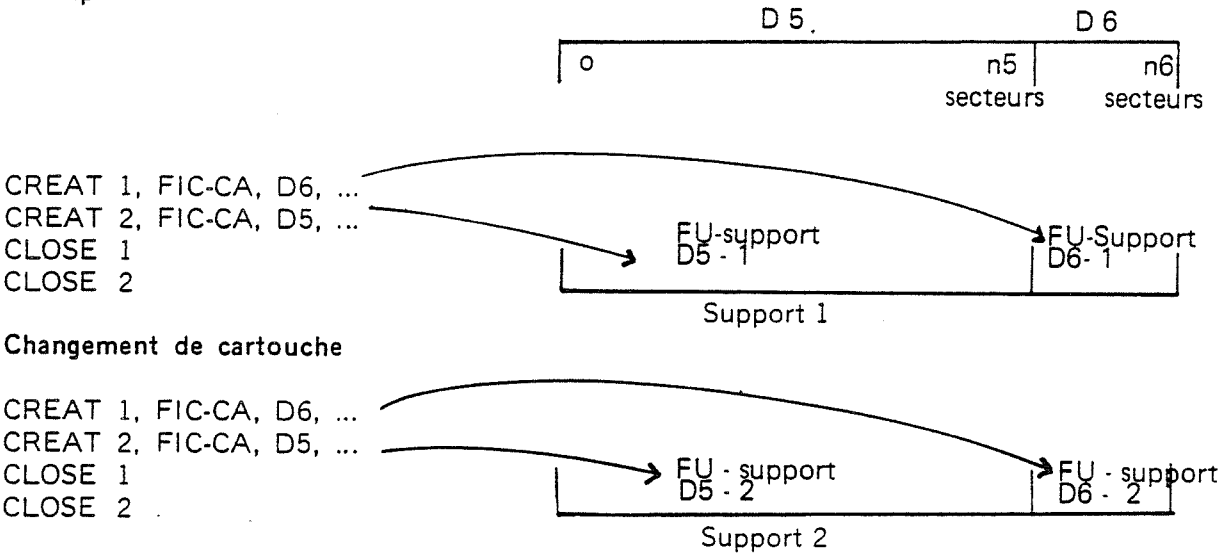
Définition 1 : Unité physique et Unité fonctionnelle sont des objets de même nature, c'est-à-dire un périphérique ou une partie de périphérique. Techniquement une FU est définie à l'aide des informations suivantes : — adresse du 1er cylindre — Longueur en nombre de cylindres.

Définition 2 : Une FU est un espace d'adressage dans une PU.

La FU-support

Au sens courant un support amovible est par exemple une cartouche, or IOCS permet de faire correspondre à la partie amovible plusieurs Unités Fonctionnelles.

Exemple :

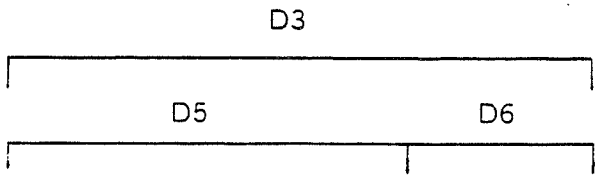


Changement de cartouche

Définition :
Une **FU-support** (ou plus simplement un **support**) est une **partie** d'un **support** physique disque **adressable** par une **Unité Fonctionnelle.**

Remarques Importantes
A la génération d'un système d'exploitation il est possible de définir au niveau de IOCS (GENIO) des FU dites de recouvrement.

Exemple :



Une telle structure à pour objectif l'archivage de la totalité d'un support physique à l'aide de IOCS.

1er Cas Support Fixe
Règle importante : La FU de recouvrement ne doit être utilisée **que** par IOCS.

2e Cas Support amovible
Règle importante : La FU de recouvrement et les FU recouvertes peuvent toutes être utilisées par FMS **mais** sur des **supports différents et** sur des périodes de **temps** qui ne se **recouvrent pas.**

Règle pratique (sans contrôle de la part de FMS si l'on n'utilise pas la gestion de vulume).
Il est interdit d'ouvrir simultanément une FU de recouvrement et les FU recouvertes correspondantes.

Ce qui est réalisé par :

```
OPEN    OLD    1, FIC-C1, D3, ...
OPEN    OLD    2, FIC-C2, D5, ...

CLOSE   1
CLOSE   2
```

Des conseils pour la définition de l'ensemble des Unités Fonctionnelles d'une installation, c'est-à-dire de l'ensemble des espaces disques adressables, sont donnés dans le manuel d'utilisation de FMS.

Le choix porte sur :
- le nombre de FU
- la taille des FU
- l'emplacement des FU par rapport aux Unités Physiques.

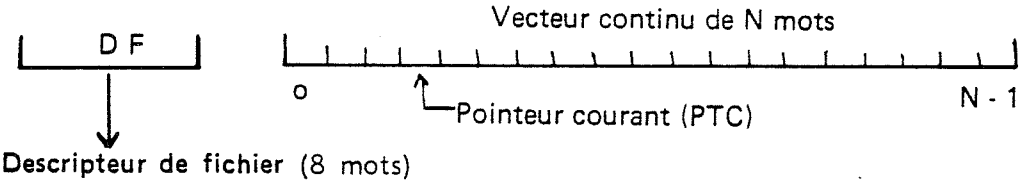
des critères de choix :
- Volume global d'informations.
- Volume des différents fichiers.
- Regroupement logique de fichiers.
- Dynamique des fichiers (variations de taille, volatilité).
- Rapidité au sens nombre d'accès disque.
- Taux d'occupation d'un support.
- Partage des supports entre les utilisateurs.

2.1.2 - Structure de base d'un fichier

a) **Description**

Tous les fichiers gérés par FMS possèdent la même structure de base, qui correspond au fait que tout fichier est un récipient d'informations identifié par un nom.

Structure de base d'un fichier :



Descripteur de fichier (8 mots)

- **FNAM** (2 mots) Nom du fichier.
- **PUBW** (1 mot) Mot de passe public.
- **EMA/SID** (4 bits) numéro d'organisation logique.
  ex : Séquentiel, Indexé...
- **S** (1 bit) type de simultanéité.
- **W** (1 bit) protection écriture.
- **OFI** (1 bit) type d'organisation physique.
  0 : Séquentielle 1 : Directe
- **Adresses physiques** d'implantation du fichier sur le support (2,5 mots).
- **Deux informations système** (TART') (NART') dont le produit définit en nombre de mots, la place physique qui doit être allouée au fichier à sa **création.**
  ex : Fichier Séquentiel TART' = 0 NART' = 1

L'objectif principal du précédent schéma est d'indiquer que tout fichier, quelle que soit sa structure d'articles est un récipient contenant un vecteur de N mots logiquement contigus.

b) Les primitives du noyau

Cette structure de base est géré par le **noyau** de FMS, qui fournit de façon **interne** pour les modules gérant les méthodes d'accès un ensemble de primitives :

1°) **Au niveau Fichier :**

- **Créer** et **détruire** un récipient (Temporaires ou Permanents)
- **Créer** et **détruire** un DF disque (Permanents)
- **Lire** un DF en mémoire centrale (Ouvrir un Permanent)
- **Mise à jour** d'un DF sur disque (Permanents : Fermer, Renommer...)

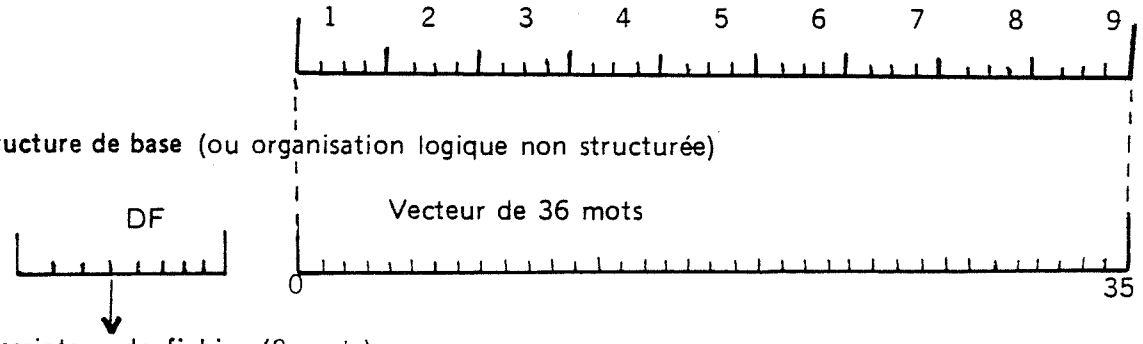
2°) **Au niveau Entrées-Sorties** (Temporaires ou Permanents).

- **Sélectionner** dans le fichier le mot d'adresse n (Positionnement absolu du pointeur courant (PTC))
- **Lire** les n mots suivants par rapport à PTC
- **Écrire** les n mots suivants par rapport à PTC, avec désallocation et allocation de place
- **Réécrire** les n mots suivants par rapport à PTC sans modifier les mots adjacents.
- **Saut arrière** de n mots par rapport à PTC
- **Saut avant** de n mots par rapport à PTC
- **Saut à la fin** (positionnement absolu du PTC sur le dernier mot du fichier).

C'est sur cette structure de base, encore appelée organisation logique non structurée, que les méthodes d'accès constituent et gèrent les différentes structures en articles des fichiers ; appelées **Organisations Logiques.**

Exemple : Un fichier direct de nom FIC-C1 constitué de 9 articles de 4 mots.

Organisation Logique n° 2 : Direct



Structure de base (ou organisation logique non structurée)

Descripteur de fichier (8 mots)

- FNAM = " FIC "
- PUBW = " C1 "
- EMA/SID = 2 (4 bits) : Direct
- S = 1 (Fichier Permanent simultané)
- W = 0 (Fichier protégé en écriture)
- OFI = 1 Fichier à organisation **physique** directe
- Adresses Physiques
- TART' = 4
- NART' = 9

La principale conséquence pour l'utilisateur du fait que tous les fichiers gérés par FMS possèdent la même structure de base est que :
Tout fichier quelle que soit son organisation logique (méthode d'accès) est accessible de façon séquentielle (mode séquentiel pur statique. Voir chapitre 5 5.2).

Exemple : Un usager peut utiliser un fichier Direct pour obtenir fonctionnellement un **"fichier séquentiel" borné et à allocation statique.**

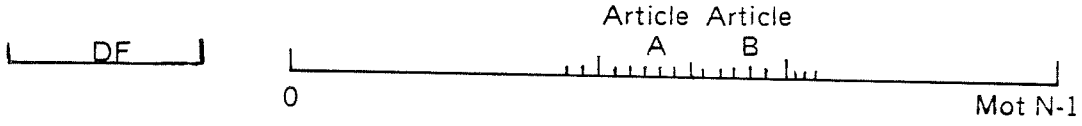
c) Organisation Logique, Organisation Physique

La structure de base des fichiers, encore appelée organisation **logique** non structurée est :

1o) une organisation **logique par rapport** à l'organisation physique décrite au chapitre 2.2.3 et qui indique qu'un récipient fichier est physiquement implanté sur un support dans une chaine de granules, et que les granules alloués à un fichier ne sont pas forcément contigus sur le support.

2o) une organisation **physique par rapport** aux organisations logiques gérées par les méthodes d'accès. En effet dans le récipient fichier, **2 mots adjacents sont physiquement** (au sens adresse disque) **contigus, aux changements de granules près.**
Ainsi l'on dira par exemple dans la description des méthodes d'accès que 2 articles sont physiquement adjacents s'ils occupent des positions adjacentes dans le vecteur de mots d'un récipient fichier.

Structure de base d'un fichier : 2 articles dits physiquement adjacents.

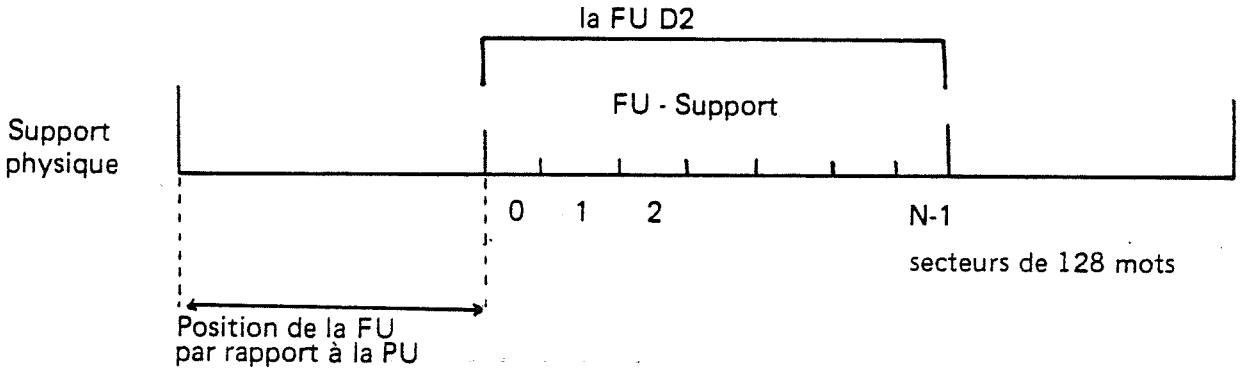


2.2 - GESTION DES SUPPORTS PHYSIQUES (sans gestion de volume)

2.2.1 - Structure des supports physiques

Comme il a été décrit au chapitre précédent (2.1.1), le récipient FU dans lequel FMS stocke des fichiers est en fait une FU-Support, c'est-à-dire la partie d'un support adressable par une FU. Le récipient FU-Support est géré par IOCS et défini par l'utilisateur à la génération du Système d'exploitation.

Schéma Physique (exemple)



Exemple : la FU D2 est définie par :
- Elle adresse le plateau fixe.
- Elle débute au 40e cylindre.
- Elle occupe 200 cylindres, soit N secteurs de 128 mots.

IOCS réalise sur un support disque des échanges d'un **nombre entier de secteurs.** Il est cependant possible :
- en lecture de ne pas charger en mémoire centrale la dernière partie du dernier secteur d'un échange.
- en écriture de compléter physiquement par des zéros, la dernière partie du dernier secteur d'un échange.

Pour qu'un support soit **utilisable** par FMS il faut y écrire un ensemble d'informations systèmes initiales, à l'aide de FUP (FUINI de FUP 4). Ces informations permettent à FMS de gérer l'allocation dynamique du support avec des paramètres **spécifiques** pour chaque support (voir chapitre suivant 2.2.2).

Remarque : La notion de FU-Support est ici utile, puisqu'il ne faut pas initialiser toutes les FU d'une installation, mais tous les **Supports** (FU-Support) fixes ou amovibles, qui devront être gérés par FMS. En particulier il faut initialiser le contenu des nouvelles cartouches que l'on met en service dans une installation.

D'une façon générale les supports disques sont **réutilisables.** Pour FMS réutiliser un support signifie écrire par FUP (FUINI) de nouvelles informations systèmes initiales. Cette opération a évidemment pour conséquence de " détruire " les anciens fichiers du support.

Un support disque est **simultanément partageable** au sens ou des usagers différents peuvent simultanément stocker des fichiers sur le même support.

a) Des FU identiques, support portable

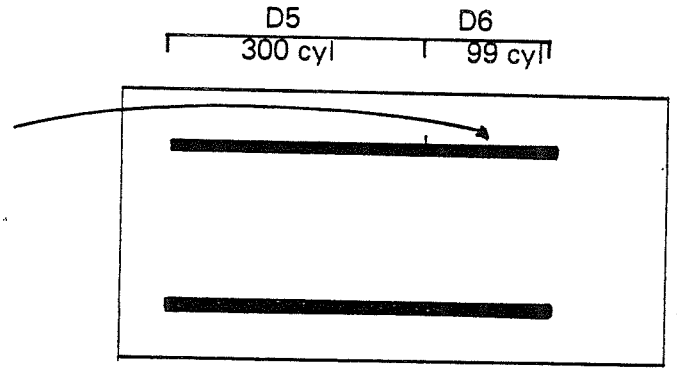
Un support disque amovible, est **portable** du point de vue de FMS si l'utilisateur peut le placer sur des **FU identiques,** et utiliser les fichiers qui y sont stockés.

Du point de vue de FMS deux FU sont identiques si et seulement si :
— Elles ont physiquement la même adresse.
— Les systèmes d'exploitation dans lesquels elles ont été définies savent pour chacunes, gérer le même nombre de granules.

Remarque : A la génération d'un système d'exploitation l'utilisateur définit, non seulement quelles FU seront gérées par FMS, mais également et pour chaque FU, le nombre maximum de granules à gérer.
Le Manuel d'Utilisation de FMS indique comment utiliser l'outil automatique de génération : GFMS16.

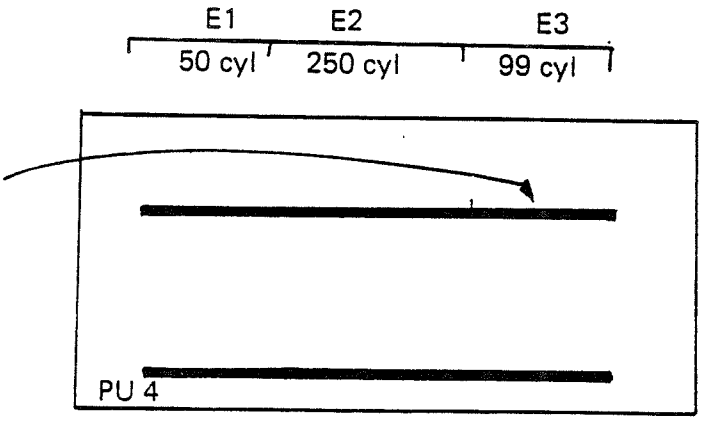
Exemple :
Installation nº 1 (1 PU)
GFMS16 : % CONFMS FU = D6 NBGRAN = 99

FU adressant la cartouche de l'unité 1



Installation nº 2 (4 PU)
GFMS16 : % CONFMS FU = E3 NBGRAN = 99

FU adressant la cartouche de l'unité 4



La FU D6 de l'installation nº 1 est identique à la FU E3 de l'installation nº 2 car :
— Elles ont toutes les deux la même adresse
  — Elles adressent le disque amovible (cartouche)
  — Elles débutent au cylindre d'adresse 301
  — Elles occupent 99 cylindres
— Les systèmes d'exploitation des installations nº 1 et nº 2 savent tous les deux gérer 99 granules, sur respectivement les FU D6 et E3 . (99 granules de 6 K mots).

b) **Un support autonome**

Un support (FU-Support) est **autonome** car :

— Il contient toutes les informations système permettant de gérer les fichiers permanents qui s'y trouvent et d'en créer de nouveaux temporaires ou permanents.

— Il ne contient que ses propres informations système. Par exemple FMS **ne** gère **pas** sur un support privilégié la liste de tous les fichiers de l'installation. Sur chaque support se trouve la liste des fichiers **du** support.

2.2.2 - Principe d'allocation dynamique

a) **Principe**

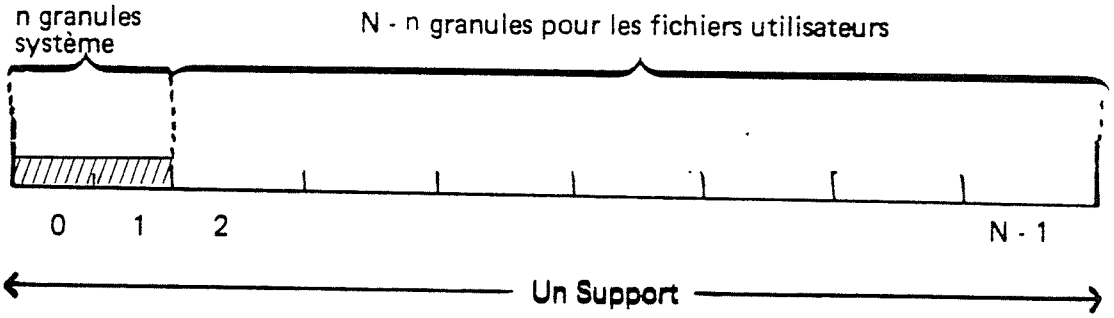
FMS se propose de résoudre complètement pour l'utilisateur, la gestion de la place physique occupée par les fichiers sur un support adressable (disque).
L'utilisateur devra seulement demander à FMS de créer un fichier en spécifiant son **nom**, ses **attributs logiques** tels que organisation Logique (Séquentiel etc...), Temporaire ou Permanent etc..., et le **support** (FU-Support) sur lequel FMS devra créer le fichier. FMS se chargera alors automatiquement d'allouer la place nécessaire au fichier sur le support désigné par l'utilisateur.

Règles :

- Un support (FU-Support) est divisé en N portions de tailles égales appelées des **granules.**
- Sur des supports différents, l'utilisateur peut définir des Tailles de granules différentes.
- **Le granule** représente donc l'unité d'allocation de mémoire secondaire adressable (disque). (Taille standard = 2 K.mots).
- Allouer de la mémoire disque à un fichier signifie trouver sur le support, des granules libres et indiquer qu'ils appartiennent au fichier.
- FMS possède donc de façon interne un outil appelé **allocateur de mémoire**, qui pour chaque support, réalise le choix de granules libres (allocation) ou qui rend libre des granules (désallocation) pour les autres fichiers du support.
- **Table d'Allocation de Granules.** L'allocateur de mémoire utilise sur chaque support, une chaîne de bits appelée **Table d'Allocation de Granules** (TAG) indiquant l'état d'allocation des granules du support.
- Soit TG la taille des granules du support, on associe de façon bijective au bit de rang i de la TAG, le granule d'adresse (i — 1) TG.
- **Une chaîne de granules.** Pour indiquer l'appartenance des granules à un fichier, on établit dans la partie système des granules un chaînage amont-aval. **Un fichier** est donc physiquement constitué **d'une chaîne de granules.**
- Lorsque l'on " détruit " un fichier il suffit de parcourir la chaîne des granules du fichier et d'indiquer dans la TAG que les bits correspondant sont libres.
- **La Table des Fichiers.** Chaque support contient également la Table des Fichiers (TF) **Permanents** stockés sur ce support. Elle est constituée d'un ensemble de descripteurs de Fichiers (DF) décrivant les noms et les attributs des fichiers.

b) Description physique d'un support géré par FMS

1o) Structure d'un support (FU-Support)



Un support est divisé en N granules. L'utilisateur paramètre sur **chaque** support à l'aide de FUP (FUINI de FUP 4) la **taille du granule** (TG).

— **Valeurs limites** de TG (en secteurs)

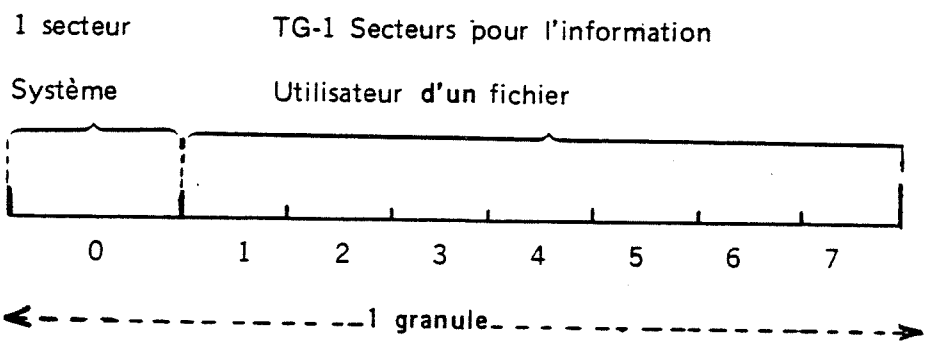
$$3 \leqslant TG \leqslant 256$$

— **Règle standard**

TG ⩽ Longueur maximum d'un échange autorisé par IOCS sur le périphérique concerné.
ex : 6 K mots sur un disque à bras mobiles.

Le Manuel d'Utilisation de FMS-.. indique, des conseils et les règles à respecter pour définir sur chaque support la taille du granule.
Taille standard : TG = 2 K mots

2o) **Structure d'un granule**



L'exemple ci-dessus décrit un granule de 1 K mots (8 secteurs).
Dans tout granule le 1er secteur est réservé au système, et contient (au moins) dans ses 2 premiers mots, si le granule est effectivement alloué à un fichier ou à la chaîne des granules systèmes :

— LAM = Ligature amont : adresse du granule précédent dans la chaîne
— LAV = Ligature aval : adresse du granule suivant dans la chaîne

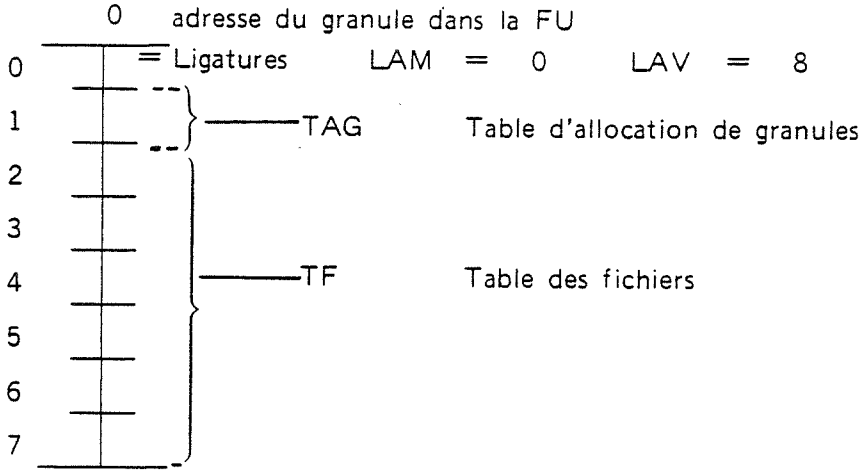
L'adresse d'un granule est : l'adresse dans la FU du premier secteur du granule. (adresse IOCS).

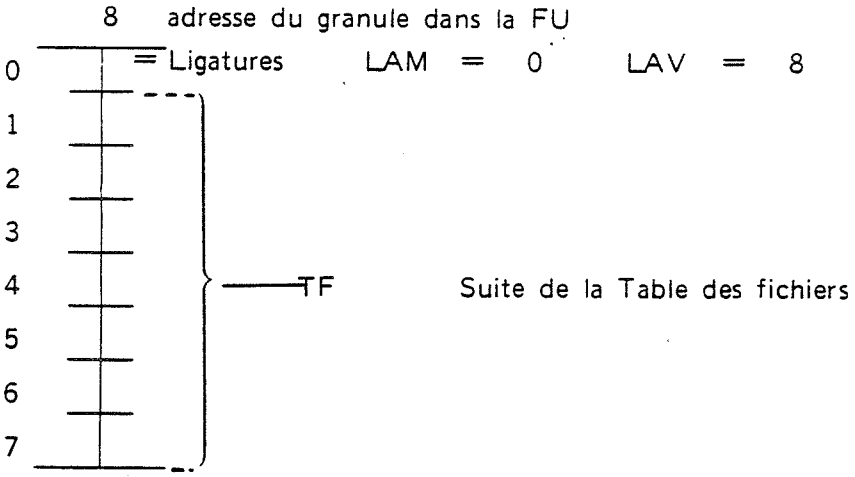
3o) Structure de FIFI le "fichier" Système

FIFI est constitué des **n premiers** granules du support.

Ces granules sont donc réservés au système pour stocker toutes les informations qui permettent à FMS de gérer ce support.

Exemple : FIFI = 2 granules de 1 K mots (8 secteurs)



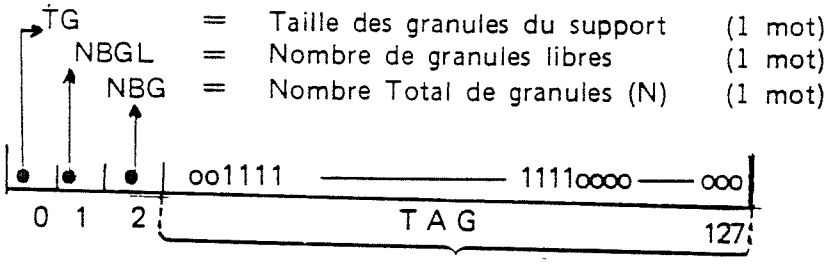


Remarque :

La commande FUINI de FUP4 (Indice d'évolution supérieur ou égal à 40) crée un fichier de nom FIFI — : S permettant d'accéder à FIFI par FMS (OPEN OLD, READ, CLOSE). Cela permet donc de lire en séquentiel, la Table d'Allocation de Granules (TAG) et la Table des Fichiers (TF).

Le fichier FIFI — : S est Direct, Simultané et protégé en écriture. Il possède une organisation physique séquentielle (OFI = 0). La Taille des articles est égale à la Taille secteur et le nombre d'articles égal au nombre de secteurs utiles de FIFI.

4º) **Structure du Secteur 1 de FIFI**



125 mots = Chaîne de bits d'occupation des granules

Le schéma ci-dessus décrit la chaîne de bits après initialisation du support (FUINI).

2 - 16

Règle :

bit i = 0 ⇔ granule occupé

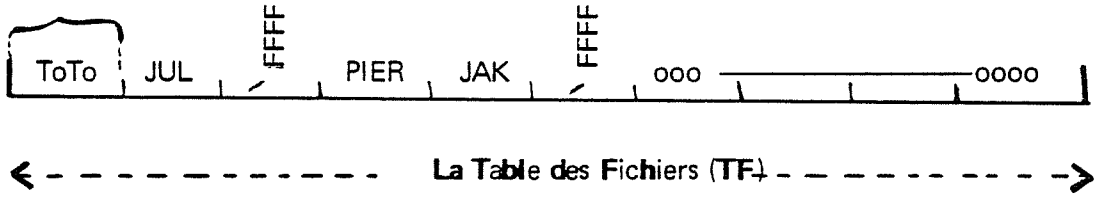
- Les 2 premiers bits " sont " occupés : ils correspondent aux 2 granules systèmes occupés par FIFI.
- Les NBG-2 bits suivants " sont " libres et utilisables pour des fichiers utilisateurs.
- Les bits suivants 2000-NBG sont marqués occupés car ils ne correspondent plus à l'espace physique du support.

5º) **Structure de la Table des Fichiers (TF)**

On peut considérer que toute la Table des Fichiers est un vecteur de mots, bien qu'elle occupe plusieurs granules.
Elle permet de stocker les **Descripteurs** des **Fichiers** (DF = 8 mots) **permanents** du support.

Exemple : Le support contient 4 fichiers Permanents.

Un descripteur de fichier (8 mots)

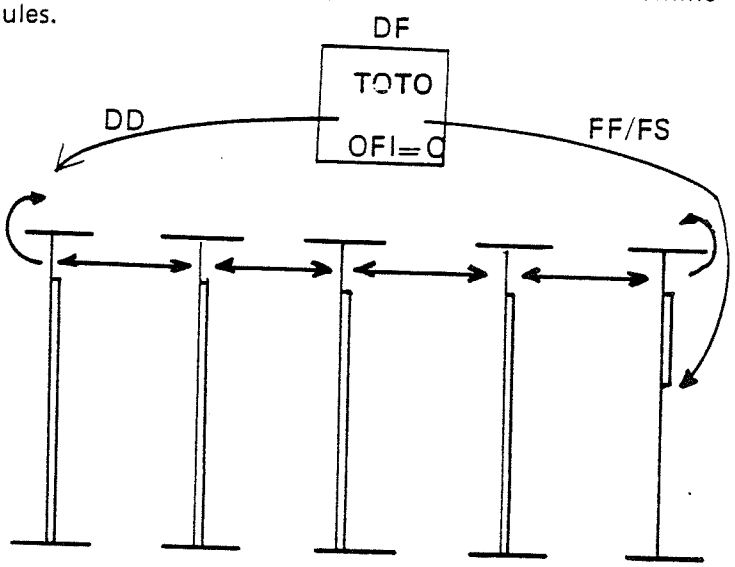


- La fin du contenu dans TF est définie par la présence des zéros. A l'initialisation du support (FUINI). Toute la TF est mise à zéro.
- Lorsque FMS détruit un fichier permanent, il met dans le premier mot du descripteur correspondant la valeur : 'FFFF .
- A l'initialisation d'un support, l'utilisateur spécifie (optionnel) le **nombre maximal** de fichiers **permanents** qu'il compte créer sur le support. Ce paramètre permet de définir le nombre n de granules système constituant FIFI.
- Un secteur de TF peut contenir 16 descripteurs de fichier.

2.2.3. - Organisation physique d'un fichier

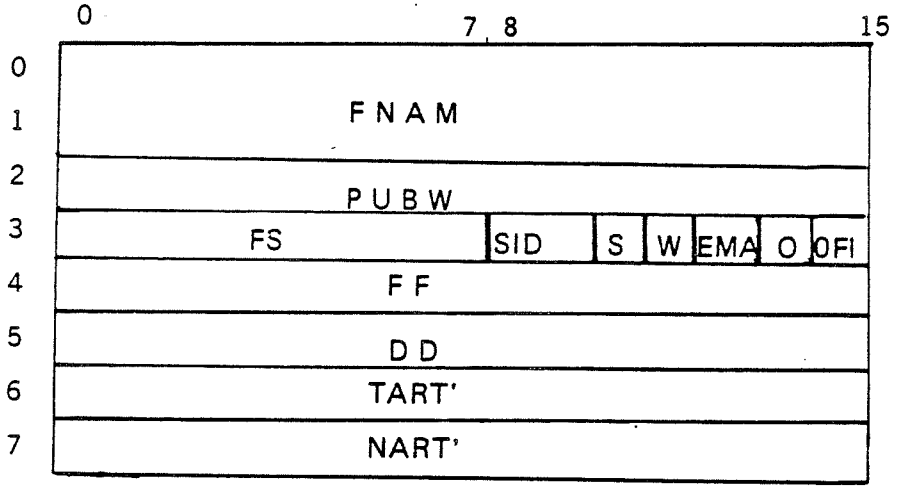
Un fichier est physiquement constitué de 2 parties :
— Un descripteur de fichier
— Une chaîne de granules

Exemple : Organisation Physique Séquentielle d'un fichier nommé TOTO et occupant 5 granules.



a) **Le Descripteur d'un fichier sur disque**

Structure d'un DF d'un fichier **sur disque,** donc d'un fichier **permanent.**



FNAM — est le nom du fichier codé en Radix 40 (3 caractères par mots)
PUBW — est le nom du catalogue codé en Radix 40
EMA/SID — est le numéro de l'organisation logique du fichier (4 bits)
ex : Séquentiel Indexé n° 3

EMA	SID
00	11

S — est le type de Permanent
S = 1 ⇔ Fichier Permanent Simultané

W — bit de protection écriture
W = 1 ⇔ Fichier accessible en écriture

OFI — est le numéro de l'organisation physique
OFI = 0 ⇒ Organisation physique séquentielle
OFI = 1 ⇒ Organisation physique Directe

DD — adresse du premier granule du fichier (adresse secteur)

FF/FS — adresse physique de fin de fichier
FF adresse secteur
FS déplacement dans le secteur. (l'adresse du mot suivant)

TART'
NART' — (NART') . (TART') = taille du fichier à sa création (en mots).

Séquentiel	TART'	=	0	NART'	=	1
Indexé	TART'	=	4	NART'	=	f (NART)
Direct	TART'	=	TART/2	NART'	=	NART
Séquentiel Indexé	TART'	=	LP/2	NART'	=	NP
Séquentiel Chaîné	TART'	=	LP/2	NART'	=	NP
Direct V	TART'	=	LIX/2	NART'	=	f (NART)

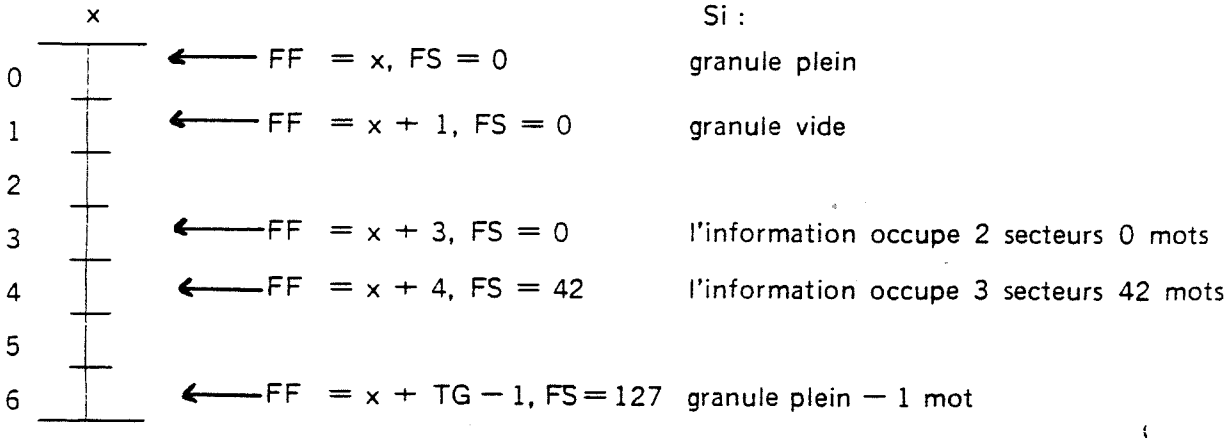
b) **La chaîne de granules**

Conventions :

— LAM — La ligature amont d'un granule adresse le granule précédent dans la chaîne.
Dans le premier granule d'une chaîne LAM adresse ce même granule.

— LAV — La ligature aval d'un granule adresse le granule suivant dans la chaîne.
Dans le dernier granule d'une chaîne LAV adresse ce même granule.

— FF/FS — Soit x l'adresse du granule



— Dans un granule l'information de l'utilisateur est contigue au mot près.
2 mots adjacents dans le fichier au sens structure de base, sont physiquement adjacents.

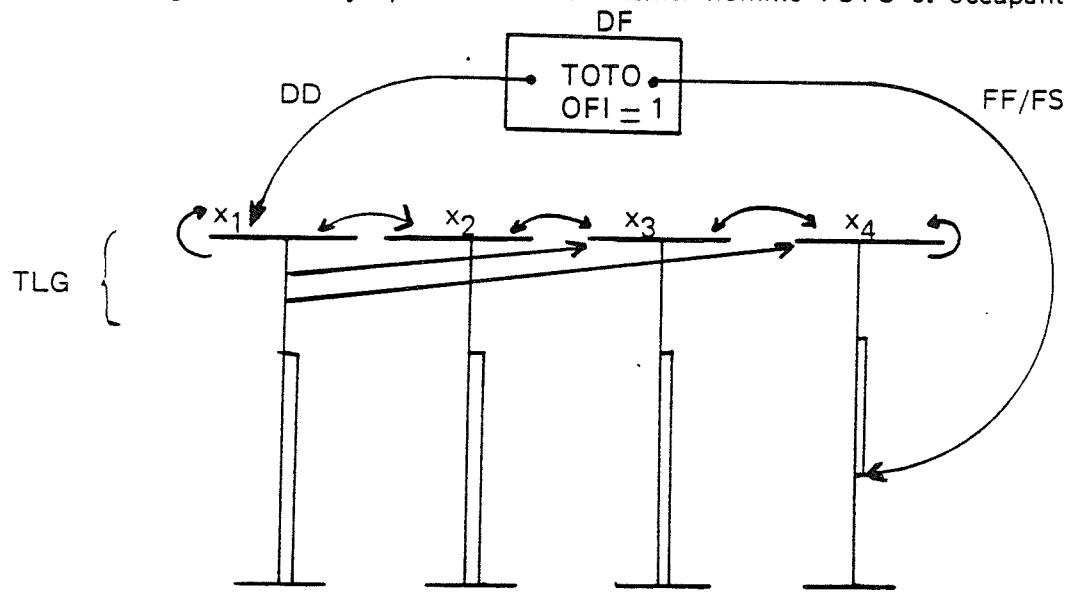
c) **Organisation physique séquentielle**

On appelle **organisation physique séquentielle** la structure décrite par le schéma situé au début de ce paragraphe 2.2.3 et qui est constituée :
— des 2 pointeurs physiques DD, FF/FS appartenant au Descripteur du Fichier
— de l'ensemble des ligatures AMONT AVAL de tous les granules d'une chaîne.

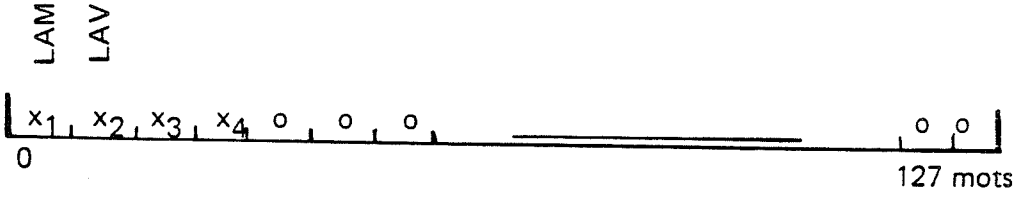
d) **Organisation physique directe**

On appelle **organisation physique directe** la structure décrite par le schéma suivant et qui est constituée :
— des informations systèmes constituant l'organisation physique **séquentielle**
— de la Table des adresses de tous les granules de la chaîne (TLG) et qui se trouve située dans le premier secteur du premier granule de la chaîne.

Exemple : Organisation Physique **directe** d'un fichier nommé TOTO et occupant 4 granules



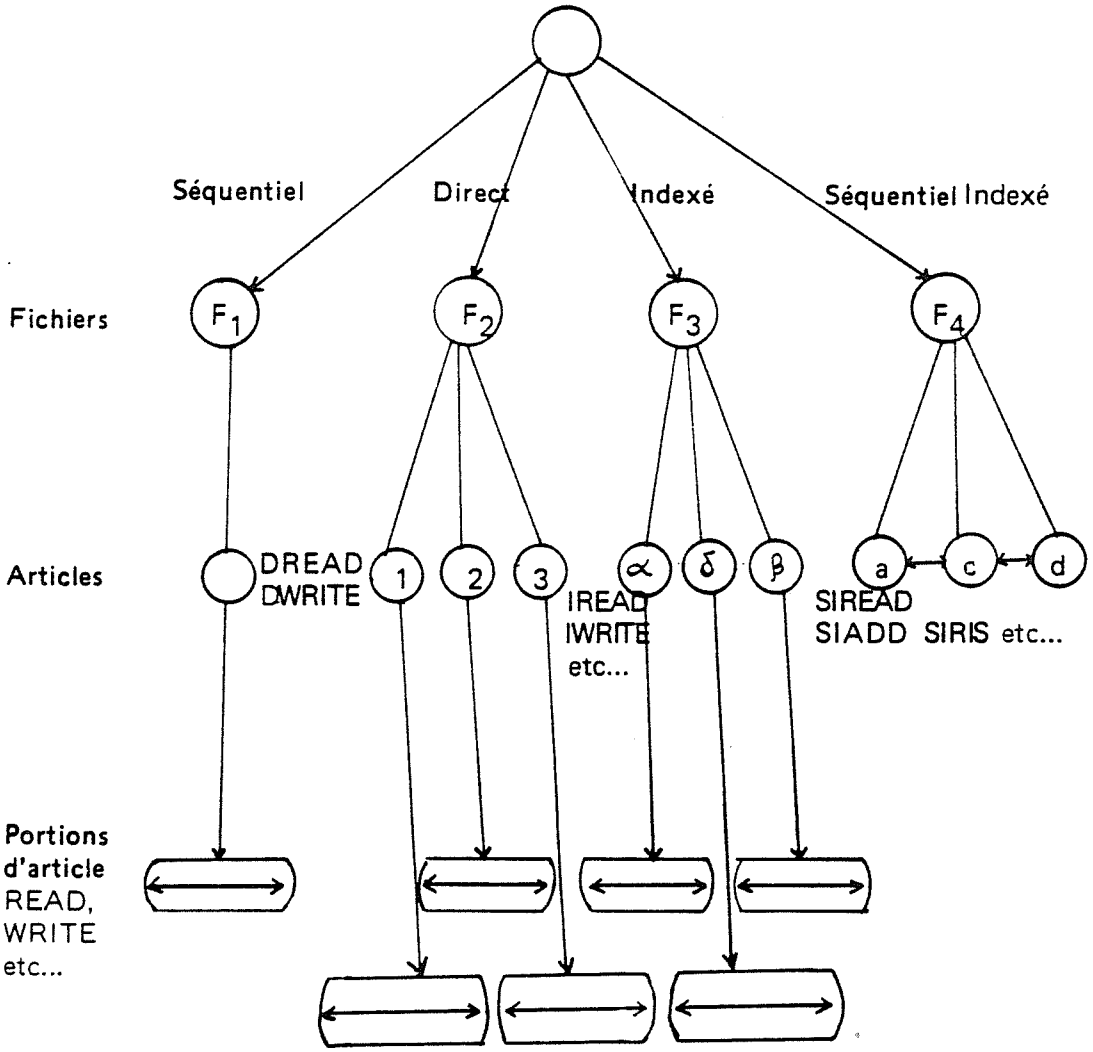
Description de la TLG correspondante:



2.3 - PRINCIPALES MÉTHODES D'ACCES

2.3.1 - Schéma Général

Quelle que soit la méthode d'accès, FMS gère trois niveaux d'information
— fichiers
— articles
— portions d'article
Ces informations sont accessibles différemment selon les méthodes d'accès.

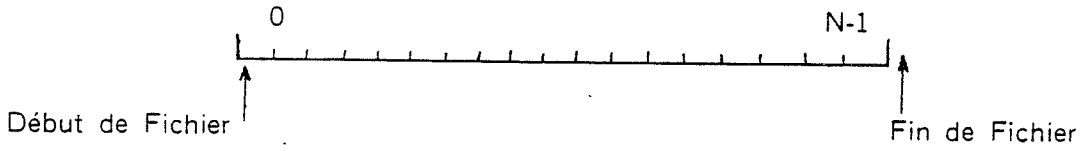


2.3.2 - La méthode d'accès : Séquentiel

a) **Organisation séquentielle**

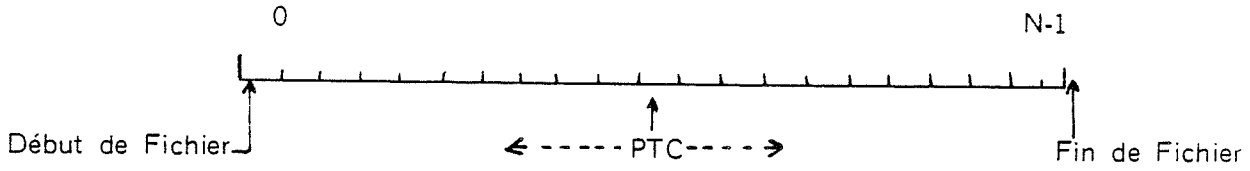
Les informations sont stockées les unes à la suite des autres dans le fichier selon l'ordre chronologique.
Le fichier est constitué d'un seul article, c'est-à-dire que l'ensemble des mots stockés n'est pas structuré.
Le fichier est de taille variable, grâce à une allocation et une désallocation dynamiques par rapport à la fin du fichier : c'est la dernière écriture qui détermine la fin du fichier qui n'est limité que par la taille du support physique (la FU)

Schéma d'un fichier séquentiel (N mots)



b) **Accès Séquentiel**

FMS permet l'accès à un article de façon séquentielle par portions d'articles
FMS gère un pointeur courant (PTC) qui peut se déplacer en avant ou en arrière en pointant toujours le prochain mot à lire ou à écrire.



Il existe 6 primitives d'accès à une portion d'article.

- READ — (n octets ou jusqu'à tel code d'arrêt) Lecture des n/2 mots suivants PTC
- WRITE — (n octets ou jusqu'à tel code d'arrêt) Écriture des n/2 mots suivants PTC
- REWIND — Saut au début de l'article
- SKEOA — Saut à la fin de l'article
- SKIPB — (n octets) saut arrière de n/2 mots
- SKIPF — (n octets ou jusqu'a tel code d'arrêt) Saut avant de n/2 mots.

Chacune de ces requêtes a pour effet de déplacer le pointeur courant en fonction du nombre de mots effectivement traités.

c) Utilisation

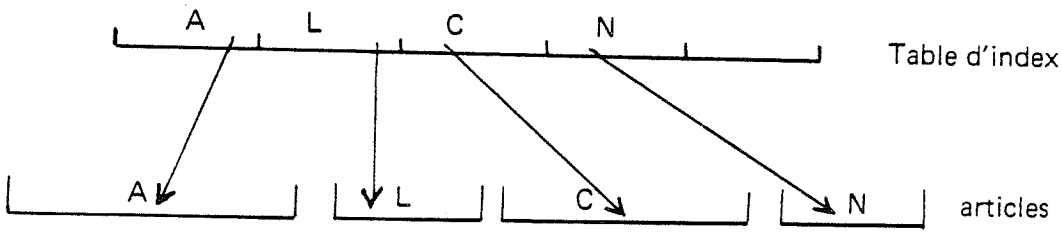
Les fichiers séquentiels de FMS s'utilisent comme des bandes magnétiques.
En raison de leur organisation simple et de leurs possibilités d'accès peu élaborées ces fichiers sont utilisés pour des sauvegardes, des archivages, des listes chronologiques.

2.3.3 - La méthode d'accès Indexé

a) Organisation

Un fichier indexé est composé d'un ensemble d'articles de taille variable identifiés par nom. Afin de permettre un accès direct à l'article selon son nom, le fichier contient une table d'index associant aux noms des articles, leur adresse (index) dans le fichier.
Le nom de l'article est de longueur fixe (8 caractères). Les articles sont stockés selon la chronologie de leur création : il n'y a pas de tri par nom.

Schéma d'un fichier indexé



Le fichier peut s'étendre dynamiquement avec allocation dynamique à la création d'un nouvel article.

b) Accès

L'indexé fournit cinq primitives permettant de manipuler les articles nommés

- IREAD (nom de l'article) lecture de l'article
- IWRITE (nom et contenu de l'article) Création d'un nouvel article
- ISUP (nom de l'article) Suppression de l'article nommé
- IRWRITE (nom et contenu de l'article) Réécriture du contenu de l'article nommé
- IRNAM (nom de l'article) Changement de nom de l'article précédemment sélectionné
- IRTIX Chargement de la table d'index en Mémoire Centrale

L'indexé permet un accès séquentiel aux articles par portions c'articles.

c) Utilisation

Ce type de fichier est intéressant dans le cas où l'on dispose d'un nombre assez faible d'articles de longueur variable identifiés par nom et où les suppressions sont assez rares.

C'est en particulier le cas de nombreux fichiers système : bibliothèque, fichiers de programmes segmentés.

2.3.4 - La méthode d'accès Direct

a) **Organisation directe**

Un fichier direct est constitué d'articles de longueurs égales, identifiés par numéro.
Le nombre total d'articles est fixé dès la création du fichier et toute la place nécessaire au fichier est alors réservée sur le support : c'est de l'allocation statique.

Schéma d'un fichier direct

b) **Accès direct**

- L'accès à un article se fait directement à l'aide de son numéro, FMS en déduit son adresse dans le fichier puis son adresse physique sur le support.
  Les deux primitives d'accès à un article sont :
  - DREAD (nº article) Lecture de l'article
  - DWRITE (nº article) Reécriture de l'article
- Le Direct permet un accès séquentiel aux articles par portions d'articles.
- En considérant qu'un fichier direct est constitué d'un seul article le Direct permet d'obtenir fonctionnellement un fichier " séquentiel " borné à allocation statique.

c) **Utilisation**

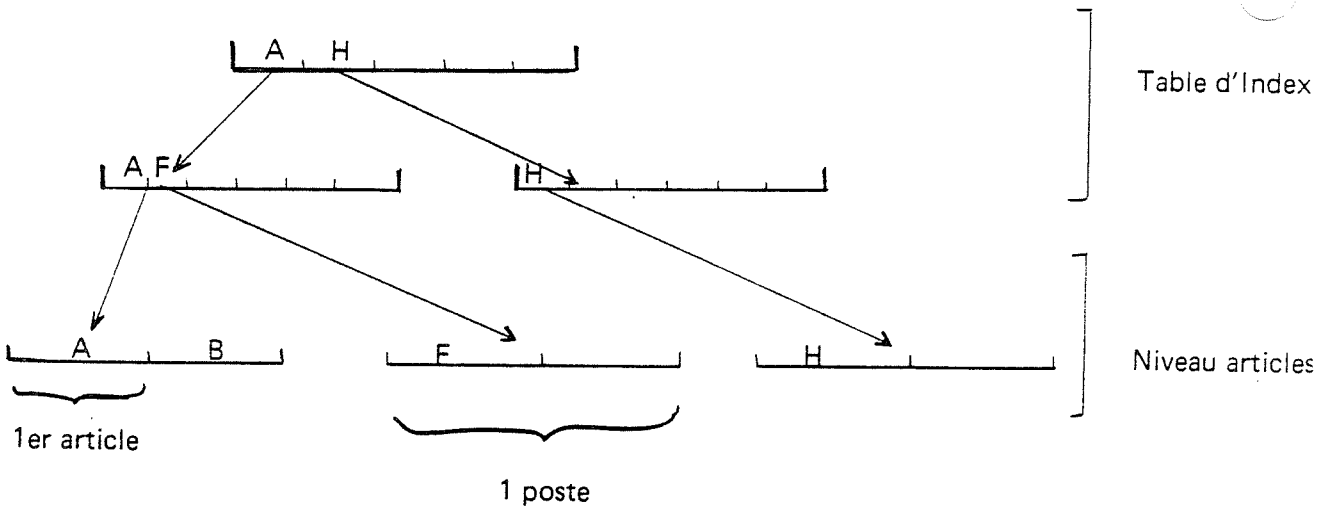
L'accès direct est l'accès le plus rapide à des informations réparties de façon aléatoire dans un fichier.
Il est à utiliser dès que les identificateurs d'articles sont des " codes pleins ", c'est-à-dire dès que la suite des codes est continue.
exemple : un numéro de 1 à N.

2.3.5 - La méthode d'accès Séquentiel Indexé

a) Organisation

Un fichier Séquentiel Indexé est constitué d'un ensemble d'articles, tous de même taille, identifiés par un nom.
Comme dans un fichier Indexé, les articles sont repérés à l'aide d'une table d'index.
Cette méthode d'accès a pour objectif de gérer 100 000 articles. Pour rendre rapide la recherche d'un nom dans la table d'index, cette table est éclatée en plusieurs niveaux, chaque niveau adressant le niveau suivant.
Le nombre de niveaux de la table d'index est dynamique, il varie dans les deux sens croissant décroissant, selon le nombre d'articles réellement présents dans le fichier.

Structure arborescente d'un fichier Séquentiel Indexé



Les articles sont triés par ordre croissant de nom. Pour FMS un nom d'article est un nombre binaire absolu de N bits. La taille d'un nom d'article est une constante de chaque fichier. Elle est définie par l'utilisateur à la création du fichier.
Lors des créations et suppressions d'articles, la Table d'Index est automatiquement réorganisée en conséquence. La place du fichier sur le support physique est réservée à sa création par FMS . A l'intérieur de cet espace, la méthode d'accès gère dynamiquement les postes devant contenir des articles ou des parties de table d'index.
Les organisations logiques et physiques d'un fichier Séquentiel Indexé ne se dégradent jamais. Il n'est donc pas nécessaire de gérer des zones de débordement ce qui nécessite des réorganisations complètes et longues du fichier.

b) Accès

L'organisation Séquentiel Indexé permet deux types d'accès à l'information
— l'accès direct à l'article à l'aide de son nom
— l'accès séquentiel permettant d'obtenir successivement la liste des articles triés selon leur nom.

Les primitives d'accès peuvent être réparties en 3 classes :
— les primitives d'accès par nom :
  SIREAD (nom d'article) lecture directe d'un article
  SIADD (nom d'article, contenu de l'article) insertion d'un nouvel article
— les primitives touchant un article précédemment sélectionné
  SISUP suppression de l'article courant
  SIWRIT (contenu de l'article) reécriture de l'article courant
— la primitive d'accès séquentiel
  SIRIS (nombre d'articles à sauter en avant ou en arrière) lecture séquentielle du nième article suivant ou précédent.

c) **Utilisation**

La méthode d'accès séquentielle indexé est particulièrement bien adaptée aux fichiers contenant un grand nombre d'articles de même taille identifiés par une clé alphanumérique. Elle a l'avantage d'éviter toute dégradation des temps de réponse quelles que soient les fluctuations du nombre d'articles. De plus, elle évite toute procédure de tri fastidieux puisque les articles sont perpétuellement rangés selon l'ordre de leur clé d'accès.
C'est la structure type des fichiers de gestion, fichiers de personnel, fichier de produits, etc...

2.4 - LE PARTAGE DES ACCES AUX FICHIERS

Le chapitre de Présentation indique comment décrire la notion de fichier en terme de ressource. Un fichier est donc également une ressource partageable entre des usagers.

2.4.1 - La notion d'usager (USR)

La notion d'usager introduite par FMS a deux objectifs :

1°) Permettre à des utilisateurs différents, d'utiliser les services d'un système d'exploitation et plus précisément ceux de FMS sans être obliger de se concerter. C'est-à-dire sans être obligés de respecter de nombreuses règles au moment de la conception et la réalisation des programmes d'application.
Par exemple, lorsque des utilisateurs différents stockent des fichiers permanents sur le même support, ils doivent respecter une règle commune à fin de garantir la non homonymie des noms de fichiers (exemple : PUBW = identificateur de l'utilisateur)

2°) Permettre à des utilisateurs différents de se partager des fichiers en respectant une règle commune simple, définie et contrôlée par FMS
En particulier il semble naturel de dire, que des utilisateurs qui ne se connaissent pas, ne réaliseront des traitements cohérents, que si FMS leur interdit d'écrire en même temps dans le même fichier.

Pour FMS un usager constitue un ensemble software cohérent vis-à-vis des requêtes d'accès aux fichiers.

Chaque usager est donc responsable de sa propre cohérence. FMS assure la cohérence de l'ensemble des usagers qui s'exécutent simultanément sous le même système d'exploitation.
Dans le contexte de Traitement par Lots en particulier avec BOS16 la notion d'usager est définie par le système et correspond à celle de " JOB ".

Dans le contexte Temps Réel en particulier avec RTES16 c'est l'utilisateur qui définit le numéro d'usager de chaque tâche. Il pourra ainsi distinguer les tâches qui sont cohérentes entre elles, de celles qui ont été conçues indépendemment les unes des autres.

Dans le contexte Temps Partagé l'usager correspond normalement à une personne utilisant une console pendant une session de travail.

L'information usager (USR : un numéro de 0 à 255) n'est jamais manipulée par les programmes d'application, mais par les superviseurs.
Lorsqu'un superviseur transmet à FMS la requête d'un utilisateur (programme, tâche) il fournit à FMS en complément d'information, le numéro de l'usager (USR) qui exécute cette requête.

Exemple d'utilisation de la notion d'usager
Lorsqu'un système Temps Réel (BOS16,RTES16) fournit simultanément des services " Foreground " et " Background ", il est indispensable que les " JOB " s'exécutant dans le contexte " Background " aient un numéro d'usager différent de celui des tâches s'exécutant dans le contexte " Foreground ".

Les notions décrites dans la suite de ce paragraphe (2.4) fourniront les exemples dans lesquels intervient la notion d'usager.

2.4.2 - L'identification d'un fichier

a) **Un fichier Permanent**

Un fichier permanent est identifié pour FMS par la concaténation des informations suivantes :

1°) FNAM : un nom composé de 6 caractères ASCII pairs.
caractères autorisés :
- Les lettres A à Z
- les chiffres 0 à 9
- 4 caractères spéciaux (:) , (;) , (<) et le caractère (Nul) (`oo) à la fin du nom seulement

exemple : **valide** : FICH (Nul) (Nul)
invalide : (Nul) FICH (Nul)
invalide : FI (Nul) CH (Nul)

2°) PUBW : un mot de passe public composé de 2 caractères ASCII pairs. Les caractères autorisés sont les mêmes 40 caractères que pour un FNAM et de plus le (Nul) est autorisé quelle que soit sa position.

3°) FU : un numéro de FU désignant le support sur lequel se trouve le fichier.

Règle : Lorsqu'un utilisateur crée un fichier permanent sur un support FMS vérifie qu'il n'existe pas déjà sur **ce support** (FU-Support) un fichier portant le **même nom, même PUBW.**
Autrement dit l'identification d'un fichier 1 est différente de celle d'un fichier 2 quand

$$(FNAM_1 \oplus PUBW_1 \oplus FU_1) \neq (FNAM_2 \oplus PUBW_2 \oplus FU_2)$$

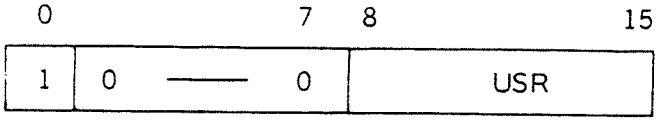
Remarque : Attention aux FU de recouvrement. Voir chapitre 2.1.1, § (d) .

b) **Un fichier Temporaire**

Un fichier Temporaire est identifié pour FMS par la concaténation des informations suivantes :

1°) FNAM : un nom composé de 6 caractères ASCII pairs. Les caractères autorisés sont les mêmes que pour le FNAM d'un permanent.

2°) " PUBW " : un fichier Temporaire ne possède pas de PUBW pour l'utilisateur. Cependant FMS assure la non homonymie des noms des fichiers temporaires créés par les usagers simultanément présents dans le système en donnant au PUBW la valeur suivante :



3º) FU : un numéro de FU désignant le support sur lequel FMS devra créer le fichier.

Règles : Pour un usager (USR) l'identification d'un fichier temporaire 1 est différente de celle d'un fichier temporaire 2 quand :

$$(FNAM_1 \oplus FU_1) \neq (FNAM_2 \oplus FU_2)$$

— FMS permet à un usager d'identifier ses fichiers temporaires comme s'il était le seul utilisateur du système.

Exemple : FMS accepte pour le compte d'usagers différents la séquence suivante :

```
USR 1        : OPEN   NEW  1,   FIC, D3, ...
USR 2        : OPEN   NEW  1,   FIC, D3, ...

USR 2        : CLOSE  1
USR 1        : CLOSE  1
```

c) Utilisation des Unités Symboliques

FMS permet d'utiliser les Unités Symboliques (SU) définies dans le manuel de référence de IOCS et d'en attendre le même service.
Il est donc possible de programmer des accès aux fichiers de façon indépendante de la configuration physique des Unités Fonctionnelles (FU).
Les SU sont utilisables pour les fichiers Temporaires et Permanents.
L'affectation d'un fichier à un support est faite par l'utilisateur à la création du fichier. Pour des fichiers Permanents elle est répétée à chaque ouverture (OPEN OLD).

Exemple :
— Affectation dépendante du support :

```
CREAT  1,   FIC-C1, D3, ...
                    ↑
CLOSE  1            FU
```

— Affectation indépendante du support :

ouverture du fichier permanent créé ci-dessus.

1º) Par commande

```
U1    D3    affectation SU    FU
```

2º) Par programme

```
OPEN OLD  4, FIC-C1, U1, --------
                     ↑
                     SU
```

Remarque : L'utilisation des Unités Symboliques ne change en rien l'identification des fichiers, le système se chargeant de traduire en FU, tout numéro de SU .

2.4.3 - La notion de chemin d'accès à un fichier

a) **Durée de vie d'un fichier**

FMS fournit un ensemble de requêtes pour que l'utilisateur puisse définir la durée de vie de ses fichiers : OPEN NEW, CLOSE, CREAT, DELET, CATAL.

Ces requêtes appartiennent à l'ensemble des requêtes dites du niveau fichier décrites dans le chapitre 4 : L'Entité Fichier (OPEN CLOSE). Elles s'utilisent pour des fichiers Temporaires et/ou Permanents selon la règle suivante :

Fichier Temporaire :

1°) Création de fichier : OPEN NEW

2°) Destruction de fichier : CLOSE ou DELET

Fichier Permanent

1°) Création de fichier : CREAT

2°) Destruction de fichier : DELET

Transformation d'un fichier Temporaire en un fichier Permanent : CATAL

Un fichier est Temporaire ou Permanent par rapport à la durée en machine d'**une** exécution d'**un** usager.

Exemple : Dans le contexte de Traitement par lot (BOS16) :

une exécution d'un usager

```
/ JOB   ← début de l'exécution
/ OPEN NEW 1, FIC, ...
|
|
|
/ CLOSE   1
/ EOJ   ← fin de l'exécution
```

Règles :

- La vie d'un fichier **Temporaire** est limitée par la durée d'exécution d'un usager dans le système d'exploitation.
- Lorsque BOS16 reconnaît la commande /EOJ il envoie à FMS la requête correspondante :
  EOJ (USR)
  qui a pour effet de **détruire** tous les fichiers **Temporaires** que l'usager n'a pas détruit (par erreur, ou abandon du JOB par BOS16 pendant l'exécution du JOB .
- La vie d'un fichier **Permanent** n'est pas limitée par la durée d'exécution d'un usager dans le système.

Exemple : Dans le contexte de traitement par lots (BOS16) : Création du fichier FIC-C3

une exécution d'un usager

```
/ JOB   ← début de l'exécution
/ CREAT   1, FIC-C3...
|
|
|
/ CLOSE   1      fermeture
/ EOJ   ← fin de l'exécution
```

— On appellera **phase de création** d'un fichier Permanent, l'exécution de l'usager limitée par les requêtes [CREAT CLOSE]
— Un fichier Permanent pourra être détruit à l'occasion d'un autre JOB

Exemple :

une exécution d'un usager

```
/ JOB ←— début de l'exécution
/ OPEN OLD 1, FIC-C3, ...    ouverture
|
|
|
/ DELET  1
/ EOJ ←— fin de l'exécution
```

b) **Utilisation d'un fichier**

On dit qu'un fichier est utilisable pour un usager, lorsqu'il est accessible par des requêtes du niveau article et/ou portion d'article effectuant les transferts d'information et/ou les créations, destruction... d'articles, ainsi que par des requêtes du niveau fichier telles que : RENUM RENAM etc...

Règles

— La phase d'utilisation d'un fichier **Temporaire** est limitée par les requêtes :

[OPEN NEW ou CLOSE / DELET]

— Les phases de création et d'utilisation d'un fichier **Temporaire** coïncident exactement.

Exemple :

Phase de création et utilisation

```
OPEN  NEW  1, FIC, ...

READ   1, ...
WRITE  1, ...

CLOSE  1  (ou DELET 1)
```

— Les phases d'utilisation d'un fichier **Permanent** sont de deux types et limitées par les requêtes :

[CREAT / ou / OPEN OLD] ou [CLOSE / ou / DELET]

— Une phase d'utilisation coïncide exactement avec la phase de création d'un fichier **Permanent.**

Exemple :

Phase de création et d'utilisation

```
CREAT  1, FIC-C1, ...

READ   1, ...
WRITE  1, ...

CLOSE  1  <-- fermeture
```

— Une phase d'utilisation existe seule lorsqu'un fichier **Permanent** est ; ouvert, utilisé, puis, fermé ou détruit.

Exemple :

Phase d'utilisation

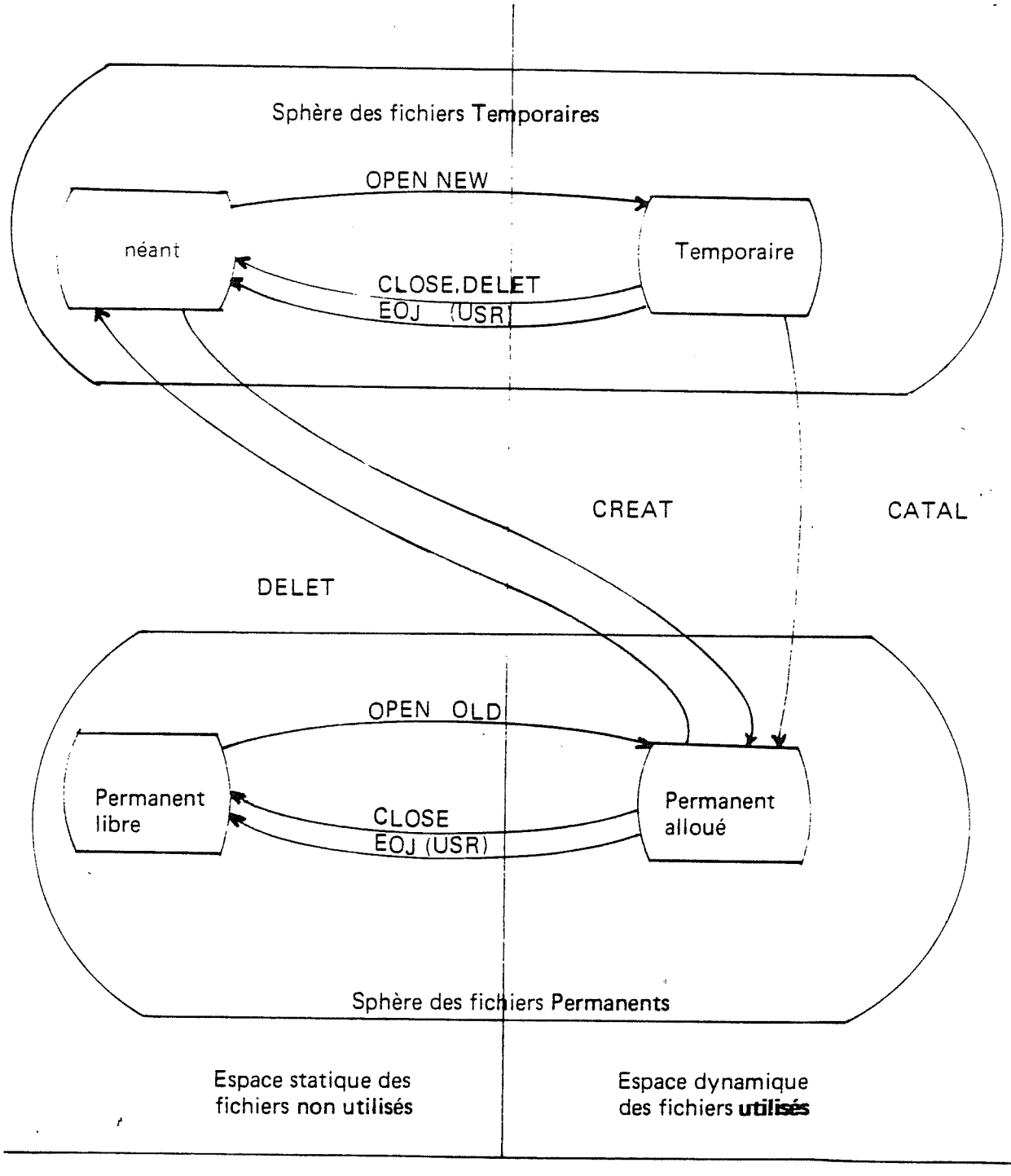
```
OPEN OLD   1, FIC-C1, ...   <-- ouverture

READ       1, ...
WRITE      1, ...

CLOSE      1    (ou DELET   1)
```

c) **Les 4 principaux états d'un fichier**

- Néant : le fichier n'existe pas
- Temporaire : le fichier est créé et utilisé
- Permanent libre : le fichier est créé mais non utilisé
- Permanent alloué : le fichier est utilisé



d) La notion de chemin d'accès (FAU)

La notion de **chemin d'accès** à un fichier est liée à la notion **d'utilisation** d'un fichier, en particulier au sens ou l'une n'existe pas sans l'autre.
FMS fournit des requêtes pour **demander** l'autorisation **d'utiliser** un fichier, (OPEN NEW, OPEN OLD, CREAT). Implicitement FMS **crée** alors un chemin d'accès à ce fichier. Cela se traduit d'un point de vue interne par la création d'une table système appelée : **File Access Unit (FAU).** Pour un usager une FAU se concrétise par un numéro appelé : **File utilisation NUMber (FNUM).**
C'est un usager qui demande à FMS de créer une FAU, un chemin d'accès à un fichier en spécifiant ;
d'une part, le numéro **FNUM** de la **FAU** c'est-à-dire du chemin d'accès au fichier ;
d'autre part, l'identification du fichier (voir chapitre 2.4.2 : L'identification d'un fichier).

Exemple : Création du chemin d'accès numéro 1

```
OPEN OLD   1, FIC-C1, D3, ...
                 \_____/
                    |
                    v
          identification d'un fichier
          Permanent
```

Après une telle ouverture, l'usager pourra accéder au fichier permanent identifié par FIC-C1, D3 (FNAM ⊕ PUBW ⊕ FU) sans répéter à chaque requête le nom du fichier, mais en désignant le numéro du chemin d'accès au fichier.

Exemples :
— Pour lire les n premiers mots du fichier (n = LBU) dans une zone d'échange, située à l'adresse ABU :

```
READ   1,  ABU,  LBU, ...
```

— Pour renommer le fichier :

```
RENAM  1,  JUL — OM, ...
```

la nouvelle identification du fichier est alors : JUL ⊕ OM ⊕ D3

Ce fichier permanent possède maintenant

```
FNAM   =   JUL
PUBW   =   OM
FU     =   D3     le fichier est naturellement toujours sur le support adressé par D3 .
```

— Pour fermer le fichier, et également **détruire** le chemin d'accès numéro 1

```
CLOSE  1, ...
```

Remarque : Après la requête ci-dessus CLOSE 1, ... Toute requête utilisant le chemin d'accès numéro 1, c'est-à-dire la FAU n° 1 est rejetée par FMS qui fournit alors le compte rendu

` 600 A : FAU inexistante (la requête est ineffective).

D'autre part, l'usager a donc laissé sur le support adressé par D3, un fichier permanent de nom JUL — OM .

Règles :

— Un FNUM, numéro de FAU, est choisi par **un** usager sans tenir compte des **autres** usagers.
Plage de validité théorique [0 255]
Cependant chaque système d'exploitation peut apporter des restrictions à cette plage que l'usager devra connaître à fin de ne pas détériorer le fonctionnement du système.

— Lorsqu'à un instant donné, 1 usager tente de créer 2 FAU de même FNUM, FMS lui fournit un compte rendu d'erreur.

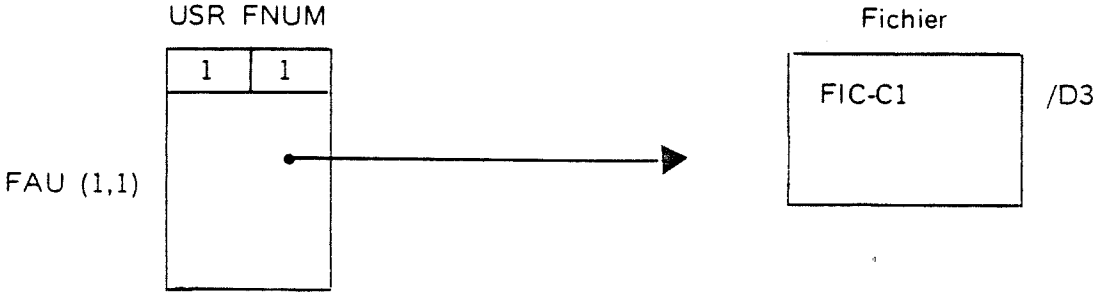
Exemple :

```
OPEN    OLD     1,  FIC -C1, D3, ...
OPEN    OLD     1,  FIC -C2, D3, ...
        sur cette requête FMS  fournit le compte-rendu ` 600 B = FAU existante
        (la requête est ineffective)
CLOSE   1, ...  ←Destruction de la FAU  1
OPEN    OLD     1,  FIC-C2,  D3, ...
        création correcte d'une autre FAU  1
CLOSE   1, ...  ←Destruction de cette deuxieme FAU 1
```

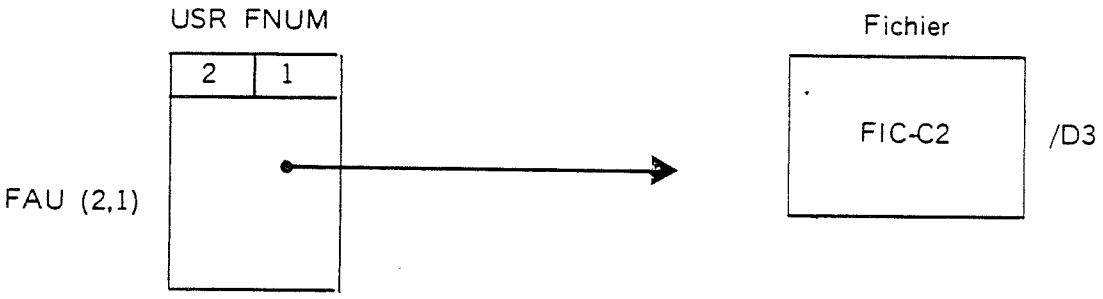
On peut de nouveau constater qu'**un** usager ne doit assurer que sa propre cohérence, et qu'il incombe à FMS de permettre à des usagers qui ne se concertent pas de s'exécuter simultanément sous un même système d'exploitation.
En effet, à l'aide de l'information numéro d'usager (USR) que lui fournit le superviseur du système d'exploitation, FMS distingue les chemins d'accès des différents usagers :

Exemple : Est donc valide à l'instant t, la configuration suivante, dans laquelle deux fichiers permanents sont ouverts et accessibles chacun par **un** chemin d'accès (par **une** FAU). De plus l'usager 1 ne peut utiliser que la FAU (1 , 1) tandis que l'usager 2 ne peut utiliser que la FAU (2 , 1).

Chemin d'accès créé par l'usager 1



Chemin d'accès créé par l'usager 2



Règle : D'un point de vue interne système, un chemin d'accès à un fichier, plus simplement une FAU, est identifiée par la concaténation de :

USR ⊕ FNUM

En résumé on peut voir sur le schéma du paragraphe précédent c) :

Les requêtes qui **créent** une FAU :

- OPEN NEW — crée un fichier Temporaire et une FAU permettant de l'utiliser
- OPEN OLD — ouvre un fichier Permanent et crée une FAU permettant de l'utiliser
- CREAT — crée un fichier Permanent et crée une FAU permettant de l'utiliser (il n'est pas nécessaire de faire un OPEN OLD après un CREAT).

Les 3 requêtes ci-dessus permettent de passer de l'espace statique des fichiers non utilisés à l'espace dynamique des fichiers utilisés.

Les requêtes qui **détruisent** une FAU :

- CLOSE — détruit une FAU et
  - détruit un fichier Temporaire
  - ou — ferme un fichier Permanent
- DELET — détruit une FAU et détruit le fichier correspondant qu'il soit Temporaire ou Permanent
- EOJ (USR) — est une requête normalement utilisée par les superviseurs et non par les usagers, elle détruit toutes les FAU de l'usager spécifié (USR) et
  - détruit tous les fichiers Temporaires correspondants.
  - ferme tous les fichiers Permanents correspondants.

Les 3 requêtes ci-dessus permettent de passer de l'espace dynamique des fichiers utilisés à l'espace statique des fichiers non utilisés.

2.4.4 - Le partage des fichiers

a) **Définitions**

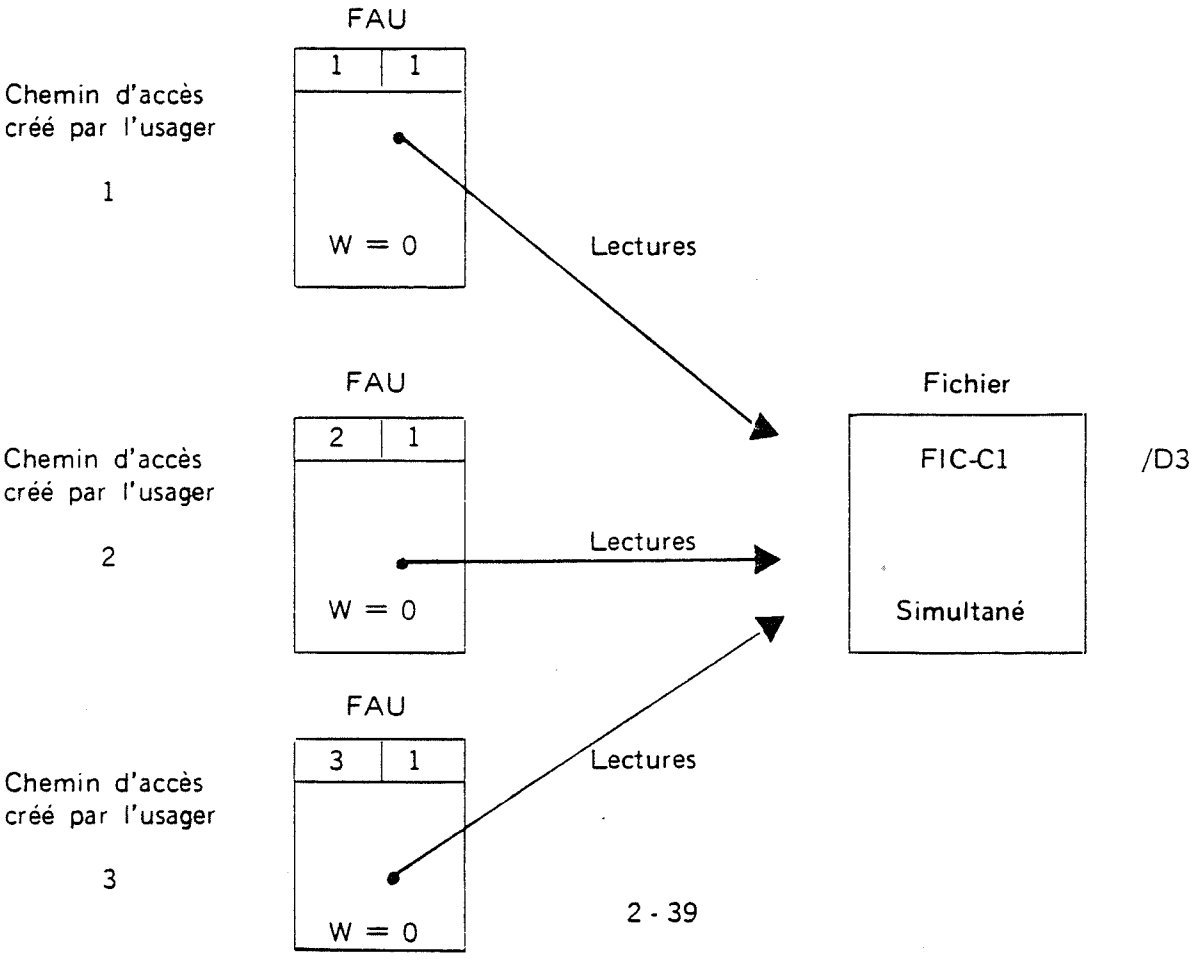
La notion de partage est liée à la notion d'usager. Dans le sens ou FMS permet ou non à **des usagers** de se **partager** des **fichiers.**

Partageable : on appelle partageable un fichier accessible par tous les usagers. Lorsqu'il est **non** partageable, un seul usager peut y accéder, en général le créateur du fichier.
Remarque :
On peut dire qu'un **Non** partageable est un fichier **Privé**, tandis qu'un **partageable** est un fichier **Public.**

Tous les fichiers **Temporaires** qu'un usager peut créer avec FMS sont **Non** partageables. Tous les fichiers **Permanents** qu'un usager peut créer avec FMS sont **Partageables.**

Partageable simultané : on appelle partageable simultané un fichier **utilisable** simultanément par plusieurs usagers. Lorsqu'il est non simultané un seul usager à la fois peut l'utiliser.

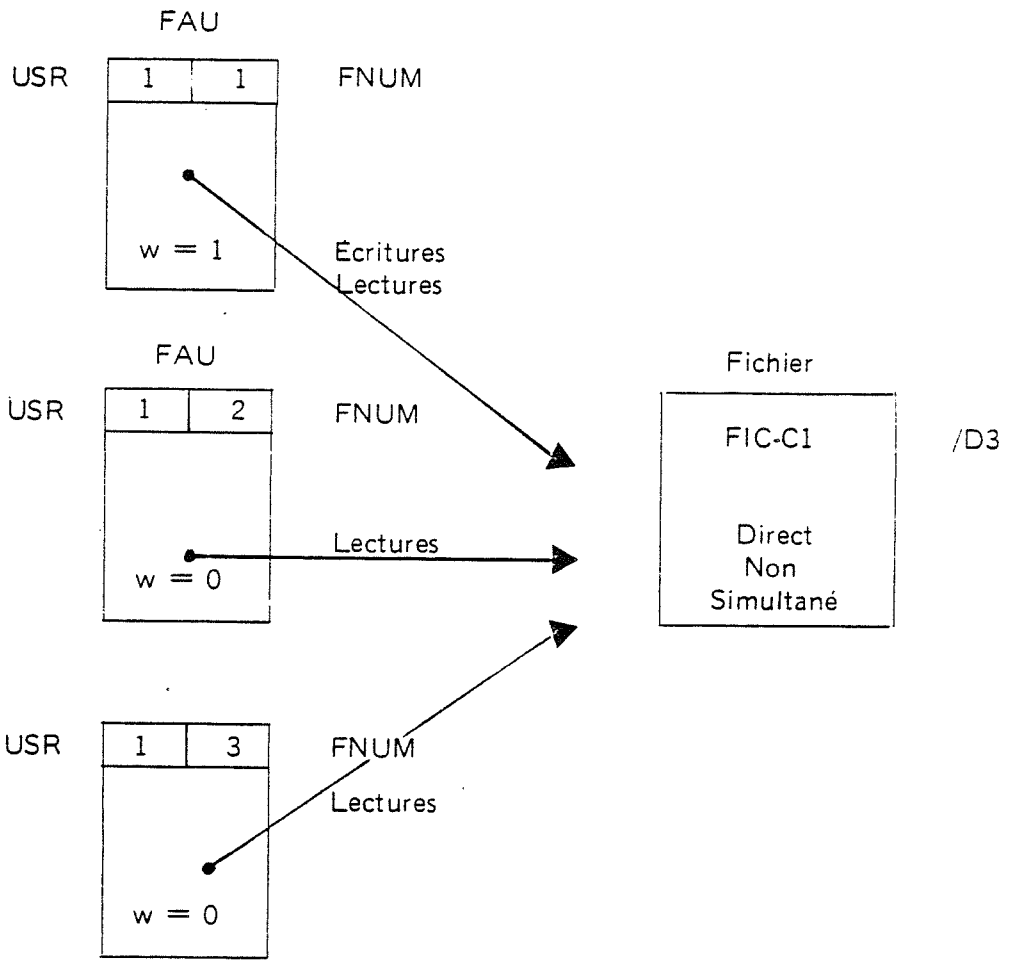
Schéma représentant à l'instant t un fichier partagé simultanément entre des usagers **différents.**



C'est l'usager qui définit à la création d'un fichier Permanent s'il sera de type simultané ou Non Simultané.

Multi accès : on dit, pour **un seul** usager (sinon on doit parler de partage), qu'il fait du multi-accès, lorsqu'il a créé simultanément plusieurs (FAU) chemins d'accès à un même fichier.

Schéma décrivant à l'instant t un usager réalisant du multi-accès sur un fichier Direct permanent NON Simultané.



l'usager numéro 1 a donc créé simultanément 3 chemins d'accès (nº 1, 2, 3) au fichier de nom FIC-C1 et se trouvant sur la FU-Support D3 .

b) **Le fichier Temporaire**

Un fichier **Temporaire** est NON partageable. C'est-à-dire qu'il est créé et utilisé par un usager et que tous les autres usagers qui s'exécutent pendant ce temps dans la machine **ne** peuvent **pas** accéder au fichier. En ce sens un fichier Temporaire est bien un fichier **privé.**
D'autre part, un fichier Temporaire a une durée de vie limitée à 1 exécution d'**un** usager en machine.

Objectifs des fichiers Temporaires

1°) **La production de programmes**

Dans le contexte de Traitement par Lots l'enchaînement des "STEP" est rendu très simple par l'utilisation de ces fichiers Temporaires.
En effet, le système est capable de gérer à la place de l'utilisateur les résultats intermédiaires produits par chaque step et que l'utilisateur ne désire pas à priori conserver.

Exemple : Compilation d'un programme écrit en PL 16, rentré en machine par le lecteur de cartes et conservation dans un fichier permanent du programme image mémoire produit.

```
/ JOB   TEMPOR, ER, D3 . . .
/ CALL  PL
/ IPLC
/ CALL  EDILE
/ ILNK
/ ELNK
/ CALL  BUILD
/ SLOD
/ CATAL IM, PROG1
/ EOJ
```

transformation du fichier Temporaire implicitement créé pour le résultat de BUILD en un fichier Permanent

PROG1 ⊕ ER ⊕ D3

2°) **Le stockage de résultats intermédiaires**

Dans un contexte Temps Réel par exemple une Tâche peut stocker sur disque dans un fichier Temporaire des résultats intermédiaires de calcul, n'intéressant pas les autres usagers. Lorsque l'usager, (la tâche) les a exploités, par exemple imprimés ou transformés en commandes au processus qu'il doit contrôler, il peut détruire son fichier Temporaire et ainsi ne pas encombrer inutilement la mémoire disque.

3°) **Remarque** : Un usager ne peut pas réaliser du multi-accès sur un fichier Temporaire. Cela aurait en fait pour lui, le sens suivant : créer deux fichiers Temporaires de même nom.

Exemple : L'usager 1 envoie les requêtes suivantes

```
OPEN   NEW   1, TEMP1, , D3, ...
OPEN   NEW   2, TEMP1, , D3, ...
```

Sur cette dernière requête FMS rend à l'usager le compte rendu `600 D : Fichier existant (requête ineffective)

```
WRITE   1, ...
WRITE   1, ...
  :
  :
REWIND  1, ...        utilisation du fichier
READ    1, ...
  :
  :
CLOSE   1             destruction du fichier Temporaire
                      TEMP1 ⊕ 1 ⊕ D3
                              ↑
                              nº USR
```

c) **Le fichier Permanent Simultané**

Un fichier **Permanent Simultané** est partageable comme tous les fichiers permanents gérés par FMS. A ce titre c'est un fichier **Public** donc accessible par un quelconque programme ou tâche qui en connaît l'identification.
De plus c'est un fichier accessible simultanément par plusieurs usagers, avec cependant une **importante restriction** contrôlée par FMS : ces différents accès sont autorisés en **lecture seulement.**
A l'aide d'une demande d'accès OPEN OLD l'usager spécifie par un booleen (W) s'il compte lire ou écrire sur le fichier. Avec un autre booleen (S = 1) il répète le fait que le fichier soit simultané, ce qui sera contrôlé par FMS.

Règles :
- Lorsqu'un usager demande la création d'une FAU pour accéder au fichier en **lecture**, FMS n'accepte sa demande que si le fichier est **libre** ou utilisé en **lecture seulement** par d'autres usagers.
  Lorsque le fichier est déjà utilisé en **écriture**, sa demande est rejetée.
- Lorsqu'un usager demande la création d'une FAU pour accéder au fichier en **écriture**, FMS n'accepte sa demande que si le fichier est **libre.**

Objectifs des fichiers Permanents Simultanés

1°) **La consultation de fichiers**

Permettre à plusieurs usagers qui ne se connaissent pas de consulter simultanément un même fichier, est un mode de partage simple et viable.
Lorsqu'un usager consulte un fichier il n'a pas connaissance des autres usagers pris en charge en même temps que lui par le système d'exploitation. C'est le cas des systèmes : Traitements par Lots en multi-programmation, Temps Partagé, Temps Réel avec Background). Exemple au chapitre 2.4.4 § a) .

2°) **Application au système**

Le système peut fournir des services de Traitement par Lots en mono-programmation (Background) et de Temps Réel (Foreground).
Au niveau du Background seulement, les possibilités de partage simultané entre des usagers ne peuvent donc pas être utilisés.
Par contre on peut employer des fichiers permanents simultanés pour que les programmes du Background et les tâches du Foreground puissent accéder de façon cohérente aux mêmes informations.

Exemple : Dans une première phase, une tâche du Foreground élabore des données dans un fichier simultané.

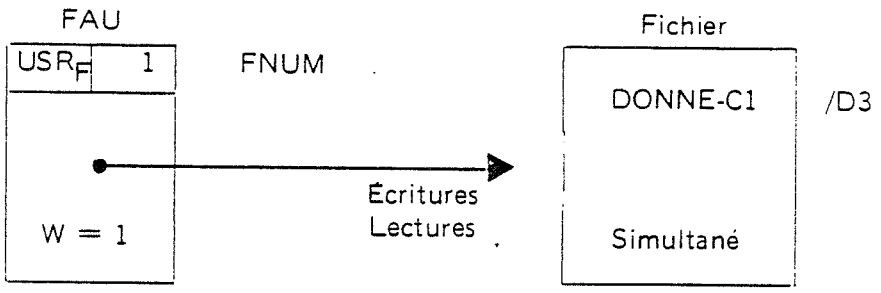
1°) ouverture en écriture

```
OPEN    OLD   1, DONNE-C1, D3, W = 1, ...
```

Cela crée un chemin d'accès identifié par

$USR_F \oplus 1$

↑ numéro d'usager de la Tâche Foreground



2°) Écritures de nouvelles données.

WRITE 1, ...

3°) Fermeture du fichier et destruction de la FAU.

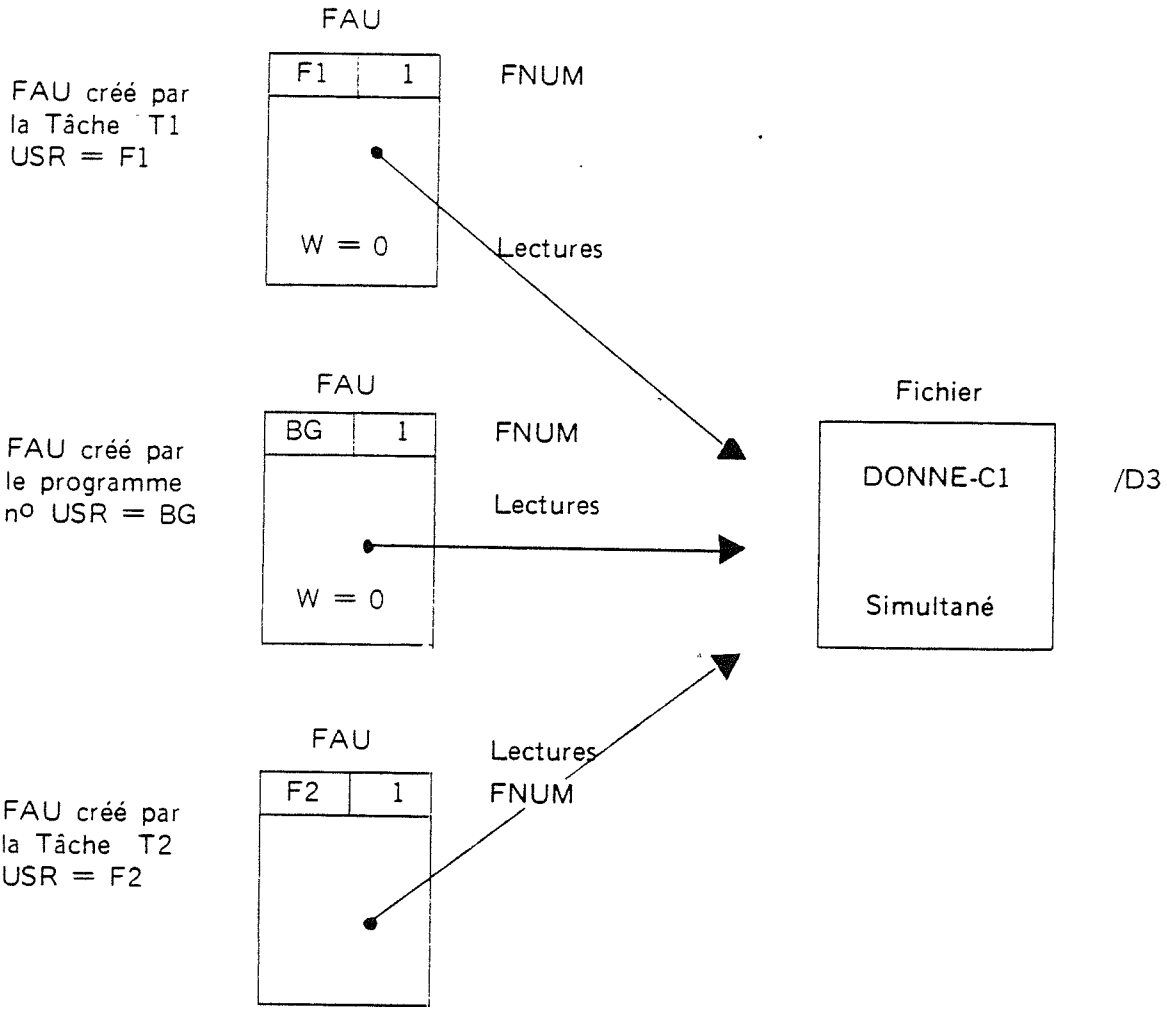
CLOSE 1, ...

Dans une deuxième phase, des Tâches du Foreground et un programme du Background peuvent simultanément lire sur ce fichier.

1°) Des ouvertures en lecture. Exemple :

OPEN OLD 1, DONNE-C1, D3, W = 0, ...
etc pour les autres (tâches, programme)

2º) La situation de partage simultané en lecture



3º) Destruction des FAU et sur le dernier CLOSE fermeture du fichier.

Programme	CLOSE	1, ...	(USR = BG)
Tâche T2	CLOSE	1, ...	(USR = F2)
Tâche T1	CLOSE	1, ...	(USR = F1)

3º) **Application au Temps Partagé**

Dans un contexte de Temps Partagé simple gérant n consoles de dialogue, il faudra associer de façon bijective à chaque poste de travail un numéro d'usager. On peut remarquer que la notion d'usager n'est pas associée aux personnes mais aux consoles. C'est-à-dire aux différents contextes d'exécution que le système d'exploitation est capable de gérer simultanément. La notion d'usager ne peut donc pas résoudre des problèmes d'informations privées ou publiques, permanentes. Seul les fichiers Temporaires permettent de stocker des informations privées.
Avec un système de Temps Partagé simple un utilisateur doit pouvoir consulter des informations permanentes communes à tous les utilisateurs et traiter des informations permanentes qui lui

sont propres. Il faudra dans les deux cas utiliser des fichiers permanents **Simultanés.**

— Dans le premier cas, les fichiers communs pourront donc être consultés simultanément par plusieurs usagers. Ils pourront être également protégés en écriture à fin d'augmenter la fiabilité du système.

— Dans le deuxième cas, le caractère privé des informations ne pourra pas être traité avec des fichiers permanents non Simultanés, car tout fichier Permanent est partageable donc Public, mais plutôt en utilisant la notion de catalogue, concrétisée par les mots de passe publics (PUBW). En effet il suffit d'associer de façon bijective un PUBW à une personne ou un groupe de travail utilisateur du système.

4o) **Remarque**

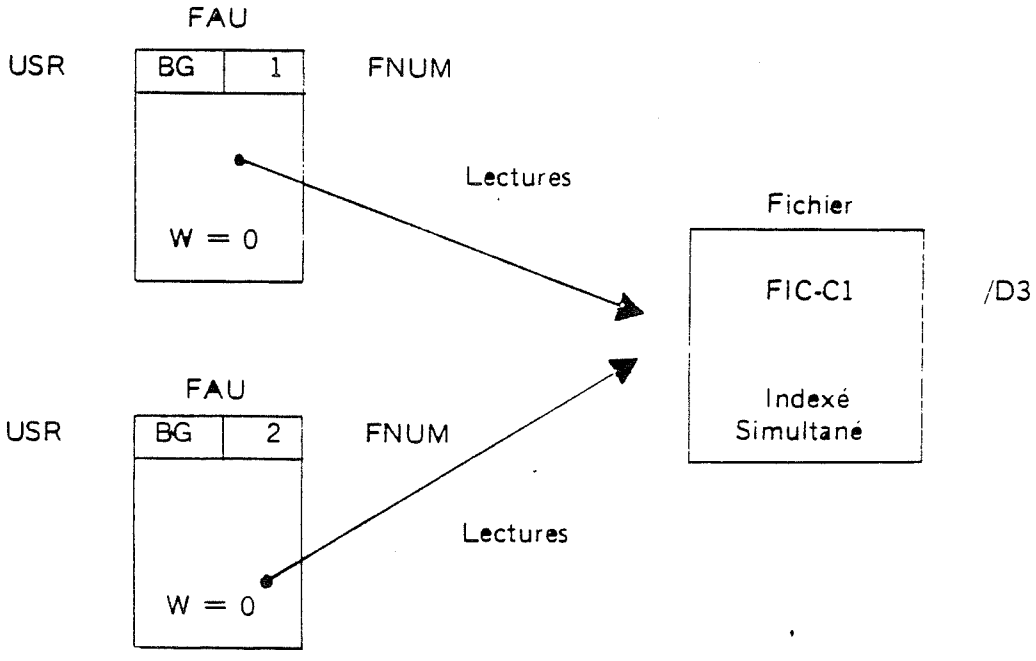
Il est également possible de réaliser du **multi-accès** en **lecture seulement** sur un fichier permanent Simultané. Lorsqu'**un** usager possède à un instant donné **plusieurs** FAU en **lecture** sur le même fichier cela lui permet par exemple de réaliser simultanément plusieurs accès séquentiels indépendants.

Exemple : Soit un programme s'exécutant sous BOS16 (usager : BG), et utilisant un fichier Indexé nommé FIC-C1, D3 et constitué de 2 articles A et B. Supposons qu'un Traitement sur ce fichier implique de **lire simultanément** les **deux articles** en **séquentiel** (Portion d'Article).

1o) Création de 2 unités d'accès FNUM 1 et 2

```
OPEN    OLD   1, FIC-C1, D3, W = 0 (lecture) ...
OPEN    OLD   2, FIC-C1, D3, W = 0 (lecture) ...
```

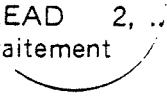
La situation de multi-accès en lecture.



2°) Traitement

```
IREAD   1,   A, ...      Sélection de l'article A par la FAU 1
IREAD   2,   B, ...      Sélection de l'article B par la FAU 2
```

```
READ    1, ...
READ    2, ...           Boucle de lecture des 2 articles, jusqu'à par exemple
traitement               Fin d'article : `600 1
```



3°) Destruction de la FAU n° 1 ... (le fichier reste ouvert)

```
CLOSE   1, ...
```

4°) Destruction de la FAU n° 2 et fermeture du fichier

```
CLOSE   2, ...
```

d) **Le fichier Permanent NON Simultané**

Un fichier permanent **NON Simultané** est partageable comme tous les fichiers permanents gérés par FMS. A ce titre c'est un fichier **Public**, donc accessible par un quelconque programme ou tâche qui en connaît l'identification.

Règle 1er Niveau

— Lorsqu'un fichier permanent NON Simultané est utilisé par un usager, FMS en interdit l'accès à toute demande provenant d'un autre usager.

Cette règle de partage permet pour un seul usager à la fois de réaliser de façon cohérente du multi-accès avec des FAU en écriture et/ou en lecture.
Cependant toutes les méthodes d'accès ne permettent pas de réaliser du multi-accès en écriture.

Règle 2e Niveau

— Pour toutes les méthodes d'accès un usager peut réaliser du multi-accès en lecture comme pour un permanent simultané.
— Le multi-accès en écriture **et** lecture n'est possible qu'avec les fichiers Direct ou Direct à Trous.

A l'aide d'une demande d'accès OPEN OLD l'usager spécifie pour ouvrir un fichier Permanent NON Simultané.

— **S** (1 bit) : S = 0 l'usager répète que le fichier est NON Simultané, ce qui est contrôlé par FMS-E.
— **W** (1 bit) :
W = 0 si l'unité d'accès doit être créée pour lire seulement.
W = 1 si l'unité d'accès doit être créée pour lire **et** écrire.
— **RWK** (2 bits): Ce paramètre est une **clé d'accès** par laquelle l'usager spécifie dans quelle condition de multi-accès, il accepte d'utiliser le fichier.

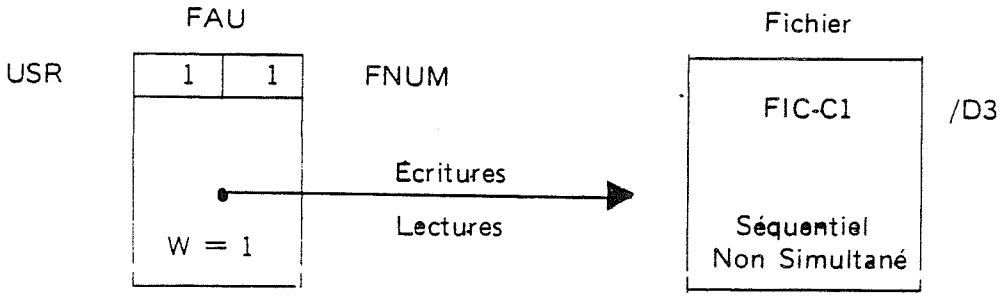
RWK = 0 : Mono-accès
RWK = 1 : Multi-accès lecture
RWK = 3 : Multi-accès écriture

Règle 3e Niveau

— **Mono-accès : RWK = 0** : l'usager veut être tout seul, au sens une seule FAU, pour utiliser le fichier.

Exemple : Requête de l'usager n° 1

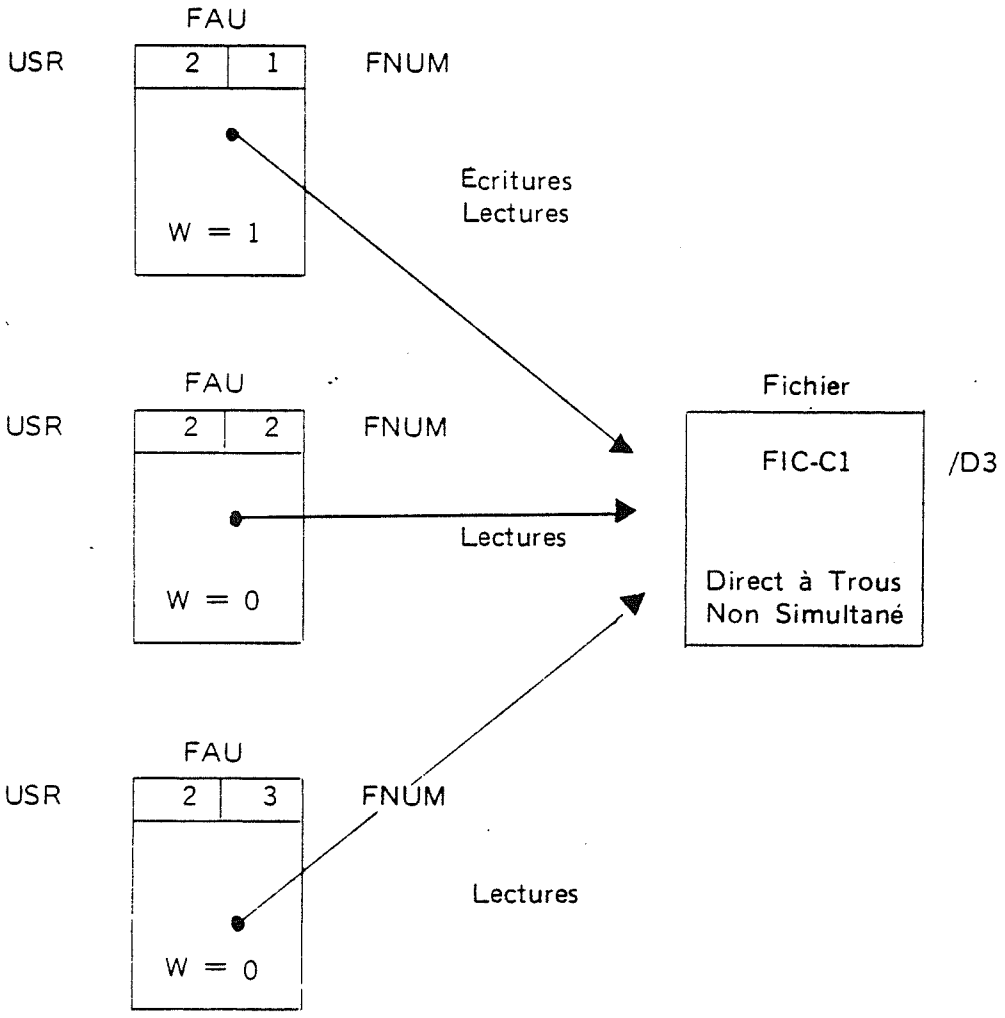
OPEN OLD 1, FIC-C1, D3, S = 0, W = 1, RWK = 0, ...



— **Multi-accès Lecture** : RWK = 1 : l'usager accepte avec sa FAU d'autres unités d'accès en lecture seulement.

Exemple : 3 requêtes de l'usager nº 2

```
OPEN    OLD,  1, FIC-C1, D3, S = 0, W = 1, RWK = 1, ...
OPEN    OLD,  2, FIC-C1, D3, S = 0, W = 0, RWK = 1, ...
OPEN    OLD,  3, FIC-C1, D3, S = 0, W = 0, RWK = 1, ...
```

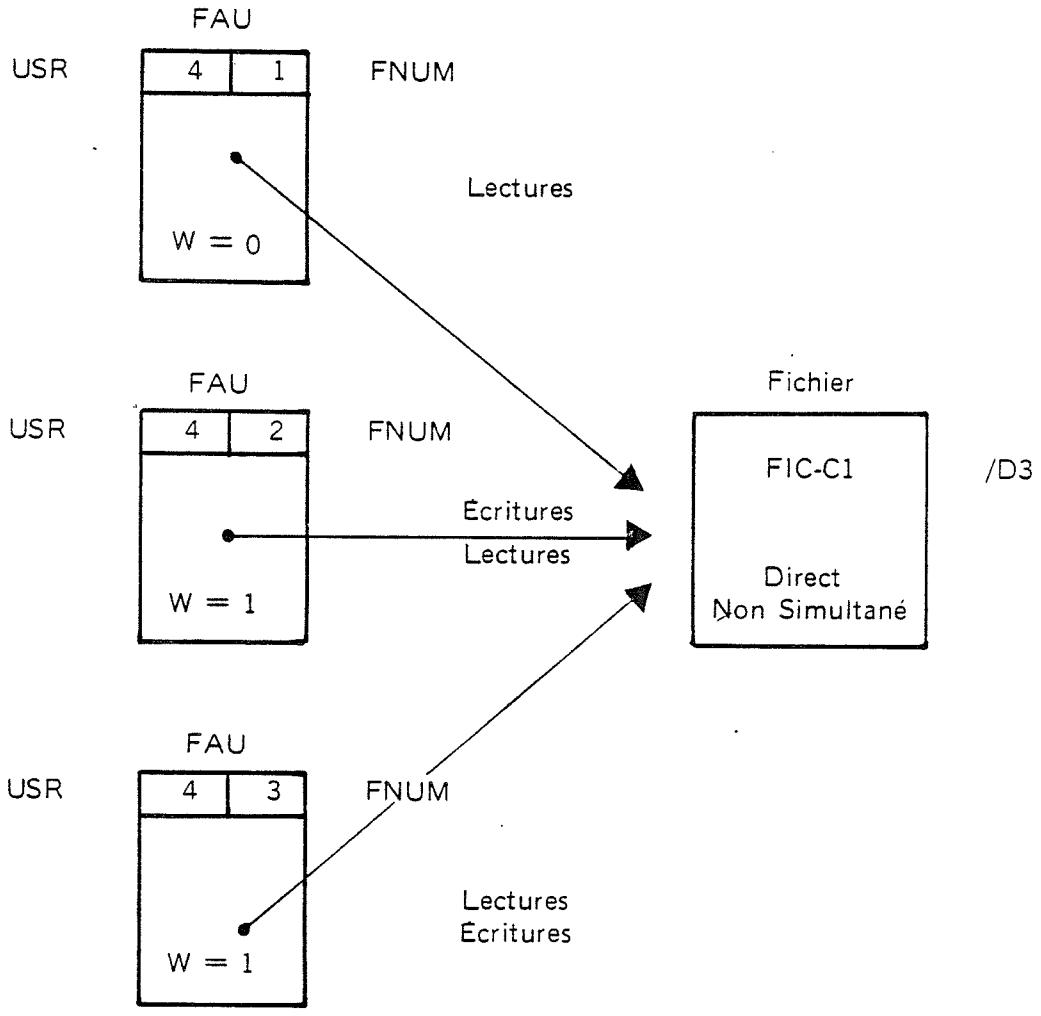


Remarque : La FAU 2.1 est en écriture. Cette possibilité est réservée à la première ouverture si elle le demande (W = 1) d'un fichier Direct ou Direct à Trous. Pour les autres méthodes d'accès **toutes** les ouvertures doivent être demandées en lecture seulement (W = 0) .

— **Multi-accès écriture : RWK = 3** : l'usager accepte avec sa FAU, d'autres unités d'accès demandées en écriture ou en lecture. Ce type de multi-accès est réservé aux fichiers Directs.

Exemple : 3 requêtes de l'usager nº 4

```
OPEN   OLD   1, FIC-C1, D3, S = 0, W = 0, RWK = 3
OPEN   OLD   2, FIC-C1, D3, S = 0, W = 1, RWK = 3
OPEN   OLD   3, FIC-C1, D3, S = 0, W = 1, RWK = 3
```

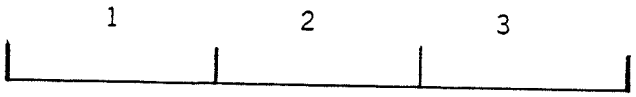


Objectifs des fichiers Permanents NON Simultanés.

1º) Utilisation du Direct en Temps Réel

Dans un système Temps Réel, plusieurs **tâches** appartenant au même usager, c'est-à-dire constituant un ensemble software cohérent, peuvent accéder simultanément en écriture à un fichier **Direct** Permanent NON Simultané. Chaque tâche crée une unité d'accès au fichier.

Exemple : Soit un fichier Direct constitué de 3 articles.

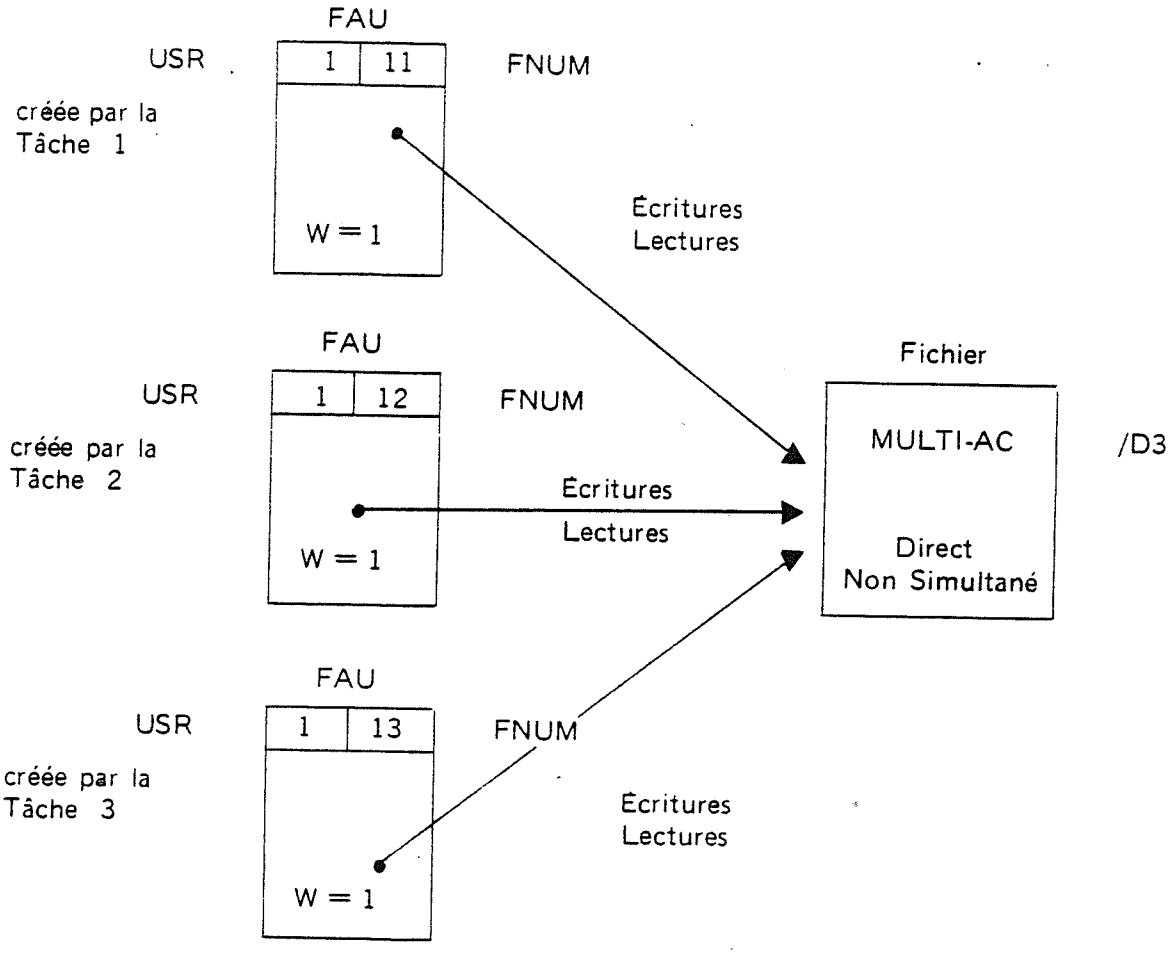


3 tâches Temps Réel désirent simultanément mettre à jour le contenu des articles en utilisant le mode Séquentiel (Portion d'Article Statique). Supposons une convention simple la tâche i gère l'article de numéro i .

1°) **Création des unités d'accès** (usager n° 1) par les tâches, respectivement n° 1, 2, 3.

```
OPEN    OLD   11, MULTI-AC, D3, S = 0, W = 1, RWK = 3, ...
OPEN    OLD   12, MULTI-AC, D3, S = 0, W = 1, RWK = 3, ...
OPEN    OLD   13, MULTI-AC, D3, S = 0, W = 1, RWK = 3, ...
```

La situation de multi-accès écriture



2°) **Traitement**

— **Tâche T1**

DWRITE	11, 1, ...	Sélection de l'article n° 1
WRITE	11, ...	Ecriture séquentielle jusqu'à la fin de l'article
CLOSE	11, ...	Destruction de la FAU

— **Tâche T2**

DWRITE	12, 2, ...	Sélection de l'article n° 2
WRITE	12, ...	Ecriture séquentielle jusqu'à la fin de l'article
CLOSE	12	Destruction de la FAU

— **Tâche T3**

DWRITE	13, 3, ...	Sélection de l'article n° 3
WRITE	13, ...	Ecriture séquentielle jusqu'à la fin de l'article
CLOSE	13, ...	Destruction de la FAU

— Le dernier close (chronologiquement) ferme le fichier et le rend libre pour être utilisé par un usager différent ou non et dans un autre contexte de multi-accès, par exemple multi-accès lecture.

2°) **Application à RTES16 avec Background**

Les fichiers Permanents NON Simultanés, quelle que soit leur méthode d'accès, s'appliquent au système avec Background.
Du fait que tous les fichiers **Permanents** gerés par FMS sont **Publics**, un programme s'exécutant en Background peut ouvrir un fichier normalement réservé au Foreground.
Pour protéger le fonctionnement Temps Réel d'une application on utilisera la règle suivante :

Règle :
— Les Fichiers Permanents du **Foreground** doivent être **NON Simultanés**
— Les Fichiers Permanents du **Background** doivent être **Simultanés.**

Lorsqu'un usager demande l'accès à un fichier Permanent (OPEN OLD) il doit préciser à l'aide d'un booleen (S) la nature de permanent :

— S = 0 le fichier est NON Simultané
— S = 1 le fichier est Simultané

FMS en contrôle la validité et rejette la demande, si le fichier n'est pas de la nature précisée.
Compte-rendu d'erreur n° `6015 : Permanent de nature différente.

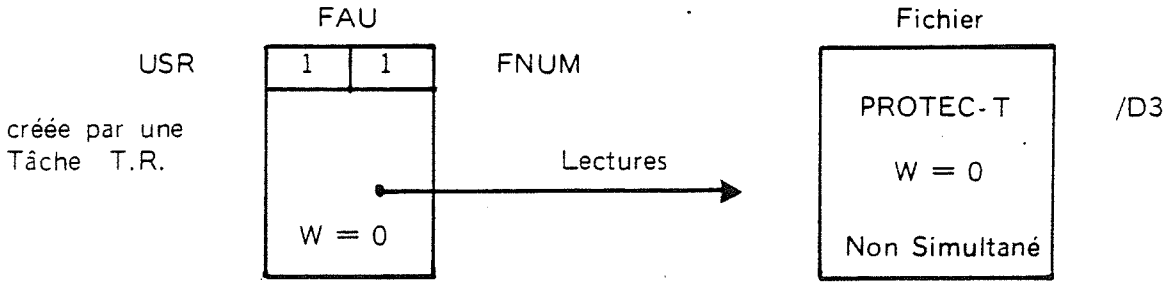
La protection des fichiers NON Simultanés du Foreground s'effectue à 2 niveaux :

— **Au niveau des requêtes (SVC)**
Dans un programme du Background qui normalement n'utilise que des fichiers Permanents Simultanés le contrôle effectué sur le booleen S, est un obstacle supplémentaire à l'accès aux Fichiers du Temps Réel. Le premier obstacle étant de connaître l'identification des fichiers. La possibilité d'interdire en écriture un fichier permanent peut être également employée pour protéger les fichiers NON Simultanés du Temps Réel. Cette troisième protection sera encore plus efficace si **tous** les fichiers permanents Simultanés du Background sont accessibles en écriture et leurs unités d'accès créées en écriture également.

Exemple d'utilisation de la protection écriture dans une Tâche Foreground sur un fichier NON Simultané, protégé en écriture.

1°) Ouverture du fichier en **lecture**

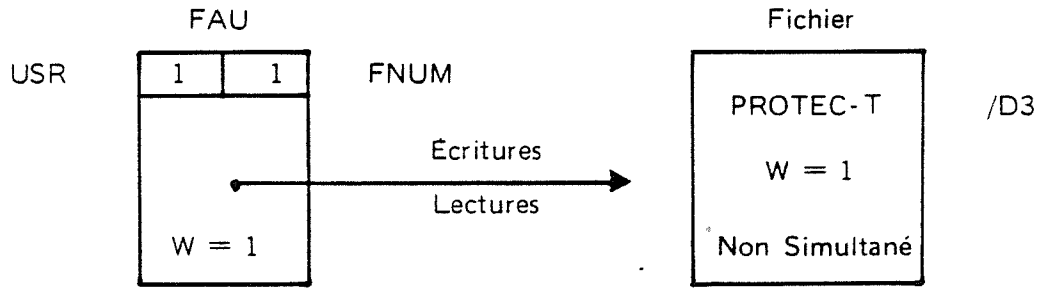
OPEN OLD, 1, PROTEC-T, D3, S = 0, W = 0, ...



A partir de cet instant tout autre usager (que le n° 1) ne peut plus accéder au fichier (il est NON Simultané). Et ceci jusqu'à ce que le fichier soit libre.

2°) Autoriser l'accès en écriture sur ce fichier. Il faut également repréciser le support SU/FU.

ALTER 1, FU = D3, S = 0, W = 1, RWK = ? , ...



A partir de cet instant la Tâche peut écrire par l'unité d'accès nº 1. Si ensuite la Tâche fermait le fichier (CLOSE 1, ...) celui-ci serait accessible en écriture par un quelconque usager y compris un programme de l'usager Background.

3º) Traitement sur le fichier, avec éventuellement du multi-accès (RWK = ?) .

4º) Fermeture du fichier en le rendant de nouveau protégé en écriture.

```
ALTER   1, FU = D3, S = 0, W = 0, RWK = 0,

CLOSE   1, ...          Destruction de la FAU 1
```

A partir de maintenant le fichier Permanent est toujours NON Simultané (S = 0 dans le dernier ALTER) et accessible qu'en lecture par un quelconque usager.

— **Au niveau des commandes du Background**
A ce niveau la protection est totale car tous les fichiers Permanents que l'on peut utiliser à l'aide des **commandes** du **Background** ne peuvent être que **Simultanés.** C'est en effet le système Background lui-même qui crée ou ouvre les fichiers à la place de l'utilisateur lorsque ce dernier le lui demande.

2.5 - LES FAU PUBLIQUES

2.5.1 - Introduction

a) **L'objectif d'une FAU publique** est de permettre à des usagers différents l'utilisation d'un même chemin d'accès à un fichier : FAU

Cet objectif est particulièrement intéressant pour les applications de consultation et mise à jour de fichiers en temps réel, supportées par un système Temps Partagé.

La notion de FAU publique permet :

— le partage des données, pour consultation et mise à jour, au niveau des fichiers, ainsi qu'au niveau des articles.
— une diminution de l'encombrement des tables du système de fichiers : diminution du nombre de FAU.
Ainsi qu'un outil de synchronisation automatique des requêtes.

Remarques :

Voir interface de programmation au paragraphe 3.5

2.5.2 - Définition d'une FAU publique

a) Une FAU publique se distingue d'une FAU privée exclusivement par la valeur de son FNUM. Celui-ci appartient à l'ensemble des FNUM public, (FNUMP) définis à la génération du système d'exploitation, avec GENFMS, par 2 plages de FNUM réservées aux FAU publique.

FNUMP $\in$ [FNUMP1, ... FNUMP2] $\cup$ [FNUMP3, ...FNUMP4]

tel que : FNUMP1 $\leqslant$ FNUMP2 $\leqslant$ FNUMP3 $\leqslant$ FNUMP4

Remarques

— il sera donc par définition impossible de créer une FAU privée avec un FNUM appartenant aux plages des FNUM publics, c'est la valeur du FNUM qui permet à FMS de reconnaitre que l'on veut créer ou utiliser une FAU privée ou publique.
— dans un système généré sans FAU publiques (option par défaut) toute la plage des FNUM est utilisable pour les FAU privées.
— pour réaliser du logiciel portable sur un ensemble de systèmes d'exploitation il est nécessaire de définir pour chacun les mêmes plages de FNUMP publics.

Dans la mesure du possible il est proposé d'utiliser les plages suivantes pour les FNUM publics:

FNUMPi $\in$ [`20 `3F] $\cup$ [`F8 `FF]
32 63 248 255
32 valeurs 8 valeurs

b) Une FAU publique se distingue d'une autre FAU publique par 2 paramètres :

— le FNUM public : FNUMP
— un numéro d'usager public : USRP $0 < USRP \leqslant 255$

A tout usager (ensemble de Tâches) est associé un numéro d'usager public géré par le système d'exploitation au même titre que l'USR et permettant de regrouper les usagers en classes d'usagers publics ayant accès aux mêmes FAU publiques.

Exemple de classes d'usagers publics, ou **groupe** d'usagers

— une classe dite système pour les FAU publiques du système d'exploitation
— une classe dite utilisateur pour les FAU publiques de l'application.

Dans cet exemple la notion d'usager public permet de protéger les FAU publiques du système d'un mauvais fonctionnement de l'application.

Remarques :

Pour assurer la compatibilité ascendante du logiciel existant il existe plusieurs solutions :

— générer un système sans mettre en oeuvre la notion de FAU publique : option par défaut avec GFMS16,
— fonctionner dans un contexte de mono-programmation,
— dans un système multi-tâches qui gère la notion de FAU publique (voir GFMS16), il faut respecter deux règles pour intégrer de nouvelles tâches utilisant la notion de FAU publique, sans recompiler le logiciel existant.

1) Pour le logiciel existant, il faut utiliser le numéro d'usager public ZERO, (ce qui est le cas actuellement)
   USRP = 0 $\Rightarrow$ FAU privée
2) Pour le nouveau logiciel utiliser des numéros d'usager publics différents, des numéros d'usagers privés exploités par le logiciel existant.

— d'une façon générale : si USRP = USR

1) le partage des FAU publiques est impossible et dans ce cas une FAU publique fonctionne comme une FAU privée
2) la synchronisation sur sémaphore fonctionne pour le multi-accès.

c) Identification des FAU

FAU Privée

USR	FNUM
0	

FAU Publique

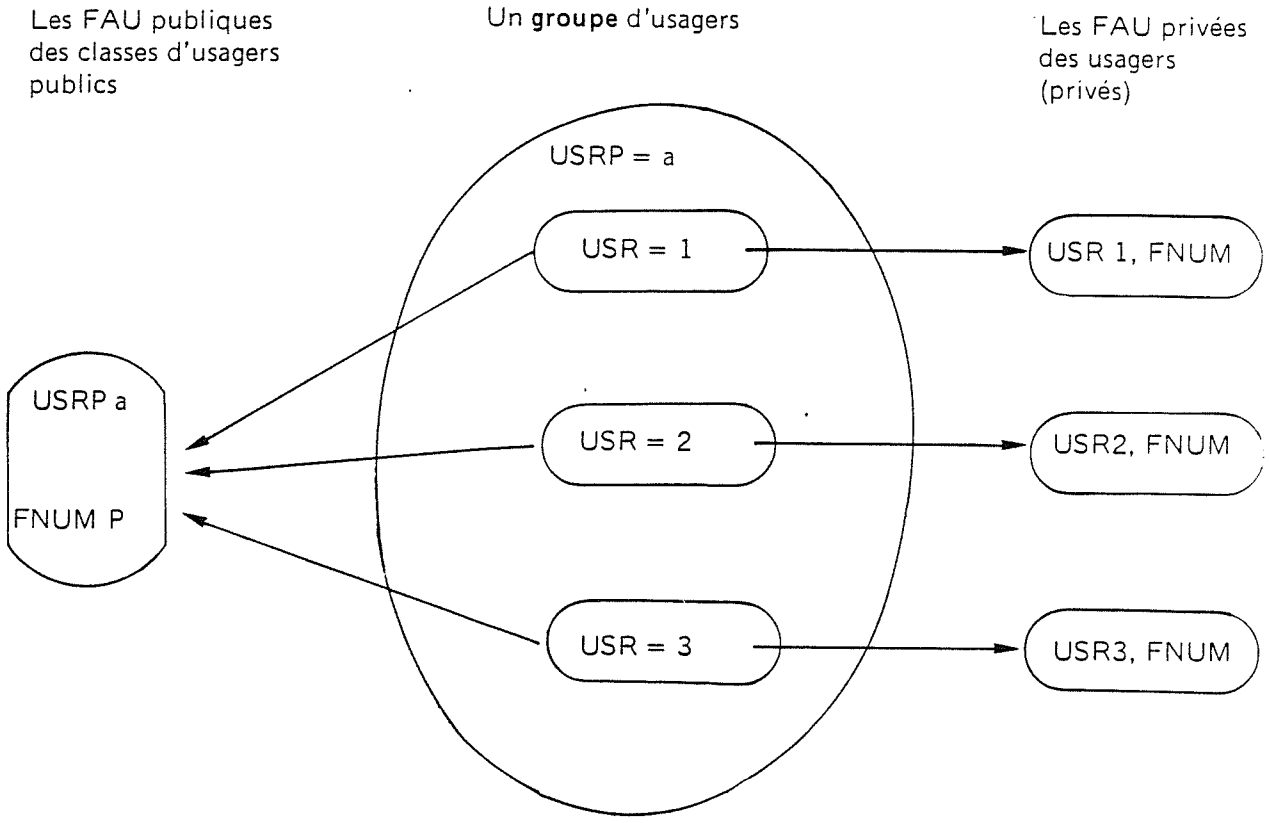
USRP	FNUMP
@ SEMA	

USRP est fourni par le superviseur dans RA pour tout appel à FMS. (cependant : USRP = 0 $\Rightarrow$ FAU privée)

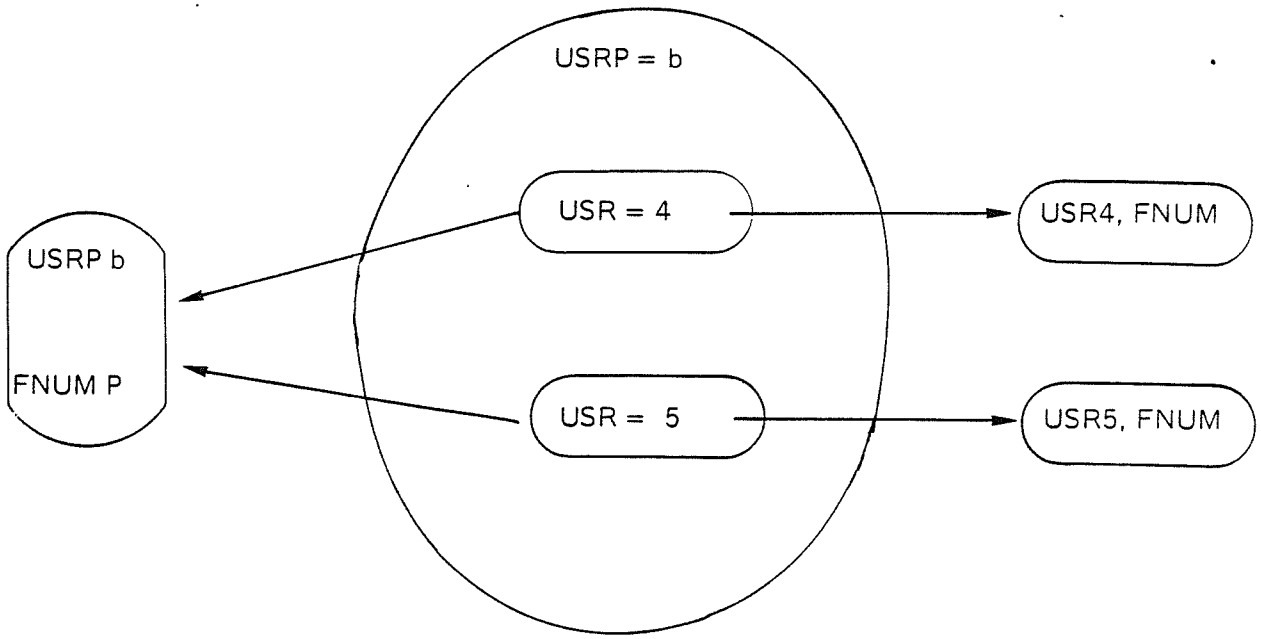
USRP	USR

FNUMP est tel que : FNUMP1 $\leqslant$ FNUMP2
ou FNUMP3 $\leqslant$ FNUMP4

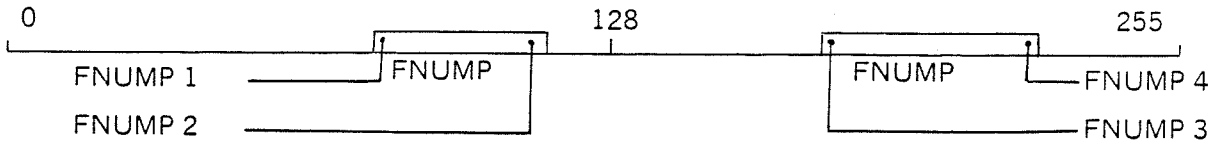
@ **SEMA** : est l'adresse d'un pavé de la ZDF contenant le sémaphore de la FAU publique.
autrement dit la création d'une FAU publique nécessite 1 pavé de plus dans la ZDF.



Un **groupe** d'usagers



Les 2 plages de FNUM publics (FNUMP) exemple :



2.5.3 - Fonctionnement d'une FAU publique

a) Une FAU publique est créée par l'une des 3 requêtes :

OPEN NEW
OPEN OLD et détruite par une requête CLOSE ou DELET
CREAT

Elle permet l'accès à des fichiers temporaires ou permanents simultanés ou non simultanés. Une FAU publique est accessible pour des requêtes d'E/S par toutes les tâches du même usager public (USRP) que celle qui a créé la FAU.

Pour un fichier permanent existant, les tâches peuvent toutes exécuter :

OPEN OLD E/S CLOSE

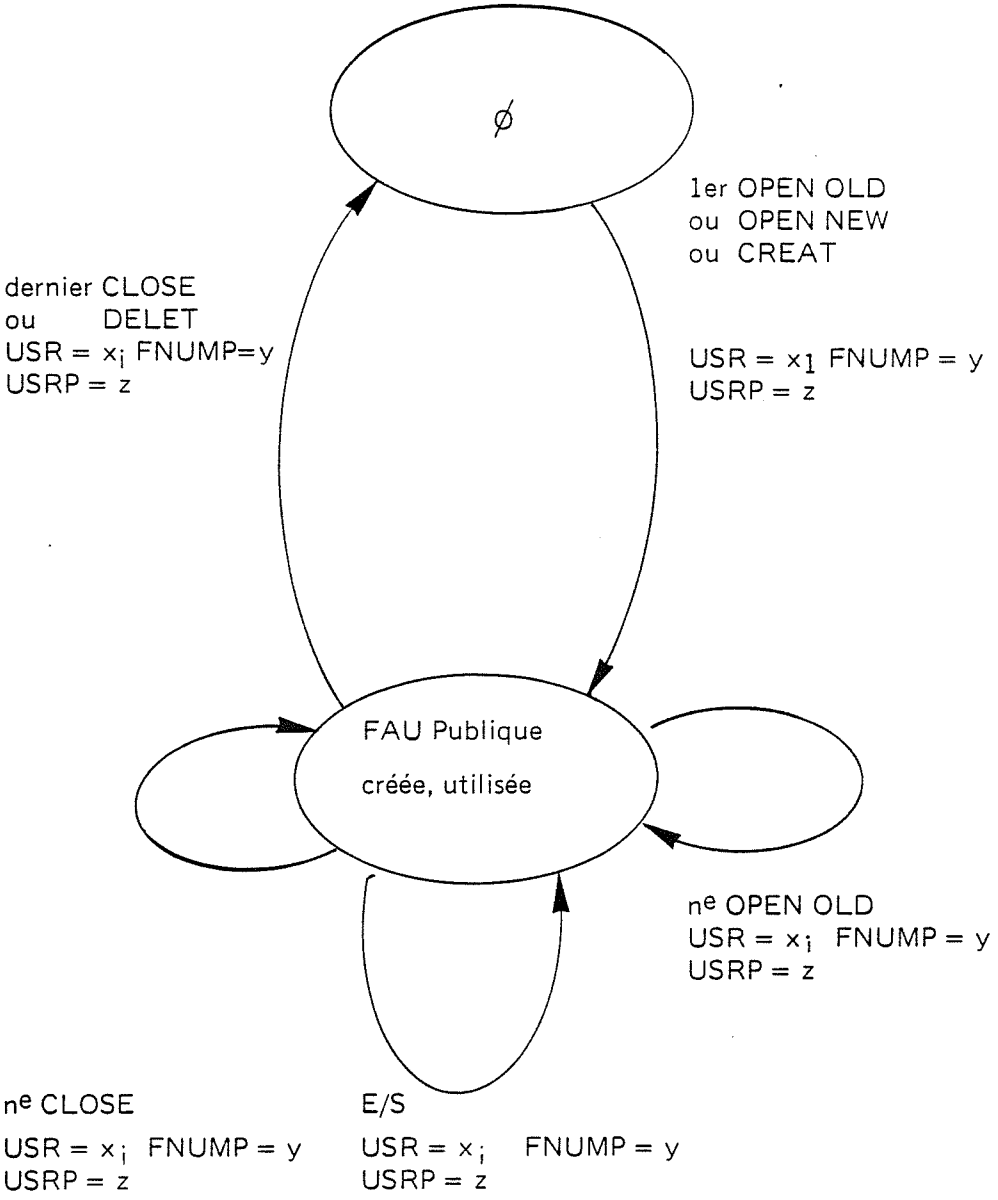
C'est le 1er OPEN OLD qui crée la FAU et dont le fonctionnement est identique à un OPEN OLD normal, c'est le dernier CLOSE qui détruit la FAU et dont le fonctionnement est identique à un CLOSE normal.

Les autres OPEN OLD et CLOSE sont "quasiment ineffectifs" ils se contentent de faire respectivement + 1 et — 1 sur le compteur de la FAU.

L'accès à une FAU publique est géré par un sémaphore d'exclusion. Il synchronise automatiquement toutes les requêtes, qui s'adressent à la FAU ; c'est-à-dire les requêtes des niveaux Article et Portion d'Article, ainsi que les requêtes 1 à 9 puis 13 et 15 du niveau OPEN CLOSE. Le sémaphore d'une FAU publique est alloué dans un pavé de la zone qui contient les FAU et les DF : la ZDF.

Pour compléter le fonctionnement d'une FAU publique et ainsi permettre au logiciel d'application de réaliser un partage de données au niveau article ; FMS fournit une nouvelle requête : ATADET : Attache Detach qui permet d'exécuter sur une FAU publique une suite de requêtes sans être interrompu, et qui synchronise automatiquement les séquences de requêtes, exclusives.

b) Enchainement des requêtes sur une FAU publique



c) Mécanisme de partage au niveau Article

A titre d'exemple le service des FAU publiques permet au programmeur d'application de réaliser un mécanisme de partage au niveau article, de la façon suivante.

Exemple : avec un fichier séquentiel Indexé

1) **Bloquer l'article :**

ATADET (AD = 0) .	Attachement de la Fau Publique permettant d'accéder au fichier
SIREAD	Lecture de l'article
Si l'article est occupé	
ATADET (AD = 1)	Libérer la FAU puis attendre que l'article soit libre ....
Si l'article est libre	
SIWRIT	Pour marquer l'article : 1 bit signifiant que l'article est occupé
ATADET (AD = 1)	Pour libérer la FAU

2) Exécution du traitement désiré avec synchronisation automatique

Au niveau de chaque requête ou au niveau des suites de requêtes

3) Libération de l'article

ATADET (AD = 0)	Attachement de la FAU
SIREAD	Pour lire l'article
SIWRIT	Pour réécrire l'article en réinitialisant le bit d'occupation
ATADET (AD = 1)	Détachement de la FAU

Remarque :

Avec un fichier Direct pour exemple il est possible de libérer l'article à l'aide d'un seul ordre DWRITE en prenant comme convention que le 1er mot de l'article contient le booléen d'occupation.

DWRITE (LBU = 2 octets) Pour ne réécrire que le 1er mot de l'article sans modifier le reste.

2.6 - LA GESTION DE VOLUME

2.6.1 - Principe général

La gestion de volume disque apporte deux services nouveaux :

— la gestion des supports : gestion de label, des tables de cylindres en défaut,
— la gestion d'espace : la reconfiguration dynamique des tables de IOCS et FMS.

L'utilisation optionnelle de ces deux outils permet d'accroître la sécurité de fonctionnement des systèmes disque, et la portabilité du logiciel et des supports.

a) **La reconfiguration dynamique des FU** est basée sur une technique qui consiste à écrire sur le disque lui-même le découpage en espaces de ce disque. Ainsi chaque support disque possède sa propre définition de structure : une table d'espace.

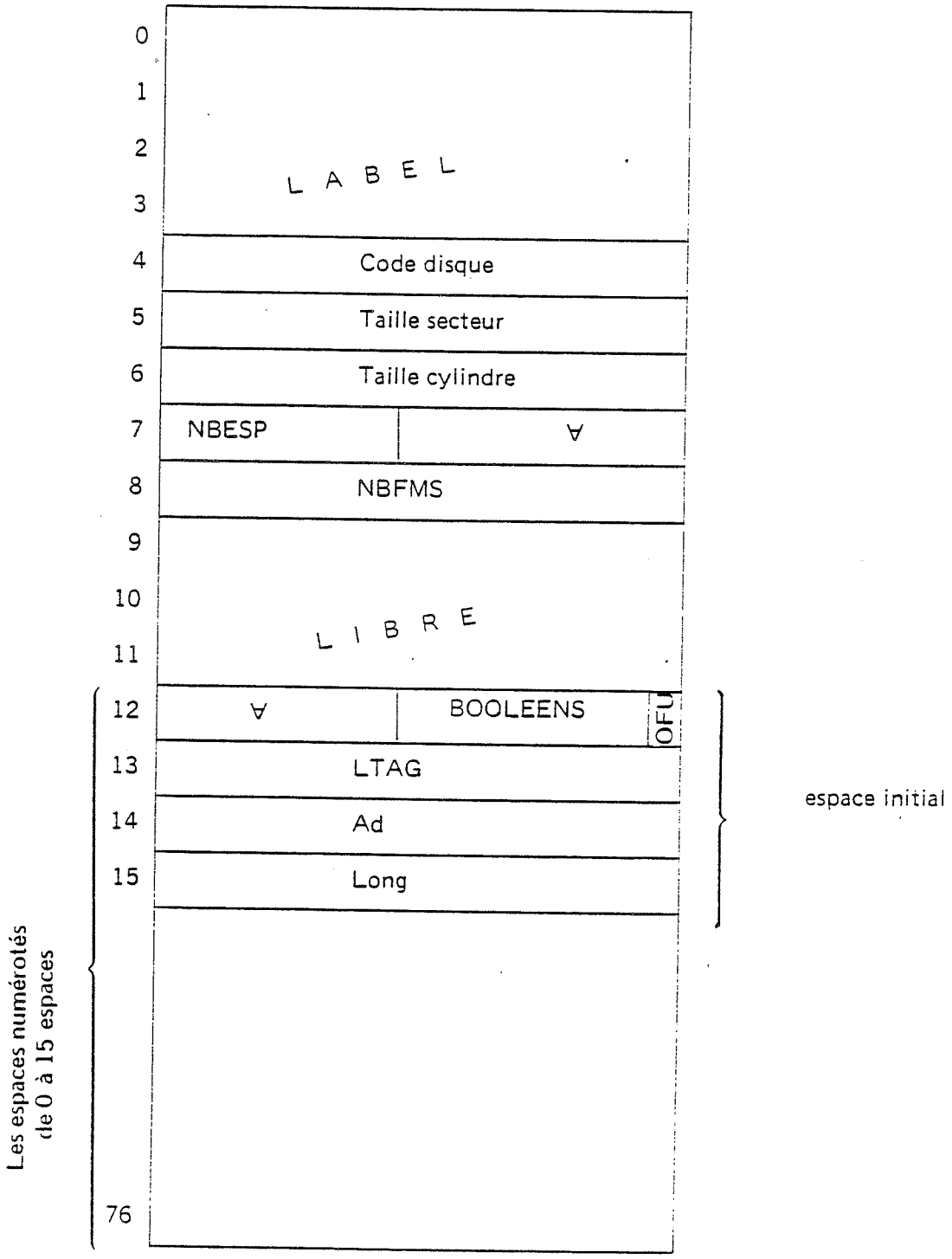
Suite à une commande de montage de volume, les tables de IOCS et FMS sont reconfigurées en fonction de ce qui a été défini dans la table d'espace du disque.

De plus au montage de volume, FMS contrôle que les FU du volume précédent sont fermées. Enfin, au niveau de l'initialisation d'un espace, FUP4 - FUINI contrôle le non-recouvrement des FU gérables par FMS sur un support donné.

b) **La gestion des supports permet :**

— la gestion des cylindres en défaut du support disque,
— un contrôle de label au montage de volume,
— un dialogue de montage et démontage de volume plus ou moins évolué selon le système d'exploitation,
— la gestion des labels par les différents utilitaires,
système : formatage, définition de structure FUP 4 : SDEF, initialisation d'espace FMS FUP 4 : FUINI archivage de fichier, etc ...

2.6.2 - La table d'espaces d'un support disque



Règle : un espace est géré par FMS lorsque le paramètre LTAG correspondant est non nul. LTAG est la longueur en mots de la chaîne de bits permettant de gérer l'allocation des granules de l'espace disque.
LTAG est mis à jour par la commande FUINI de FUP 4.

*
La table d'espaces est décrite de façon précise dans la notice IOCS. Il en est rappelé ici le contenu afin de comprendre le fonctionnement général des outils qui assurent le service de gestion des volumes.

Cette structure est initialisée par le programme de formatage d'un disque, et mise à jour à l'aide de l'utilitaire de définition de structure FUP 4 = SDEF. De plus elle est accessible par la SVC IOCS : Read structure, en lecture seulement, à tout processeur en mode esclave pour lui permettre de connaitre les informations qui caractérisent un support : par exemple son label.

Un booleen OFU permet au logiciel FMS - FUP de distinguer deux organisations physiques de FU disques :

L'organisation standard (OFU = 0) gérée par KERADR ; elle est adaptée à des disques de taille inférieure ou égale à 32 K secteurs, et à des fichiers de taille petite ou moyenne.

L'organisation grands disques (OFU = 1) gérée par KERGDI ; elle est adaptée à des disques dont la taille théorique logicielle peut atteindre environ 2 milliards de mots (31 bits), et à des fichiers de grande taille.

Le couple taille de secteur en mots et taille de cylindre en nombre de secteurs permet de calculer la taille du cylindre en mots sur ce type de disque et ensuite la taille en mots d'un espace disque connaissant sa longueur en nombre de cylindre. Cette nouvelle information apportée par la gestion de volume permet de faire des contrôles, ou par exemple de rendre optionnel les paramètres de la commande FUINI.

a) Le principe d'affectation FU-espace disque

Pour chaque Unité Physique ou emplacement d'unité physique, par exemple pour une cartouche, un système d'exploitation configuré sait gérer de 1 à 16 FU ordonnées.
— Il existe au moins 1 FU, la 1ère appelée la FU initiale : de rang 0.

*** 1 FU est** un nom (ex D8) choisi parmi les 28 FU gérées par FMS et FUP. Sans homonymie pour un système configuré.

Sur chaque support disque, par exemple une cartouche, est écrit par le formateur disque et FUP4 : SDEF, la structure du support constituée de 1 à 16 espaces disques ordonnés.
— Les espaces disque sont numérotés : NESP = 0 à 15.
— Il existe au moins 1 espace le 1er appelé l'espace initial NESP = 0.

— L'espace initial débute obligatoirement sur le cylindre zéro.
— L'espace initial est défini par le programme de formatage, de telle façon qu'il recouvre la totalité de l'espace formaté.

La longueur de l'espace initial est modifiable par la commande SDEF de FUP 4.
— Les différents espaces disques d'un même support peuvent se recouvrir pour ce qui est de leur utilisation par IOCS.

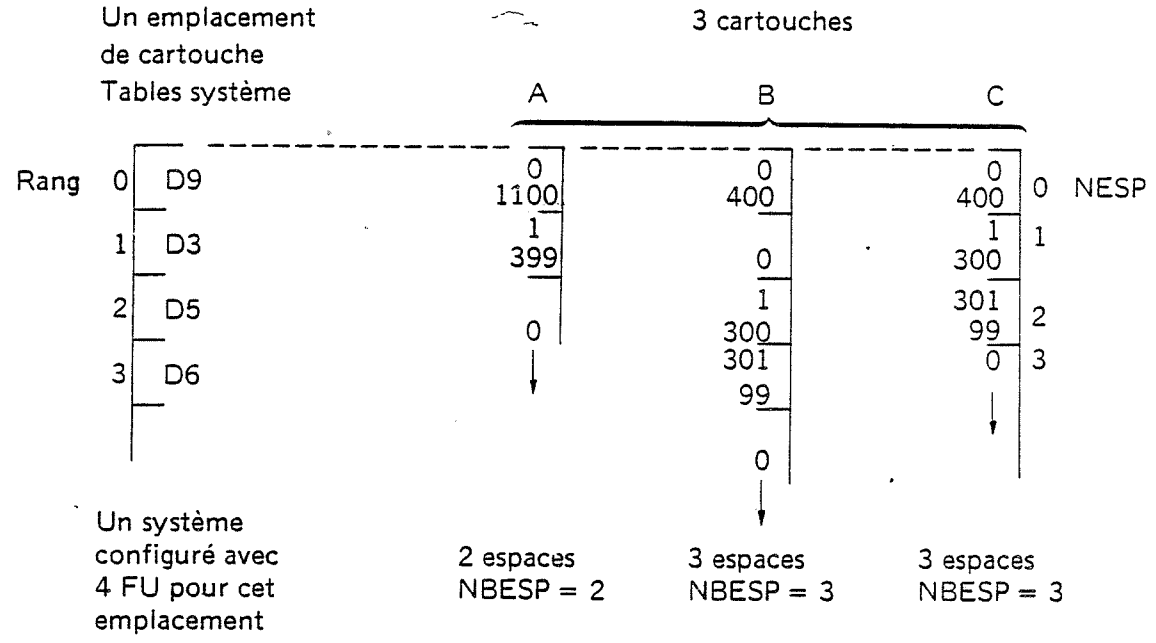
* **Un espace disque** est défini par la commande SDEF de FUP4.

— ADi : l'adresse du cylindre sur lequel commence l'espace disque.
— LONGi : la longueur de l'espace disque en nombre de cylindres.

*** Règle d'affectation FU-espace**

Le $i^{ème}$ espace disque est affecté à la $i^{ème}$ FU de l'emplacement.

*** Exemple : spécial pour une programmation sur FU**

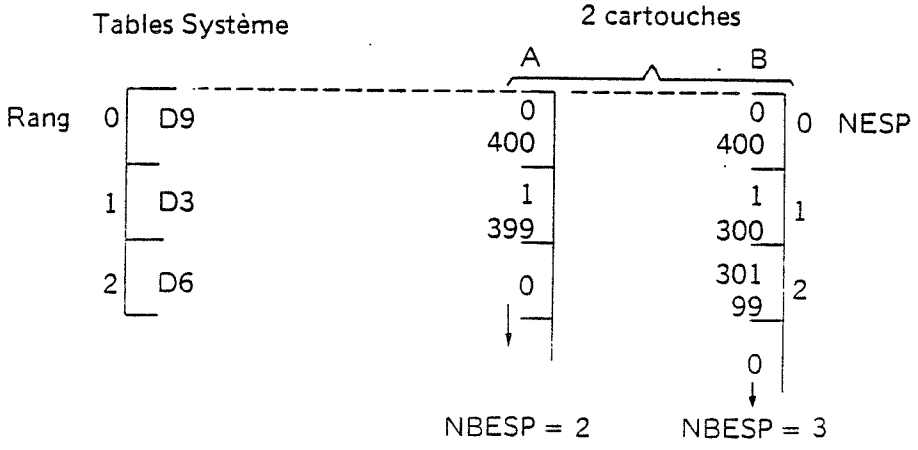


Remarque : l'association (FU adresse, long) est figée.

Le fait de laisser un vide dans une structure disque (cartouche B) permet de figer l'affectation FU-espace lorsque l'application est **programmée sur FU.** Pour que les espaces [1,300] [301 , 99] s'appellent respectivement D5 D6 ils doivent être respectivement de rang 2 et 3 dans la structure disque.

Exemple : standard de programmation portable

Lorsque l'application est **programmée sur SU** (symbolic Unit). La génération GENIO n'est plus liée aux couples (adresse, longueur) mais au nombre d'espaces.



Règle IOCS :

Pour chaque unité physique, les tables système IOCS doivent être générées avec un nombre de FU égal au plus grand nombre d'espaces définis (FUP 4 : SDEF) sur un support.

Règle FMS :

Pour chaque unité physique dans les tables système de FMS le descripteur de PU (DPU) doit être généré (% PU FMS) avec une longueur permettant de gérer le plus grand nombre de granules (PUNBGRAN) initialisés sur le ou les espaces FMS d'un support.

Dans l'exemple ci-dessus il s'agit de, soit pour la cartouche A le NBG de l'espace 1, soit pour la cartouche B la somme des NBG des espaces 1 et 2.

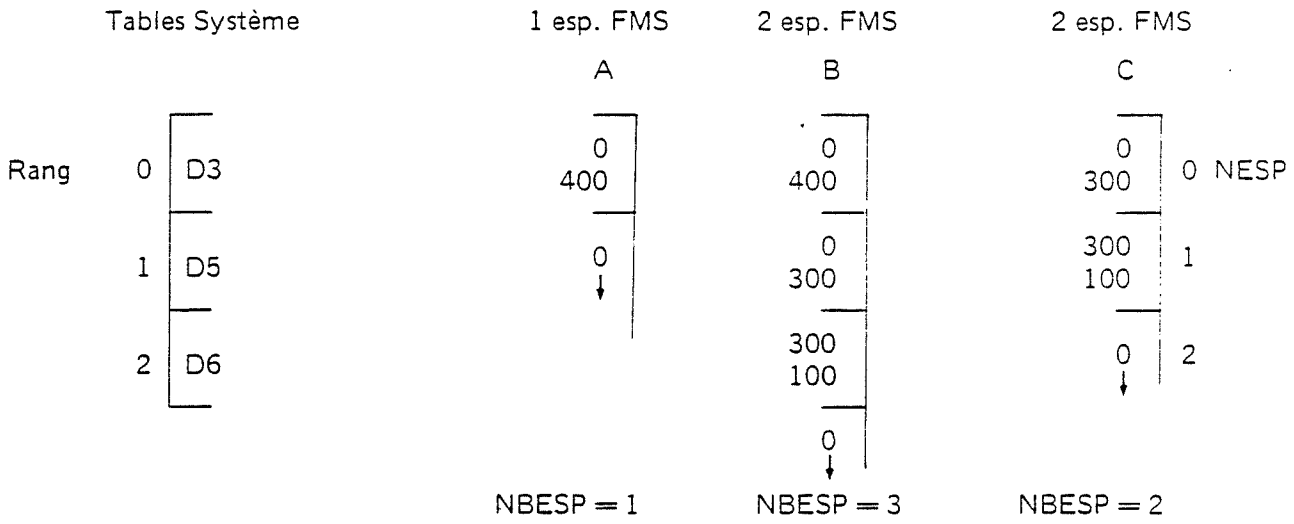
PUNBGRAN = max de (NBG_{A1} , $NBG_{B1} + NBG_{B2}$)

Pour la génération du DPU identifié par D9.

*

Exemple : Spécial avec l'organisation grand disque

L'organisation physique KERGDI permet de faire débuter un espace FMS sur le cylindre zéro.
La solution suivante gère 2 structures de disk-pack : 1 ou 2 espaces FMS



On peut remarquer sur le disk-pack B la présence de l'espace initial formaté de 400 cylindres permettant de traiter par IOCS tout le disk-pack.

On peut remarquer sur le disk-pack C un espace initial qui ne recouvre pas tout le disque. Cela permet de gagner une FU dans les tables d'IOCS. Il est possible d'utiliser l'espace initial par FMS avec l'organisation grand disque, en effet les informations FMS ne commencent que sur le secteur 8 ce qui laisse la place nécessaire pour RAPD et les tables de structure du volume.

b) Un espace disque géré par FMS et FUP

Lorsqu'un espace disque est géré par FMS, l'information LTAGi est différente de zéro. Elle a pour valeur la longueur en mots de la Table d'Allocation de Granules (TAG) construite sur cet espace disque par FUP4 : FUINI.

De plus NBFMS comptabilise le nombre de FU gérées par FMS sur l'ensemble du disque : cartouche, disk-pack.
Ces deux informations sont mises à jour par FUP4 : FUINI et lui permettent de contrôler le non-recouvrement des espaces disques gérés par FMS. Elles permettent également de reconfigurer dynamiquement les tables de FMS (TDFU) lors du montage d'un volume.
FMS possède pour cela deux requêtes CREATE FU et DELET FU exclusivement réservées aux Systèmes d'exploitation et décrites dans le manuel d'Utilisation de FMS. Elles sont utilisées par les superviseurs lorsqu'une commande de montage ou démontage de volume est émise par l'utilisateur ; dans un JOB par exemple.

La table des descripteurs de FU et d'unités physiques (TDFU) est décrite dans le Manuel d'Utilisation de FMS ainsi que sa génération par GFMS16.

Règle générale :

Une génération FMS permettant la reconfiguration dynamique identifie une Unité Physique par la FU initiale qui lui correspond dans la génération IOCS.

2.7 - LA GESTION DES GRANDS DISQUES

2.7.1 - Principe général

Le noyau grand disque KERGDI de FMS permet la gestion simultanée :

- des petits disques de taille inférieure à 4 Méga-mots
- des grands disques de taille supérieure à 4 Méga-mots avec une limite théorique logicielle à 2 milliards de mots.

KERGDI gère sur les grands disques des secteurs de taille variable. Le paramètrage pour FMS en est effectué à la génération d'un système pour chaque Unité Physique, au niveau des Unités Fonctionnelles correspondantes.
La longueur d'un secteur varie de 128 à 2048 mots et doit être une puissance de 2.
Cependant un FMS configuré (GFMS16) pour un système, gère au maximum 3 tailles secteurs différentes. On ne peut donc pas connecter à un même système plus de 3 classes de periphériques différents du point de vue de la taille secteur.
Les disques actuellement utilisés ont une taille de secteur figée à 128 mots (2^7).

La technique d'allocation dynamique mise en oeuvre par FMS sur les petits disques est également appliquée pour la gestion des grands disques.
Le noyau KERGDI distingue 2 types d'organisation :

1°) **L'organisation standard** : elle est décrite au chapitre 2.2.2. KERGDI les gère de la même façon que KERADR avec cependant 2 petites améliorations techniques :

- la possibilité de traiter des échanges FMS de plus de 6 K mots au niveau article (32 K maxi — 1) ou portion d'article (8 K maxi — 1) lorsque la taille des granules est supérieure à 6 K mots ; la limite IOCS sur les petits disques,
- l'Accès Séquentiel Rapide (ASR) : c'est-à-dire la faculté d'exploiter avec l'option ADR, la TLG en mémoire centrale pour réaliser en 2 accès disque au lieu de 3, la lecture ou la réécriture d'un article ou portion d'article, alignés secteur, mais à cheval sur 2 granules,
- disques avec ou sans gestion de volume (gestion d'espace).

2°) **L'organisation grands disques** : elle est décrite dans les chapitres qui suivent.

- l'organisation physique des grands disques est adaptée à la gestion d'espaces disque de 2 milliards de mots. Elle permet en particulier la gestion des adresses disque sur 2 mots (31 bits).
- l'organisation physique des grands disques gérée par FMS permet de stocker dans un espace disque des fichiers statiques de 2 milliards de mots. Cette fonctionnalité est utilisable par les méthodes d'accès Direct, Direct à Trous, Séquentiel Indexé.
  Deux techniques permettent d'atteindre cet objectif, à savoir : la gestion de granules de 3 à 32 K secteurs, ainsi que la gestion de 1 à 32656 granules par espace disque.
- Le choix du type d'organisation est fait par l'utilisateur lorsqu'il initialise un espace par FUINI de FUP4.
  Org. STD : OFU= 0 Org. GDI : OFU= 1
- Sur un même disque, les différents espaces peuvent être initialisés avec des organisations différentes : STD ou GDI.
- Attention : l'organisation grand disque GDI ne fonctionne que sur les disques avec gestion d'espace.

*

Remarque importante :

En ce qui concerne les trois points mentionnés ci-dessous KERGDI offre un service identique à celui de KERADR.

L'organisation physique définie pour les grands disques est opérationnelle, c'est-à-dire **performante** dans les cas suivants :

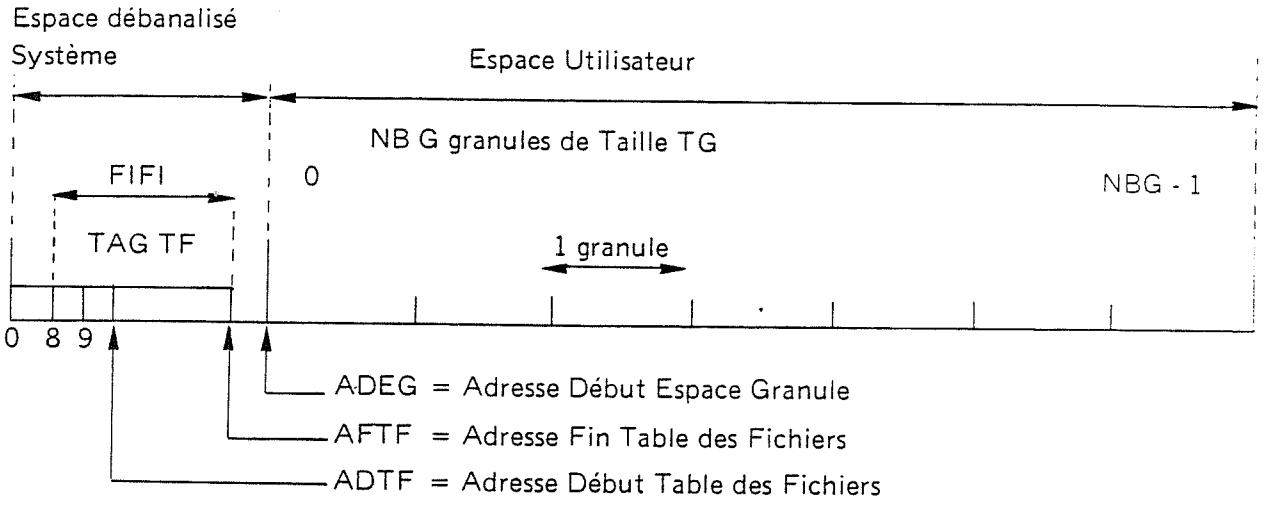
— FMS gère de 1 à 500 fichiers par Unité Fonctionnelle, la limite étant le nombre de granules défini sur le support.

— FMS gère environ 10 000 fichiers en ligne dans la mesure où la capacité disque le permet. Soit 28 Unités Fonctionnelles de 370 fichiers.

— FMS permet d'ouvrir simultanément 50 à 100 fichiers.

Autrement dit l'objectif principal de KERGDI est d'être capable de gérer sur un grand disque, quelques grands fichiers plutôt qu'un grand nombre de petits fichiers.

2.7.2 - Organisation physique d'une Unité Fonctionnelle Grand Disque



L'espace disque d'une Unité Fonctionnelle (FU) est divisé en deux parties :

- L'espace système est implanté à partir du début de la FU et contient le Fichier des Fichiers (FIFI) c'est-à-dire principalement, la table d'Allocation de Granule (TAG) et la table des Fichiers (TF).
- L'espace utilisateur est découpé en granules, il est implanté à partir du fond de la FU.

Cette organisation physique a deux objectifs :

- Lorsque la FU ne contient qu'un seul fichier, la taille du granule doit être maximale et la place occupée par FIFI minimale. Exemple limite : NBG = 1 granule et FIFI occupe 3 secteurs. Cette solution était impossible avec l'organisation physique des petits disques, car FIFI occupait au moins 1 granule.
- L'espace granule est cadré à la fin de la FU pour permettre d'optimiser la taille granule en rapport avec la taille cylindre.

Remarque :

La limite entre l'espace système et l'espace granule est définie par la Taille de l'espace granule. Autrement dit logiquement on implante d'abord l'espace granule au fond de la FU. Il reste alors au début une place débannalisée pour l'espace système qui doit être suffisante pour contenir FIFI, et de plus inférieure à 32 K secteurs, pour que l'information ADEG soit une adresse secteur disque exprimable sur un mot (15 bits).

Règles :

(1) Taille FU = Taille Espace Granule + Taille Espace Système

(2) LFIFI + 8 secteurs $\leqslant$ Taille Espace Système $<$ 32 K secteurs.

Remarque importante :

L'organisation physique des grands disques se présente sous 2 formes différentes :

- l'une pour les **petites FU**, c'est-à-dire dont la taille en nombre de secteur est inférieure ou égale à 32 K secteurs. Dans ce cas toutes les adresses secteur sont exprimables sur 1 mot (15 bits).
- l'une pour les **grandes FU**, c'est-à-dire dont la taille en nombre de secteur est supérieure à 32 K secteurs. Dans ce cas toutes les adresses physiques sont exprimables sur 2 mots (31 bits).

Les 2 formes sont choisies automatiquement à l'initialisation d'une FU par FUP4 : FUINI.

Elles permettent de minimiser la taille des informations système sur disque et en mémoire centrale lorsque l'on gère des petites FU sur un grand disque.

Structure de l'espace débanalisé système

FIFI commence au secteur d'adresse 8 dans la FU. Les 8 premiers secteurs de la FU sont réservés en particulier à RAPD lorsque la FU commence au cylindre zéro d'une unité de disque.
FIFI occupe un nombre entier de secteurs, variable selon le nombre de granules et le nombre de descripteurs de Fichiers que l'on peut stocker dans la Table des Fichiers. FIFI occupe au minimum 3 secteurs.
La place disque située entre la fin de FIFI et le début de l'espace granule est inutilisée. Cette place perdue est de longueur nulle si la taille du granule est un sous-multiple de la taille de la place restante dans la FU pour l'espace utilisateur.

La taille totale de l'espace débanalisé système est :

* au minimum = 8 + 3 = 11 secteurs
* au maximum = 2^{15} secteurs.

2 7.3 - Description d'un granule

Dans une petite FU grand disque, la structure d'un granule est identique à celle d'un petit disque. Dans une grande FU grand disque la seule différence est dûe au fait que les ligatures du granule sont exprimées sur 2 mots.

- LAM = Ligature Amont = Adresse secteur sur 31 bits.
  Le 1er mot contient les 15 bits poids fort, le 2ème mot contient les 16 bits poids faible. Le bit zéro du 1er mot est à zéro.

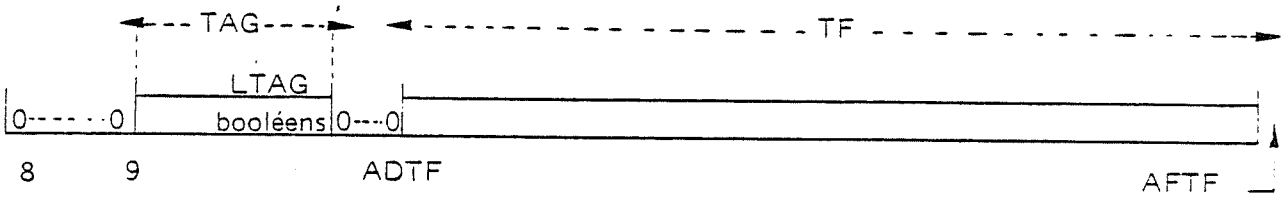
- LAV = Ligature Aval = idem.

Règle :

soit TG la taille du granule en nombre de secteurs :

$$3 \leqslant TG \leqslant 2^{15} - 1 \quad \text{secteurs}$$

2.7.4 - Description de FIFI



ADTF = Adresse de Début de la Table des Fichiers.
AFTF = " " Fin " " " " "

FIFI est organisé en 3 parties :

— le 1er secteur est inutilisé. Il peut servir au logiciel d'application qui y accèderait par IOCS pour y stocker par exemple un Label au niveau de la FU-support alors que FMS-E et FMS-G gèrent des labels au niveau du support physique : cartouche, pack, etc...
Le 1er secteur est initialisé à zéro par FUINI.

— La TAG (Table d'Allocation de Granules) contient :
Des informations système (7 mots) telles que la taille du granule TG, le nombre total de granules NBG, etc...
La chaine de bits d'allocation des granules, la TAG occupe à partir du secteur d'adresse 9 un nombre de mots n1 tel que :

LTAG = (NBG / 16) ÷ 1 si reste ≠ 0
n1 = 7 ÷ (NBG/16) ÷1 si reste ≠ 0.
de plus soit LSEC la longueur du secteur et LTAGS le nombre de secteurs occupés par la TAG
LTAGS = (n1 / LSEC) ÷ 1 si reste ≠ 0.

— La TF (Table des Fichiers) contient la suite des descripteurs des fichiers (DF) **permanents** de la FU.
Elle débute sur le 1er secteur qui suit la TAG.
Elle occupe un nombre entier de secteurs LTF tel que : soit NFIC le nombre de fichiers à gérer sur petite FU grand disque :

$$LTF = \frac{NFIC}{LSEC / 8} \quad \div 1 \text{ si reste} \neq 0$$

Sur une grande FU grand disque :

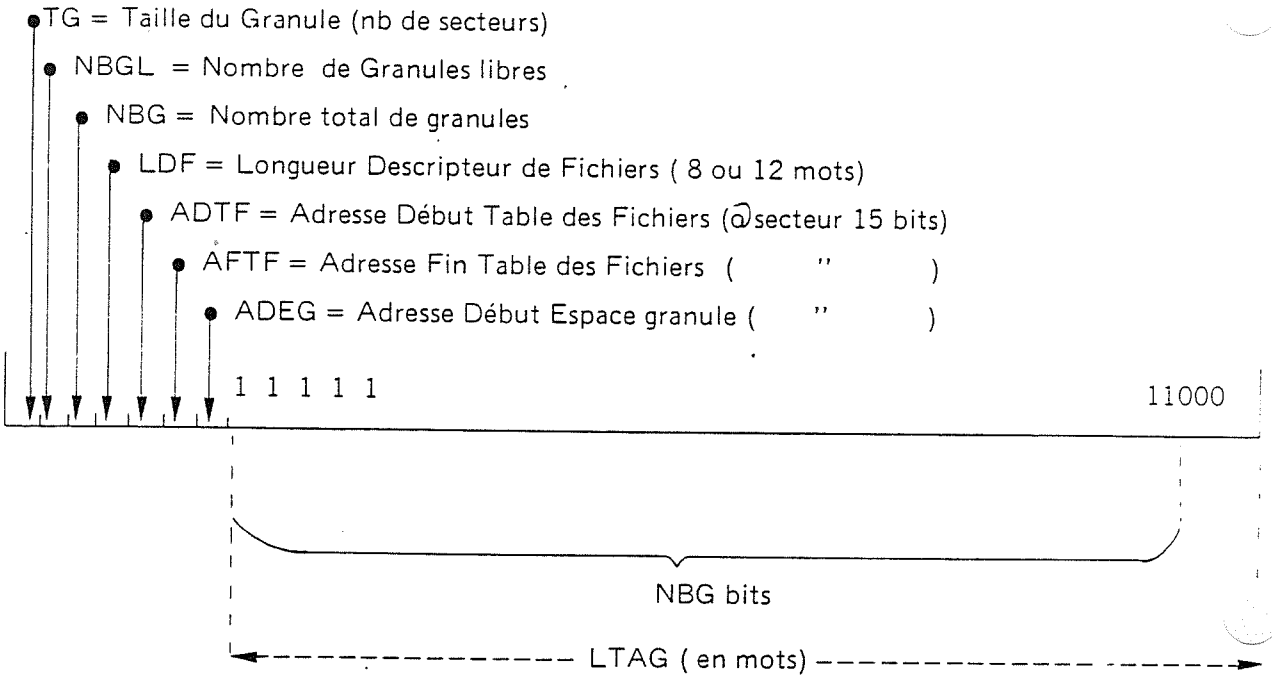
$$LTF = \frac{NFIC}{LSEC/12} \quad \div 1 \text{ si reste} \neq 0$$

De plus :

AFTF = ADTF + LTF

— FIFI : le fichier des fichiers
Taille totale au sens FUINI : LFIFI = 1 ÷ LTAGS + LTF
Taille utile au sens FDESCRIPT = LFIFI — 1

2.7.5 - Table d'Allocation de Granules : TAG



Le schéma ci-dessus décrit la chaine de bits après initialisation par FUP4 : FUINI.

Les derniers bits du dernier mot sont initialisés à zéro.

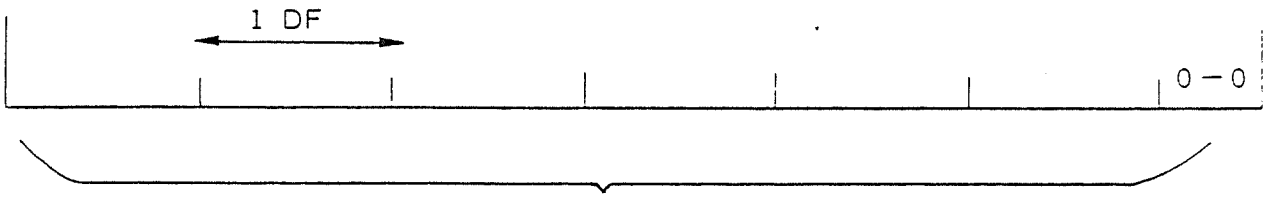
Règles .

- $3 \leqslant TG \leqslant 2^{15} - 1$ secteurs
- $0 \leqslant NBGL \leqslant NBG \leqslant 32656$ granules
- LDF = 8 ou LDF = 12 mots
- $10 \leqslant ADTF \leqslant 26$ @secteur
- $11 \leqslant AFTF \leqslant 3292$ "
  $11 \leqslant ADEG \leqslant 2^{15} - 1$ "

2.7.6 - Table des Fichiers (TF)

La TF des grands disques possède la même organisation que dans les petits disques. Elle est initialisée à zéro par FUP4 : FUINI.
Un DF supprimé par le DELET d'un fichier permanent est marqué par 'FFFF dans le 1er mot.
La seule différence réside dans le fait que la taille du DF est de 8 mots dans une petite FU grand disque et 12 mots sur une grande.

Un secteur de TF dans une grande FU grand Disque :



$$\text{n DF par secteurs : } n = \frac{LSEC}{12}$$

Par exemple : n = 10 pour LSEC = 128 mots, reste 8 mots nuls.

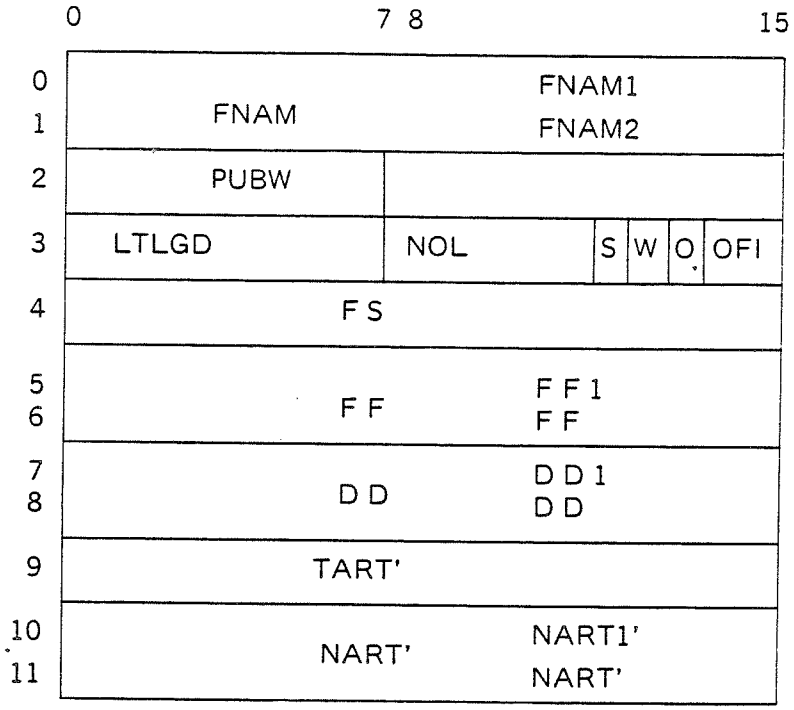
Remarque :

Dans une petite FU Grand disque, la structure d'un DF 8 mots est identique à celle d'un DF petit disque Sauf le FTYP contenant NOL, S, W, OFI qui est organisé comme dans un DF de grande FU. NOL = EMA ⊕ SID occupe les bits 8 à 11 inclus.

Remarque :

La commande FUINI de FUP4 (Indice d'évolution supérieur ou égal à 40) crée un fichier de nom FIFI — : S permettant d'accéder à FIFI : par FMS (OPEN OLD, READ, CLOSE). Cela permet donc de lire en séquentiel, la Table d'Allocation de Granules (TAG) et la Table des Fichiers (TF).
Le fichier FIFI — : S est Direct, Simultané et protégé en écriture. Il possède une organisation physique séquentielle (OFI = 0). La taille des articles est égale à la taille secteur et le nombre d'articles égal au nombre de secteurs utiles de FIFI.

2.7.7 - Le Descripteur de Fichier grande FU (LDF = 12 mots)



FNAM	Valeur en Radix 40 6 caractères $< 40^3$	16 bits
PUBW	" " 2 caractères $< 40^2$	11 bits
LTLGD	Nombre de granules d'un fichier statique (OFI = 1 $1 \leqslant LTLGD \leqslant 128$	8 bits
NOL	Numéro d'organisation logique $0 \leqslant NOL \leqslant 8$	4 bits
S	Fichier Permanent Pargageable Simultanément S = 1	1 bit
W	" autorisé en écriture (W = 1)	1 bit
OFI	Fichier Statique si OFI = 1	
FS	Déplacement dans le dernier secteur $0 \leqslant FS < LSEC$	15 bits
FF	Adresse du dernier secteur du fichier	31 bits
DD	Adresse du premier granule du fichier	31 bits
TART'	Taille de l'article, poste, ou index en mots	15 bits
NART'	Nombre d'articles, poste, ou index	31 bits

2.7.8 - L'organisation physique d'un fichier

a) Fichier Dynamique

L'organisation physique d'un fichier dynamique est dite séquentielle. Elle est constituée comme pour les petits disques de :

— **Un descripteur de fichier** (DF) contanant en particulier :
  * DD : l'adresse de début du fichier
  * FF, FS : l'adresse de fin du fichier

— **Une chaine de granules** telle que chaque granule contient 2 ligatures.
  * LAM : Adresse du granule amont
  * LAV : Adresse du granule aval.

Les adresses physiques DD, FF, LAM, LAV sont des adresses secteur de 15 bits sur les petites FU et de 31 bits sur les grandes.

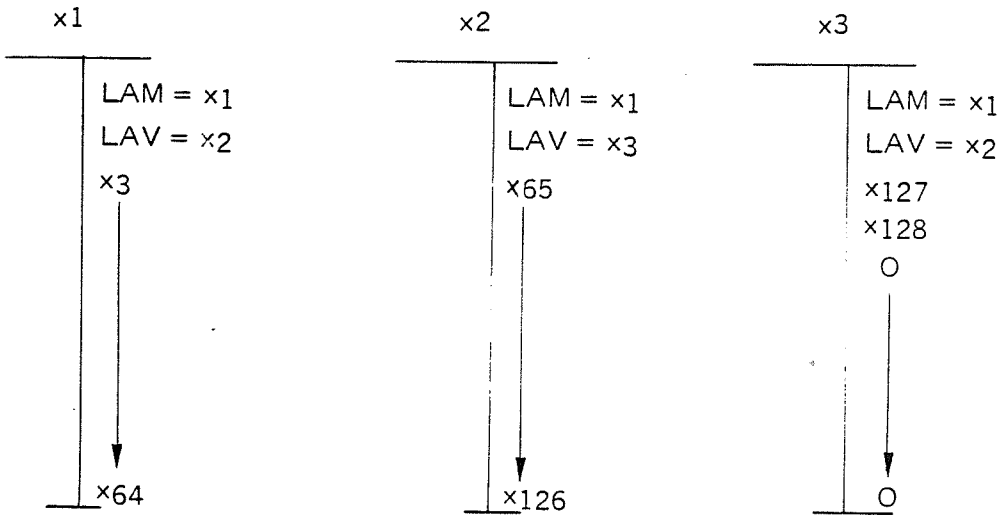
b) Fichier Statique

L'organisation physique d'un fichier statique est dite directe. Elle est constituée comme pour les petits disques de :

— **L'organisation physique séquentielle**
— **Une table des ligatures de granules** (TLG) se trouvant dans le secteur système des premiers granules du fichier.

Dans les grands disques un fichier statique occupe au maximum 128 granules. L'objectif recherché est de minimiser la taille d'une TLG lorsqu'elle est en mémoire centrale avec l'option ADR. En conséquence la taille du granule doit être grande, voir maximale, pour stocker des grands fichiers sur disque.

Une TLG contient des adresses secteur sur 15 ou 31 bits selon la taille de la FU. Elle occupe au maximum les secteurs système des 3 premiers granules du fichier.



Remarque : Le choix de 126 granules pour un fichier statique dans une grande FU avec 128 mots de taille secteur minimise le nombre d'accès disque en accès direct standard

3 — DESCRIPTION GENERALE

3 — DESCRIPTION GENERALE

Le lecteur trouvera dans ce chapitre et ceux qui suivent la description de l'interface de programmation des requêtes (SVC) à FMS, et la description de leur fonctionnement externe.

Le lecteur trouvera plus précisément dans ce chapitre les conseils généraux de programmation d'une requête à FMS.

3.1 - UNE REQUETE A FMS : INTERFACE 64 K

3.1.1 - Les requêtes FMS.

FMS est activé par un ensemble de requêtes chacune traitant une catégorie de Services.
— Une requête pour le niveau Fichier : OPEN-CLOSE.
— Une requête pour chaque type de Fichier.
Par exemple il existe une requête SVC (FMSD) ; où FMSD = `3B, qui fournit l'ensemble des services d'accès aux articles d'un fichier DIRECT.
Une requête programmée correspondant à une instruction machine appelée SVC : Appel Superviseur.
Une instruction machine SVC est recueillie par le superviseur (partie SVC Handler) qui lui-même appelle FMS.

Tableau des requêtes FMS-E

SVC	Valeur hexadécimale	OPEN CLOSE et METHODES D'ACCES	N° org. Logique
FMS	`38	OPEN CLOSE Général	—
FMSS	`39	SEQUENTIEL Bufférisé	0
FMSI	`3A	INDEXE Bufférisé	1
FMSD	`3B	DIRECT et DIRECT A TROUS	2
FMSX	`28	SEQUENTIEL INDEXE	3
FMSC	`29	SEQUENTIEL CHAINE	4
FMSV	`2A	DIRECT LONGUEUR VARIABLE	5
	`2B		6
	`2C		7
	`2D		8

3.1.2 - Programmation d'une requête

Les requêtes d'accès aux fichiers peuvent se programmer dans différents langages : Assembleur, PL1600, FORTRAN, ... Le principe en est le même pour tout langage.

— Pour chaque requête adressée à FMS, l'utilisateur doit fournir un ensemble de paramètres qui définissent très précisément quel traitement doit être effectué.

— Par exemple lorsque l'utilisateur désire lire l'article de nom DUPONT, il doit bien évidemment fournir entre autre à FMS cette information : DUPONT

— Les paramètres d'une requête à FMS sont physiquement stockés en mémoire centrale dans une table de commande appelée : FCB (File Control Block).
Le programme appelant transmettra donc en fait à FMS l'**adresse** du FCB.

— La programmation d'une requête à FMS dépend du langage utilisé par le programmeur d'application. Il faudra donc éventuellement se reporter à une notice Langage (ex : Manuel de Référence FORTRAN). Dans la présente notice sera décrite la programmation en PL1600 d'une requête à FMS.

— **Programmation en PL1600**

1°) Chargement des paramètres de la requête dans le FCB correspondant.

2°) Appel : exemple d'appel à la méthode d'accès DIRECT.

```
RA : = @ FCB ;
SVC     (FMSD) ;          <<FMSD = `3B
→ retour à l'appelant.
```

3°) Test du compte-rendu de requête (PR) transmis par FMS à l'utilisateur dans le 4e mot du FCB (mot 3) et dans le registre A.

— **Remarques**

1o) FMSD est une constante initialisée à la valeur du numéro de requête correspondant.

2o) En PL16 ne pas oublier de déclarer l'instruction SVC.

3o) Toutes les adresses que l'utilisateur transmet à FMS sont des nombres absolus sur 16 bits (pas de bit Index) et :
— en mode esclave relatives à SLO.
— en mode maître des adresses absolues.

4o) Le retour à la Tâche appelante se fait normalement en fin d'échange. Cependant, dès qu'une anomalie est détectée, FMS rend la main à la tâche en lui signalant l'erreur à l'aide du compte-rendu : PR.

3.1.3 - L'interface SVC : appel au superviseur

a) **Les registres**

Tous les registres de l'utilisateur sont restitués à la valeur qu'ils avaient au moment de l'appel SVC. Sauf :

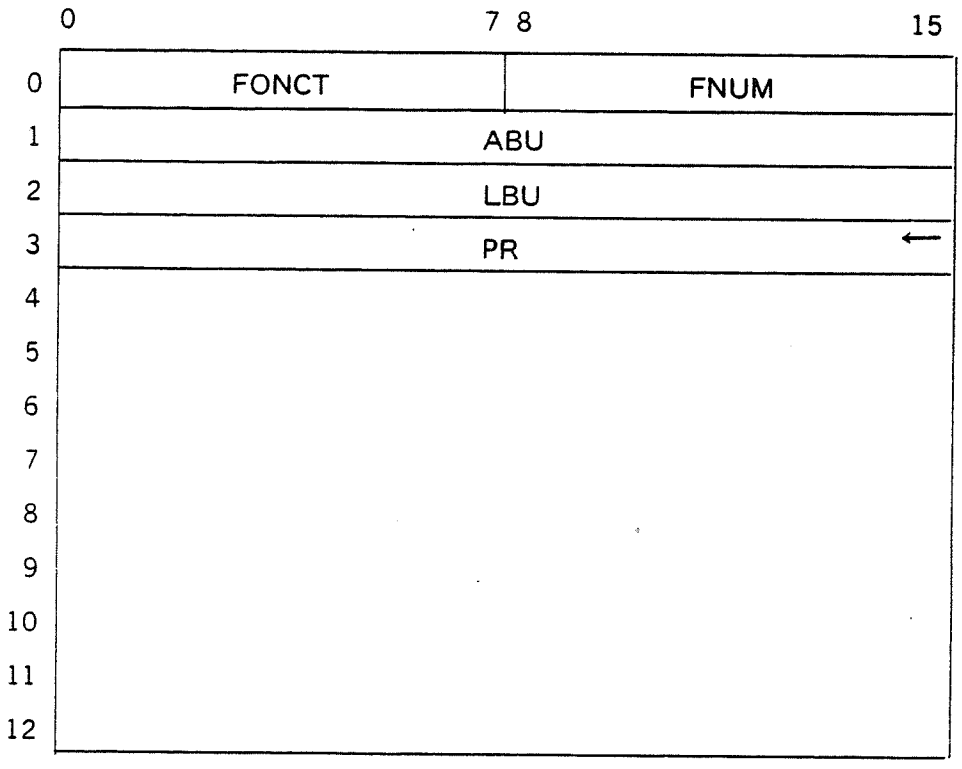
A : en entrée = l'adresse du FCB (nombre absolu sur 16 bits)
en sortie = PR : le compte-rendu de traitement de requête.

X : est détruit par l'appel au superviseur.

S : les indicateurs V et C sont détruits. Le bit Maître/Esclave est restitué.

b) **Le FCB : File Control Block**

Dessin général d'un FCB



Un FCB occupe **13 mots.** FMS contrôle, lorsque le programme appelant est en mode esclave que les 13 mots sont bien dans la zone utilisateur définie par [SLO SLE], et en mode maître que les 13 mots sont bien " dans " la mémoire centrale du calculateur (comparaison par rapport à SYSMEM).

SYSMEM = mémoire débanalisée : `C

Les paramètres d'un FCB sont chargés avant l'appel à FMS. Pendant le traitement de la requête correspondante par FMS, le FCB ne doit pas être modifié (adresse, contenu) par une autre tâche de la machine. Il en est de même pour toutes les zones de mémoire utilisateur, également utilisées par FMS : pile de l'appelant (Kstore), zone d'échange, buffer de Travail.

Le paramètre FONCT permet à l'utilisateur de spécifier pour une requête donnée, quelle prim' devra être exécutée par FMS.

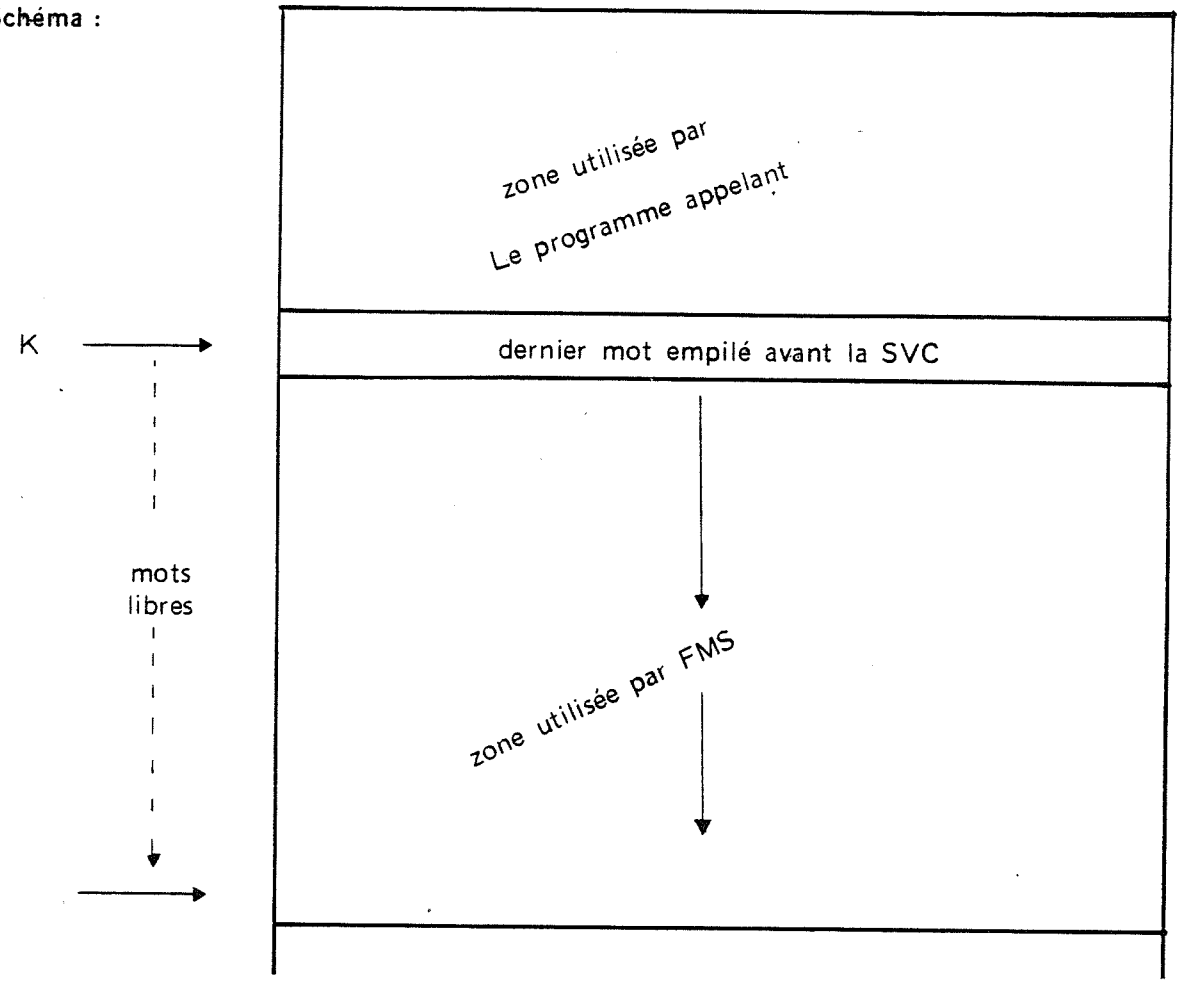
Exemple :

```
FONCT :  =  `01   ;     « primitive de lecture d'un article
RA    :  = @FCB   ;
SVC      (FMSD)   ;     « d'un fichier DIRECT
```

En règle générale FMS n'écrit rien dans un FCB. Le PR est chargé dans le FCB par FMS sauf dans le cas où l'adresse du FCB est déclarée invalide, auquel cas le compte-rendu se trouve seulement dans le registre A. Cependant pour certaines méthodes d'accès FMS fournit à l'utilisateur, dans le FCB, des informations (ex : n° d'un article lors de sa création dans un fichier Direct à Trous).

c) **La Kstore : pile de la tâche appelante**

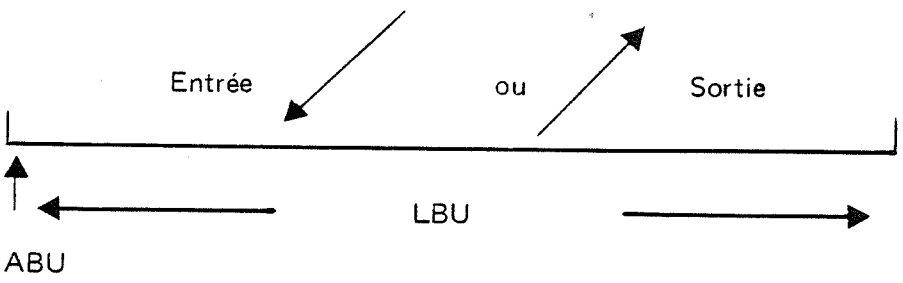
Schéma :



Selon que la machine et le système supportent ou non le mode privilégié, FMS est soit intégré au système, soit déporté dans une autre partition.
La taille de la KSTORE de la tâche appelante doit être telle que, à chaque appel SVC il reste 70 mots utilisables si FMS est intégré, 10 mots si FMS est déporté.
Remarque : Le superviseur lui-même empile quelques mots avant l'appel à FMS.
Voir pour cela la notice du superviseur utilisé.

d) **La zone d'échange**

Schéma d'une zone d'échange



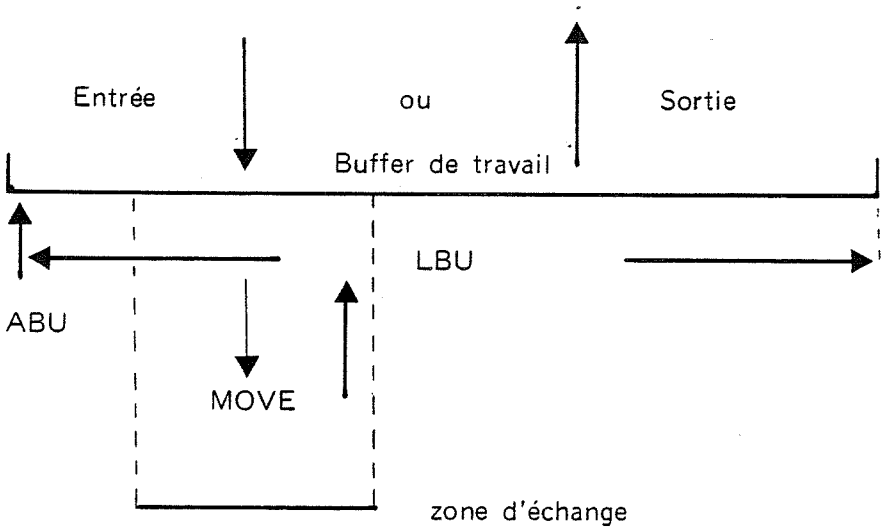
La zone d'échange reçoit ou fournit les informations (article ou Portion d'Article) que l'utilisateur désire lire ou stocker sur la mémoire secondaire disque.

L'implantation d'une zone d'échange en mémoire centrale est définie par deux paramètres contenus dans le FCB :
- ABU = adresse (nombre absolu sur 16 bits)
- LBU = longueur. Une longueur est toujours exprimée en octets bien que FMS ne sache traiter que des **mots**. Cette longueur si elle est acceptée impaire par FMS, est toujours apairée au mot supérieur.
  Lors de la création de certains articles ou portions d'articles elle en définit la longueur et, dans les autres cas la quantité d'information (nombre de mots) que l'utilisateur désire lire ou écrire dans un article.

Une fois que l'exécution d'une requête est **terminée** l'usager peut utiliser la zone d'échange (FCB également) pour un autre traitement.

e) **Un buffer de travail**

Schéma d'un buffer de travail



Un buffer de travail est une zone de mémoire centrale que l'utilisateur " prête " à FMS et plus précisément à une **Unité d'Accès** (FAU) à un Fichier pour qu'elle réalise et/ou optimise les échanges avec le disque.
Un buffer de travail est géré **exclusivement** par FMS. **L'utilisateur ne doit pas le modifier** (adresse, contenu) **pendant toute la vie de la FAU** qui lui correspond :
Exemple : [OPEN OLD CLOSE]
Remarque : Sous RTES16 et BOS16 faire attention à la programmation des tâches non résidentes. Il leur est conseillé de fermer les fichiers avant de quitter le système (CLOSE).

Les principales méthodes d'accès utilisant un buffer de travail :
- **Séquentiel**
- **Direct**
- **Indexé**
- **Séquentiel Indexé** (obligatoire)
- **Séquentiel Chaîné** (obligatoire)

3.1.4 · PR : Le compte-rendu de requête

Valeur de PR	Signification
0	**Primitive correctement exécutée** — Pour la lecture ou l'écriture d'un article cela signifie que LBU (apairé) est bien la longueur de l'article connu de FMS dans le fichier disque. Toute la zone d'échange est chargée (LBU [+ 1] /2 mots).
0 ⩽ PR⩽ `3FFE	**Primitive correctement exécutée** — Cette forme de compte-rendu est fournie par les requêtes adressées à la méthode d'accès : **Séquentiel** (READ, WRITE, SKIPB, SKIPF). Elle fournit le compte d'octets (pair) effectivement transféré ou sauté. — Il se peut que le PR soit inférieur au compte (LBU) demandé. Cela signifie que la fin ou le début de l'article a été rencontrée.

Les Avertissements

`6001	**Fin d'article** — Le compte-rendu signifie fin d'article et/ou fin de fichier. — Cela signifie que le pointeur courant désignait déjà la fin de l'article, avant l'exécution de cette requête. — La requête est ineffective.
`6002	**Début d'article** — Le compte-rendu signifie début d'article et/ou début de fichier. — Avant l'exécution de cette requête le pointeur courant désignait déjà le début de l'article. — La requête est ineffective.
`6003	**Article du fichier plus long** — L'article du fichier est plus long que la zone d'échange spécifiée par le FCB. — En lecture la zone d'échange entière est remplie avec le début de l'article. — En écriture toute la zone d'échange est transférée dans l'article (cadré à gauche). La fin de l'article est laissée inchangée.
`6004	**Article du fichier plus court** — L'article du fichier est plus court que la zone d'échange spécifiée par le FCB. — En lecture tout l'article est placé dans la zone d'échange. Il est cadré à gauche, et le reste de la zone d'échange est inchangée. — En écriture l'article entier est reécrit avec le début de la zone d'échange.

`6005 **Article incorrect**

— Le contenu de l'article contrôlé par FMS-E est incorrect. Voir méthode d'accès.
— La requête n'est en général pas ineffective.

`6006 **Fin de chaîne**

— Avant l'exécution de cette requête le pointeur logique courant désignait déjà la fin de chaîne.
— Voir méthode d'accès.

`6007 **Début de chaîne**

— Avant l'exécution de cette requête le pointeur logique courant désignait déjà le début de chaîne.
— Voir méthode d'accès.

Les erreurs

`600A **FAU inexistante**

— Le FNUM précisé dans le FCB n'est le numéro d'aucune FAU créé par cet usager (USR fournit par le superviseur à chaque requête).
— Cette erreur arrive fréquemment lorsque l'adresse du FCB est incorrecte.
— La requête est ineffective.

`600B **FAU existante**

— Le FNUM précisé dans le FCB correspond à une FAU déjà créé par cet usager (USR fournit par le superviseur pour chaque requête).
— Cette erreur peut arriver lorsque l'adresse du FCB est incorrecte.
— La requête est ineffective.

`600C **Fichier inexistant**

— Il n'existe pas de fichier identifié par : FNAM + PUBW + SU/FU
— Cette erreur peut arriver si l'affectation SU ⟶ FU est incorrecte, ou si le bon support n'est pas en ligne.
— Cette erreur peut éventuellement arriver si la FU-Support n'est pas initialisée. Vérifier le contenu de la FU-Support à l'aide de FUP2 (FUST ; FLIS ; FDES).
— La requête est ineffective.

`600D **Fichier existant**

— Il existe déjà un fichier temporaire identifié par : FNAM + USR + SU/FU
— Il existe déjà un fichier permanent identifié par : FNAM + PUBW + SU/FU
— Cette erreur peut arriver si l'affectation SU ⟶ FU est incorrecte, ou si le bon support n'est pas en ligne.

— La requête est ineffective.

`600E **Article inexistant**
— Le fichier accessible par cette FAU (FNUM) ne contient pas l'article dont l'identificateur, numérique ou alphanumérique, a été fourni dans le FCB ou la zone d'échange.
— La requête est en règle générale ineffective, voir méthode d'accès, en particulier pour les problèmes de sélection d'article.

`600F **Article existant**
— FMS réalise un contrôle de non-homonymie, et dans le fichier accessible par cette FAU (FNUM), il existe déjà un article de même identificateur.
— La requête est en règle générale ineffective, voir méthode d'accès, en particulier pour les problèmes de sélection d'article.

`6014 **Protection écriture**
— Lors de la création d'une FAU
Soit le fichier est protégé en écriture W = 0
Soit le fichier est NON Simultané et ni Direct ni Direct à Trous et l'utilisateur précise une RWK = 3 ou une RWK = 1 avec demande d'écriture.
— La FAU désignée par le paramètre FNUM ne permet que de lire sur le fichier correspondant.
— La requête est ineffective.

`6015 **Permanent de nature différente**
— La nature de permanent, simultané ou NON simultané, précisée dans le FCB à l'aide du bit S n'est pas celle définie à la création du fichier.
— La requête est ineffective.

`6016 **Fichier saturé**
— Le nombre maximum d'articles que le fichier est capable de contenir est atteint.
— La requête est en règle générale ineffective. Voir méthode d'accès, en particulier pour les problèmes de sélection d'article.

`6017 **Fichier trop long**
— En création : le Fichier est créé par FMS avec une organisation physique directe et compte tenu des paramètres TART et NART du FCB et de la Taille du granule sur la FU-Support choisie (SU/FU), le fichier occuperait plus de 128 granules.
— A la création d'un article : FMS réalise l'allocation dynamique d'un article dont la taille ne doit pas dépasser 32 K mots.
— Pour une création de fichier la requête est ineffective.
— Pour la création d'un article voir méthode d'accès.

`6018 **Incompatibilité primitive fichier :**

— Au niveau fichier : le type du fichier, Temporaire ou Permanent, n'est pas celui attendu selon la primitive (ex:CATAL).
— Au niveau article : le Type du fichier c'est-à-dire son organisation logique (EMA/SID) ne peut pas être géré par la méthode d'accès utilisée. (SVC (FMS .) ; numéro de requête = FMS.)
— Au niveau article, l'erreur arrive fréquemment lorsque le FNUM est incorrect.
— La requête est ineffective.

`601A **Erreur d'enchaînement**

— En règle générale cette erreur est liée au mécanisme de sélection d'article. Cela signifie qu'aucun article n'est sélectionné par cette FAU, soit parce que la requête de sélection n'a pas été lancée, soit parce qu'elle s'est terminée anormalement.
— Cette erreur peut arriver si le FNUM est incorrect.
— Dans les autres cas : voir méthode d'accès.
— En règle générale la requête est ineffective voir cependant la méthode d'accès.

`601E **Fichier occupé :**

— Les conditions de Partage Simultané ou de Multi-Accès dans lesquelles se trouve le fichier à cet instant, ne permettent pas d'autoriser les actions que l'usager demande à entreprendre sur le fichier.
— Les actions sont définies par le type de requête. Exemple : CATAL, DELET, ou dans la requête OPEN OLD par les paramètres W et RWK.
— Le fichier est désigné par la FAU correspondante FNUM, ou par son identification FNAM + PUBW + SU/FU.
— La requête doit donc respecter les règles communes du Partage des fichiers. Voir requête correspondante ou chapitre 2.4 Introduction : Le partage des accès aux fichiers.
— La requête est ineffective.
— Remarque : FMS ne met pas l'usager en attente. C'est à lui d'utiliser les services du système d'exploitation et éventuellement de se mettre en accord avec les autres usagers.

`601F **Primitive en cours**

— Ce constat d'erreur peut arriver du fait que dans un contexte Temps Réel, 1 usager peut être constitué de plusieurs tâches. Toutes les tâches d'un même usager peuvent donc utiliser les mêmes chemins d'accès aux Fichiers (FAU).
— Cela signifie qu'au moment de l'exécution de cette requête une primitive était en cours pour le compte d'une autre tâche mais sur cette même FAU : USR + FNUM
— La requête est en général ineffective. (Voir l'interface de la requête)

Erreurs filtrées par les superviseurs

`6020 **Zone de Pavés saturée**
— La ressource manquante à cet instant est un nombre n de pavés la zone système : ZDF
— Cette requête de création d'une FAU demande un nombre n de pavés en fonction de :
  — la méthode d'accès correspondante : 1 ou 2 (DF)
  — des conditions de partage
  — de l'option de performance : Accès Direct Rapide
— Recalculer pour la zone ZDF, le nombre de pavés nécessaire à cet instant là et regénérer un système d'exploitation. Pour dimensionner la ZDF voir manuel d'utilisation chapitre 3
— La requête est ineffective.

`6021 **FU saturée**
— La FU-Support désignée par SU/FU ne contient pas à cet instant, un nombre de granules libres suffisant pour exécuter la requête.
— Pour une création de fichier la requête est ineffective.
— Pour une allocation dynamique il se peut que la requête soit partiellement exécutée. Le fichier est cohérent du point de vue de FMS et donc son contenu peut être relu.
— Cette erreur peut arricer lorsqu'à la suite de fausses manoeuvres, un certain nombre de granules n'ont pu être récupérés sur ce support. Pour récupérer des granules perdus sur la FU-Support, utiliser FUP (FUP2 : FUST ; FDES, FUP8 ; FUVA, FUP4 : FUCL, FUP2 : FUST, FDES).

`6022 **Table des fichiers saturée**
— Le nombre maximum de fichiers Permanents que la FU-Support peut contenir est atteint. La requête est ineffective.
— Le nombre maximum de fichiers Permanents Nfic que l'on peut créer, stocker sur un support est fonction des paramètres qui ont été donnés à l'initialisation de la FU-Support par FUP4 : FUINI ; FU, tg, nbg [, nfic]
— Lorsque nfic est fourni : Nfic $\leqslant$ inf (nbg — 1, nfic)
— Lorsque nfic est minimal : Nfic $\leqslant$ inf (nbg — 1, (tg — 2)16)
— **Solution** : Sauvegarder les fichiers importants et réinitialiser la FU-Support avec de nouveaux paramètres.
Maximum : nfic = nbg — 1. Puis restituer sur la FU-Support les fichiers Permanents sauvegardés.

`6028 **Erreur de syntaxe**
— Un des paramètres au moins du FCB est incorrect voir interface de programmation correspondant.
— La requête est ineffective.
— Cette erreur peut arriver lorsque soit l'adresse du FCB soit le numéro de requête, est incorrect. Voir chapitre 3.1 : Une requête à FMS.
— Cette erreur peut arriver lorsqu'une tâche maitre utilise un buffer de travail translatable, ou lorsqu'il est hors partition pour une tache esclave (FAU privées).

`6029

Adresse de FCB invalide

— Voir chapitre 3.1.3 l'interface SVC par. b) le FCB : File Control Block.

— La requête est ineffective. Le FCB de l'appelant n'est pas modifié ; le compte rendu se trouve dans le registre A seulement.

`602A

SU ou FU non gérée par FMS (ou IOCS)

— L'utilisateur a spécifié dans son FCB le numéro d'une FU non géré par FMS et/ou par IOCS. La définition des FU gérées par FMS et IOCS est réalisée à la génération du système à l'aide de GFMS16 et GENIO.

— L'utilisateur a spécifié dans son FCB un numéro de SU qui est invalide, ou pour lequel l'affectation par commande SU - FU est invalide.

— FU-SUPPORT non initialisée par FUINI ou mal initialisée. Rappel l'utilisateur fournit à l'initialisation d'une FU SUPPORT FUINI,FU,tg,nbg, et à la génération du système d'exploitation GFMS16

% CONFMS FU=D3 NBGRAN=300 (exemple)

Règle : nbg ⩽ NBGRAN

Rappel : pour le disque système, le type d'organisation standard ou grand disque, créé par FUINI en fonction du paramètre OFU (implicitement OFU = 0), doit correspondre à ce qui a été défini à la génération pour cette FU % CONFMS [OFU =]. Pour un disque volume le paramètre OFU mis à jour dans la structure de volume par SDEF doit être identique à celui qui a été défini par FUINI pour cet espace.

— La requête est ineffective.

`602B

Méthode d'accès non gérée

— Cette version opérationnelle de FMS ne contient pas la méthode d'accès correspondant au numéro de requête spécifié.

— **Exemple :** SVC (FMSX) ; alors que cette version ne contient pas le Séquentiel Indexé.

— **Exemple :** Pour toute requête erreur sur la valeur du numéro de requête :

— La requête est ineffective.

Erreurs graves

`6032
`6033
`6034

Informations Systèmes Invalides

— L'un de ces constats d'erreur signifie que soit le Système d'Exploitation en mémoire centrale fonctionne anormalement, soit que des informations système de FMS sur la FU-Support correspondante sont invalides, soit les deux.

— Il faut recharger un système (bootstrap), et vérifier le contenu des supports en ligne à cet instant là, à l'aide de FUP (FUP2 : FUST ; FDES, FUP8 : FUVA, FUP4 : FUCL).

— Le Traitement en cours sur le fichier correspondant s'est termine anormalement. Et ce fichier est dans un état incorrect.

— `6032 : cette erreur est détectée par le noyau de FMS lorsqu'il contrôle le chaînage des granules d'un fichier.

— `6033 : cette erreur est détectée par le noyau de FMS lorsqu'il y a tentative de désallocation du granule ZERO du support correspondant.
— `6034 : cette erreur est détectée lors d'un auto-contrôle des informations système en mémoire centrale de FMS.
Cette erreur peut également intervenir lorsque la génération du système est incorrecte, en particulier lorsque la FU spécifiée n'est pas connue de IOCS. Ou alors parce que le compte d'octets rendu par IOCS est différent de ce que lui a demandé FMS.

`6035 **FU Verrouillée par IOCS**
— Cette erreur est détectée par IOCS à l'aide des informations de la TBMCOM, ou par le superviseur BACKM lorsque une FU n'a pas été affectée au Background.
— Si possible revérifier la génération du système.
— Vérifier qu'aucune requête de verrouillage (Fonction de positionnement : ROU) n'a été lancée sur cette FU disque.
— En Background utiliser la commande GIVE.
— FMS fournit également ce compte rendu lorsque la longueur de l'échange qui lui est demandée est supérieure à la limite d'IOCS sur le support,et que la taille du granule sur ce support est aussi supérieure à la limite d'IOCS. Voir notice FUP (FUINI).

``4 . . . **Erreur hardware :** `4000 + mot d'état PU
— Cette erreur est détectée par IOCS lors d'un des échanges que lui demande FMS
— Le traitement de la requête est interrompu par FMS.
— En fonction du mot d'état PU corriger l'éventuelle panne hardware, puis agir comme pour les compte-rendus : Informations Système Invalides.

Remarques générales

— Lorsque FMS fournit un compte-rendu non inventorié pour la requête correspondante, il faut considérer cela comme une **erreur grave** et agir comme pour le compte-rendu : Informations Système Invalides.
— Il faut noter également, que l'interface ici décrit, est celui de l'appel au superviseur SVC. Cet interface peut donc être éventuellement modifié par le traitement d'un superviseur spécifique. Dans ce cas, il faudra se reporter au manuel correspondant.

3.2 - CARACTERISTIQUES EXTERNES DE FMS

3.2.1 - Le contexte multi-tâche

FMS est une partie d'un Système d'Exploitation gérant un contexte multi-tâche. Lorsqu'une tâche ou programme appelant, demande le fonctionnement d'un service de FMS il s'adresse en fait au superviseur du Système d'Exploitation à l'aide d'une requête SVC. C'est donc le " SVC Handler " qui active FMS.

- FMS fonctionne pour le compte d'une tâche software s'exécutant en mode maître ou en mode esclave.
- FMS s'exécute avec le contexte de la tâche appelante, mais en mode maître.

Remarques

- **Il est conseillé pour garantir la fiabilité du système de travailler en mode esclave chaque fois que cela est possible.**
- FMS convertit automatiquement les paramètres d'adresse, fournis par une tâche appelante en mode esclave. Le retour à la tâche appelante se fait normalement en fin d'échange, lorsque le traitement de la requête par FMS est considéré comme terminé.

Remarque

- L'utilisation de l'option de bufferisation des entrées/sorties séquentielles, réalise de par son fonctionnement, en lecture anticipée et écriture retardée, un traitement avec retour immédiat.

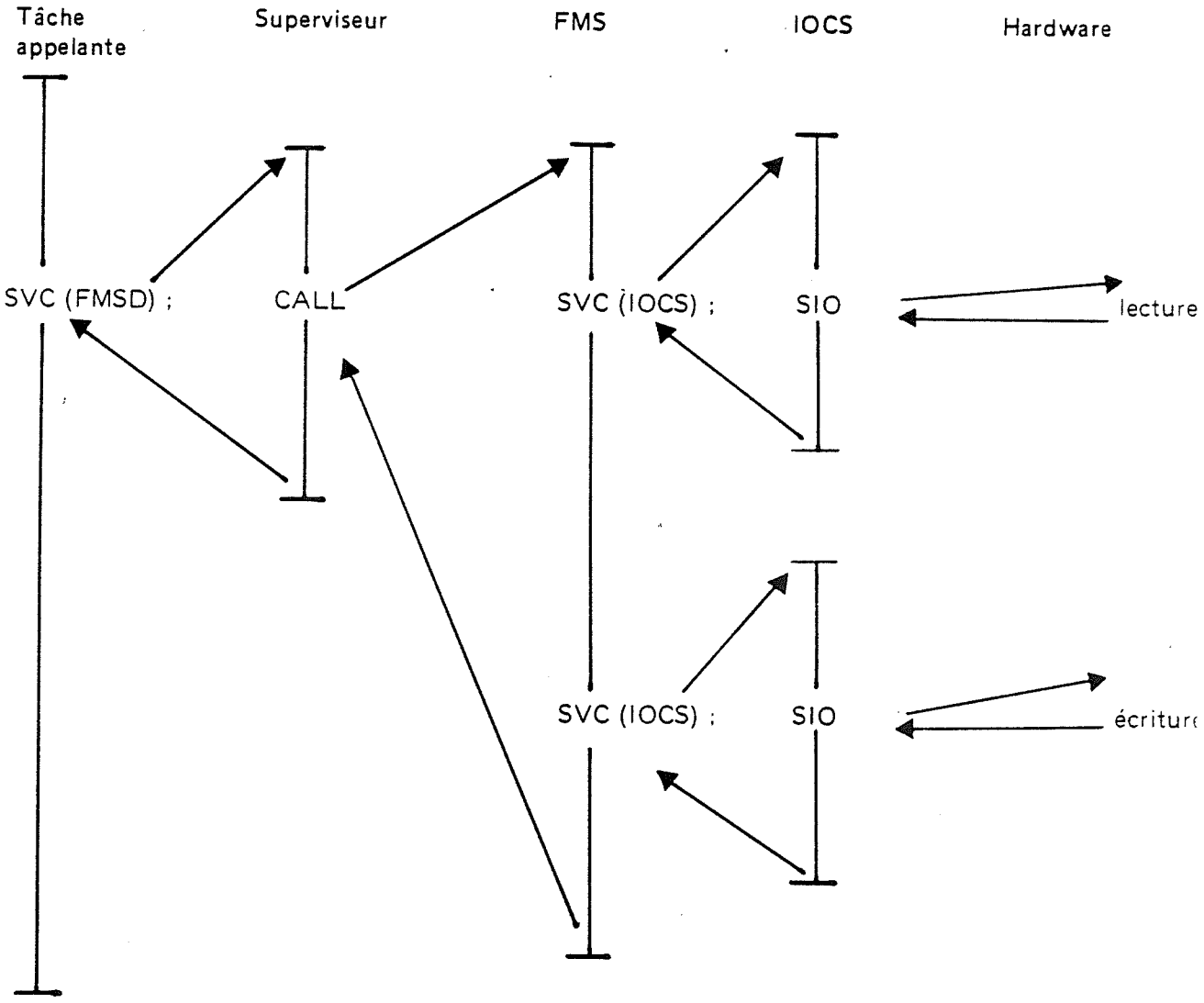
FMS est réentrant pour, de 1 à 128 tâches software ou selon la macro %USER du FMS déporté.

Remarques

- Certaines requêtes sont totalement réentrantes (avec cependant des zones critiques de faible durée, ex : algorithme de recherche d'une FAU).
  - requêtes FMSS : Séquentiel
    FMSD : Direct et Direct à Trous
    FMSX : Séquentiel Indexé
    FMSC : Séquentiel Chainé
    FMSI : Indexé
    FMSV : Direct Longueur Variable
- D'autres requêtes sont bloquantes. Une tâche est donc éventuellement mise en attente (Request Release) de fin du Traitement de la requête en cours pour une autre tâche.
  - requêtes FMS : OPen CLose

3.2.2 - Les entrées-sorties physiques

Le déroulement temporel des instructions de traitement d'une requête



Remarque :

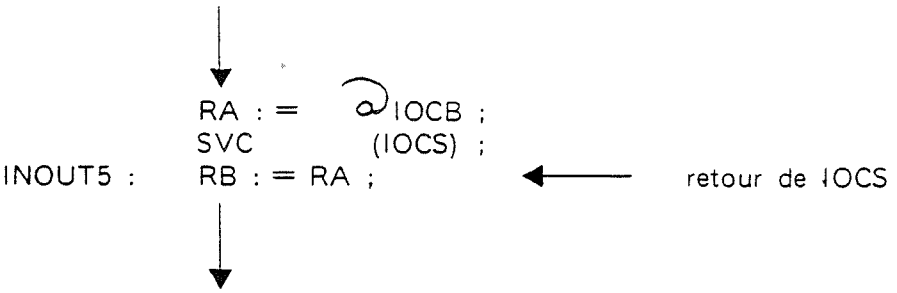
— FMS ne possède qu'un seul point d'entrée. Ce point d'entrée est connu de l'ingénieur système. Il est défini par la référence externe.

SVFMS

— FMS n'appelle IOCS qu'en un seul point du programme (FMS). Le point de retour d'IOCS est connu de l'ingénieur système. Il est défini à l'aide de la référence externe.

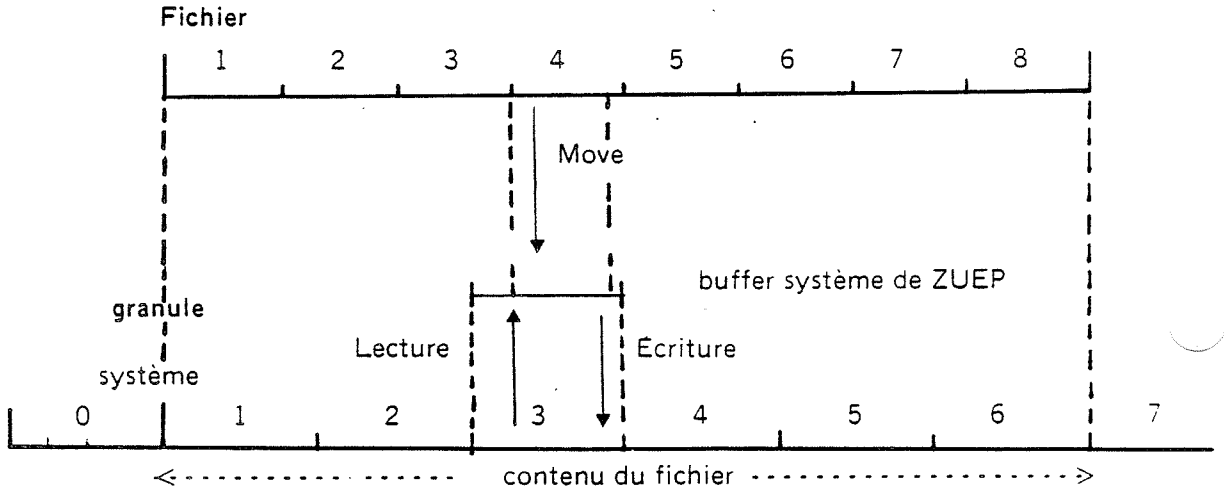
INOUT5

Listing FMS :



FMS gère automatiquement des entrées-sorties au niveau du mot, dans les fichiers stockés sur une mémoire secondaire où l'Unité d'Enregistrement Physique (UEP) est le secteur.

Implantation Physique d'un fichier Direct de 8 articles dans un granule de 1 K mot.



Dans un granule l'information de l'utilisateur est contigue physiquement au mot près.

FMS utilise un buffer système de la zone ZUEP pour résoudre les problèmes d'alignement au secteur et donc traiter des échanges d'article ou de portion d'article au niveau du mot.

Sur les **deux** schémas précédents on indique du point de vue des **entrées-sorties physiques**, comment se déroule une reécriture de l'article numéro 4 (DWRITE) (avec l'option ADR).

1°) une lecture du secteur disque concerné dans un buffer système de la zone ZUEP.
2°) un " move " de la zone d'échange dans ce buffer système.
3°) une écriture du buffer système sur le disque.

Remarque : Lorsque à l'instant t , la zone de pavés ZUEP est saturée, la tâche appelante pour laquelle travaille FMS est mise en attente. Cette tâche est relancée en fonction des règles générales de priorité et de la libération des pavés de la zone ZUEP, par les entrées-sorties **physiques** en cours et qui utilisent également un **buffer système** (UEP).

3.3 - LES OPTIONS DE PERFORMANCE DE FMS

3.3.1 - La bufferisation

a) **Présentation**

La bufferisation fonctionne pour l'accès séquentiel au niveau Portion d'Article, pour les fichiers de type Séquentiel, Indexé ou Direct.
La bufferisation a comme objectif de diminuer le temps d'accès à l'information en réduisant le nombre d'entrées-sorties physiques. Elle permet également de diminuer la charge des disques par rapport à celle de l'Unité Centrale.
La bufferisation nécessite un buffer de travail qui doit être fourni par l'utilisateur au moment de l'ouverture ou de la création du fichier. Le buffer est ensuite géré par le système jusqu'à la fermeture ou la destruction du fichier : le fonctionnement de la bufferisation est donc totalement transparent à l'utilisateur.

b) **Fonctionnement**

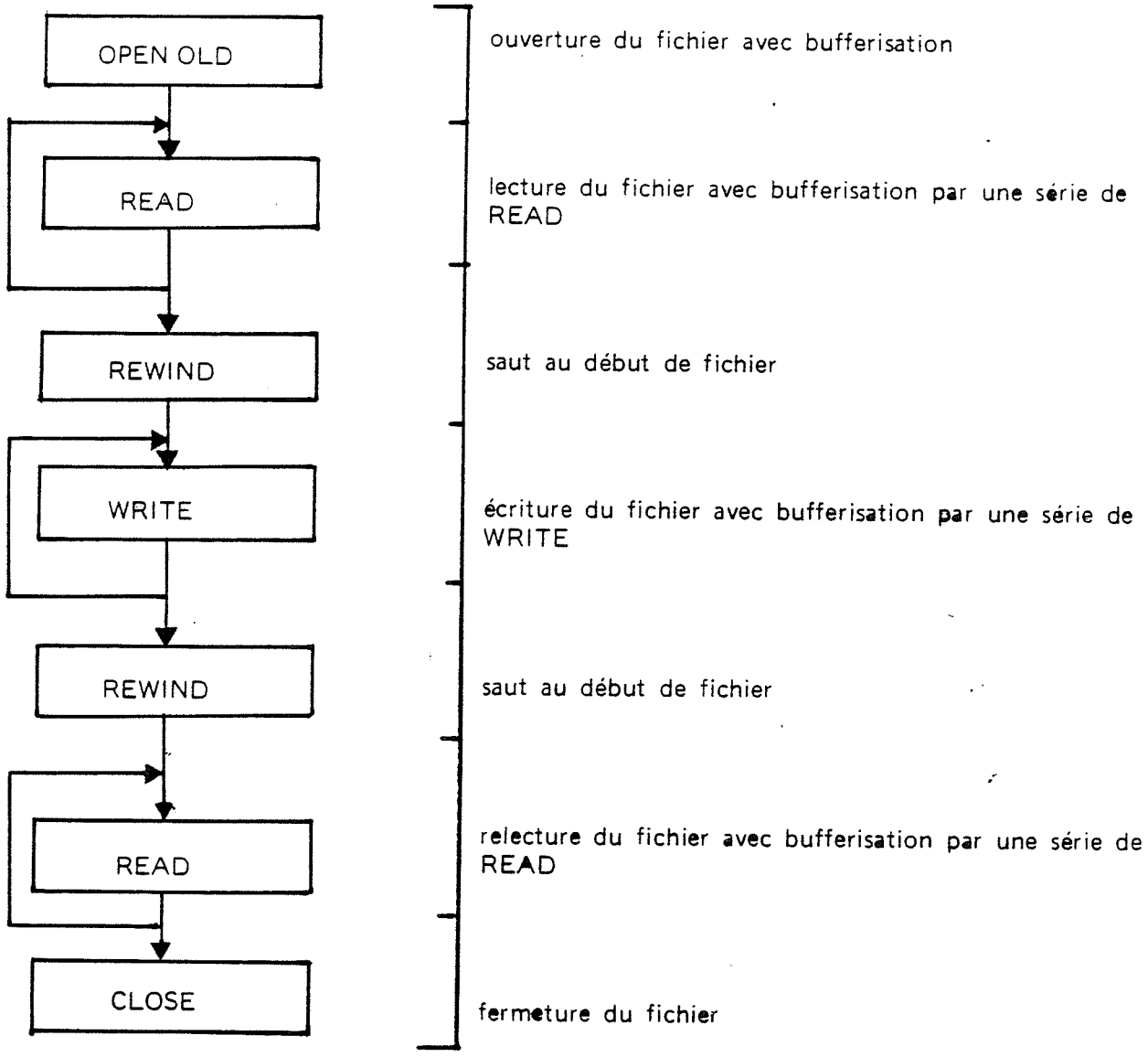
La bufferisation en écriture

La bufferisation en écriture consiste à mémoriser dans le buffer ce qui a été demandé d'écrire sur disque et l'écriture sur disque est effective mais sectorisée par le fait même du buffer.
Cette écriture peut être retardée, dans ce cas l'écriture ne sera effectivement faite sur le disque que lorsqu'on a besoin du buffer pour alimenter une autre zone du fichier en mémoire. L'écriture retardée est demandée à l'aide de la primitive PURGE (voir MU).
Attention ! Dans le cas de l'écriture retardée, l'écriture sur disque est alléatoire et effective qu'au CLOSE ou au PURGE. La primitive PURGE est décrite dans le chapitre Primitives systèmes du manuel d'utilisation.

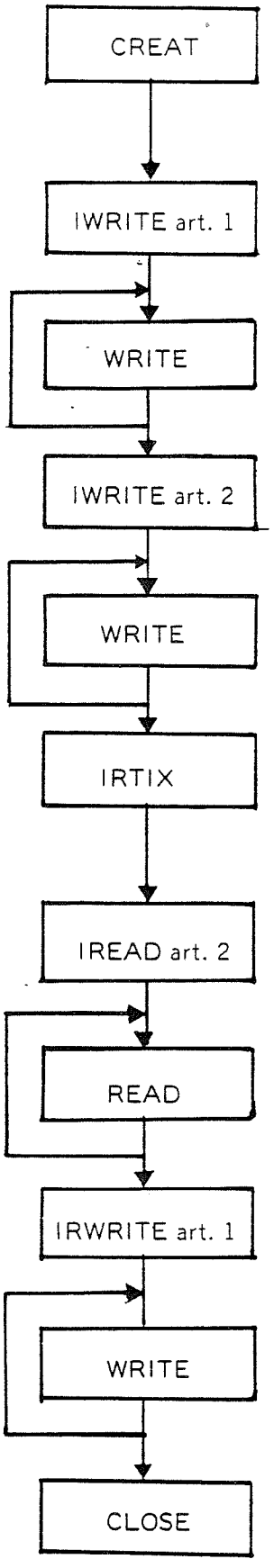
La bufferisation en lecture

La bufferisation en lecture consiste à faire de la lecture anticipée, permettant de minimiser les accès disque.

Exemple : Fichier Séquentiel



Exemple : Fichier Indexé



Création du fichier avec bufferisation

Écriture du premier article avec bufferisation

Écriture d'un second article avec bufferisation

Lecture de la table d'index sans bufferisation

Lecture du second article avec bufferisation

Reécriture du premier article sans bufferisation (car reécriture)

Fermeture du fichier

L'efficacité de la bufferisation

L'efficacité de la bufferisation peut dépendre de deux facteurs : la taille du buffer et le fait qu'il corresponde à une frontière de secteur.

1°) La taille du buffer

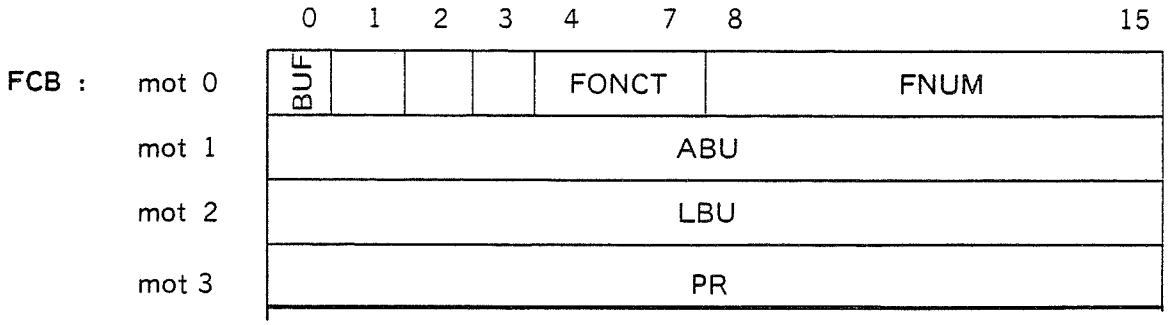
Le buffer fourni par l'utilisateur doit être grand par rapport aux lectures ou aux écritures qu'il demande. Evidemment, plus le buffer fourni est grand, plus la bufferisation est efficace.
Le buffer minimum est de 2 secteurs soit 256 mots plus 4 mots de gestion soit au total de 260 mots.

2°) Bufferisation cadrée sur une frontière de secteur

La bufferisation cadre automatiquement les échanges sur des frontières de secteurs (plus de découpage avec le ZUEP).

c) **Utilisation**

La bufferisation est attachée à une FAU créée sur un fichier : l'utilisateur demande donc la bufferisation à l'ouverture ou à la création d'un fichier (primitives OPEN-OLD, OPEN-NEW, CREAT). La bufferisation reste active jusqu'à la primitive de fermeture du fichier (CLOSE, EØJ).
Pour les autres requêtes l'interface de programmation reste inchangé.
Codage dans le FCB des paramètres nécessaires à la bufferisation pour les primitives : OPEN-NEW, OPEN-OLD, CREAT



BUF : = 1 si bufferisation demandée

= 0 sinon (ABU, LBU peuvent être quelconque).

ABU : adresse du buffer donné par l'utilisateur pour la bufferisation (16 bits)

LBU : longueur en octets de ce buffer.
$520 \leqslant LBU \leqslant 65534$
En fait, la longueur utilisée est multiple d'un secteur plus 8 octets soit :
$8 + 256 x$
Pour communiquer une taille de buffer plus grande voir la primitive PURGE décrite dans le MU.

Compte-rendu Du fait que la bufferisation effectue de l'écriture retardée ou de la lecture anticipée, les messages d'erreur risquent d'être décalés par rapport à la primitive demandée.
Donnons un exemple :
L'utilisateur, après avoir fait de la bufferisation en écriture, ferme son fichier : les messages d'erreur qu'il risque de recueillir peuvent aussi bien concerner l'écriture de ce qui reste dans le buffer que la fermeture du fichier.

3.3.2 - L'Accès Direct Rapide :

a) **Présentation**

L'option Accès Direct Rapide (ADR) a pour objectif de permettre l'accès aux articles des fichiers de type Direct et Direct à Trous en 1 **seul accès disque.**
Par extension, l'option ADR fonctionne pour tous les fichiers à **Organisation Physique Directe :**

- Indexé sur lequel on a passé l'utilitaire de FUP10 : FAST
- Direct et Direct à trous
- Séquentiel Indexé
- Séquentiel Chaîné

b) **Fonctionnement**

L'utilisateur n'a rien à fournir à FMS pour réaliser l'option ADR. Il se contente de le commander à l'aide d'un booléen lors de la création d'une FAU sur le fichier désiré.
FMS utilisera une zone système (des pavés de la ZDF) pour charger en mémoire centrale la Table des ligatures des granules (TLG) occupés par le fichier. Il faut donc prévoir à la génération du système la place nécessaire dans la ZDF.
Paramètres à prendre en compte :

— Le nombre de fichiers ouverts simultanément en mode ADR.
Remarque :
Lorsque plusieurs FAU ont été créées sur un même fichier avec l'option ADR, la TLG est unique. Toutes les FAU profitent du fait que l'une d'entre elles l'a demandé.

— L'occupation de la TLG en mémoire centrale pour chaque fichier ouvert en mode ADR :
Règle : pour un fichier :
- Taille de la TLG en MC pour l'Accès Direct Rapide :
  . Type d'organisation disque :
    STD . Un pavé pour 28 granules
    GDi . Petite FU : Un pavé pour 28 granules
    . Grande FU : Un pavé pour 14 granules.

Exemple :
- Pour un fichier occupant sur disque 2 granules, la TLG occupe 1 pavé de la ZDF.
- Pour un fichier occupant 28 granules, la TLG occupe :
  . Un pavé de la ZDF si ce fichier se trouve sur une FU à organisation standard, ou sur une petite FU à organisation GDi.
  . Deux pavés de la ZDF si ce fichier se trouve sur une grande FU à organisation GDi.
- Pour un fichier occupant 128 granules, la TLG occupe :
  . 5 pavés de la ZDF si ce fichier se trouve sur une FU à organisation standard ou sur une petite FU à organisation GDi.
  . 10 pavés de la ZDF si ce fichier se trouve sur une grande FU à organisation GDi.

Fichier Direct : 1 seul accès disque

Conditions à réaliser pour que la lecture ou l'écriture d'un article d'un fichier de type Direct, s'effectue en 1 accès disque :

A la **création du fichier Direct** (OPEN NEW, CREAT)

— **Taille du fichier : moins de 128 granules** (obligatoire)
Remarques :
- Taille maximum théorique d'un fichier à organisation physique directe :
  4.177 920 mots

— **Taille de l'article : multiple de secteur**
TART /2 = K x 128 mots

Remarque :
— **Si possible** également : la taille de l'article doit être un sous-multiple de la taille utile du granule. Soit : TART en octets, tg en secteurs :

$$K \cdot (TART/256) = tg - 1$$

A l'utilisation du fichier (OPEN NEW, OPEN OLD, CREAT)

— Commander le fonctionnement de l'option Accès Direct Rapide à l'aide du booleen : ADR = 1, et NLC = 1 (Pour DWRITE en 1 accès disque).

Fichier Indexé : IREAD 1 seul accès disque.
— Créer un fichier de moins de 128 granules où la taille des articles est multiple du secteur.
— Passer sur le fichier l'utilitaire de FUP10 : FAST.
— Demander l'option ADR à l'ouverture du fichier.
— Utiliser la primitive IRTIX.
Remarque : L'utilisation de la primitive IWRITE détruit automatiquement (transparent à l'utilisateur) l'organisation physique directe créé par FAST.

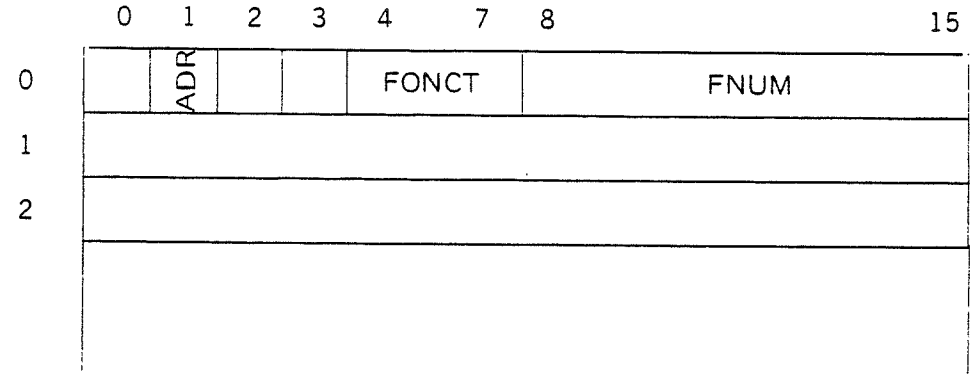
c) **Utilisation**

Le fonctionnement de l'option ADR est attaché à une FAU créée sur un fichier. L'utilisateur le demande donc à l'ouverture ou à la création d'un fichier (primitives OPEN OLD, OPEN NEW, CREAT). Le fonctionnement de l'option ADR se termine à la fermeture du fichier (CLOSE, DELET, EOJ).

Pour les autres requêtes l'interface de programmation reste inchangé.

Codage dans le FCB des primitives : OPEN NEW, OPEN OLD, CREAT.

FCB



ADR = 1 Accès Direct Rapide demandé
= 0 sinon

3.3.3 - La lecture de contrôle

a) **Fonctionnement**

L'utilisateur a la possibilité au niveau de chaque FAU de commander comment IOCS doit réaliser toutes les écritures associées à une FAU.

Écriture avec lecture de contrôle

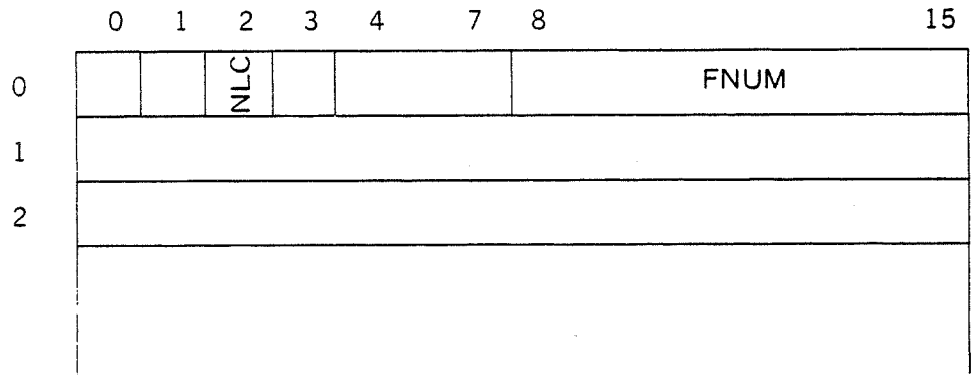
1°) Écriture de l'information
2°) Lecture de contrôle
— Lecture fictive avec calcul du checksum
— Comparaison avec le checksum lu.
3°) Retour automatique (x fois) au 1°) si erreur.

Écriture simple de l'information

b) **Utilisation**

Codage dans le FCB des primitives : OPEN NEW, OPEN OLD, CREAT

FCB



NLC = 1 Écriture simple
= 0 Écriture avec lecture de contrôle

3.4 - UNE REQUETE A FMS : INTERFACE 1024 K

L'interface de programmation des requêtes à FMS permet de réaliser :

— Une programmation en **mode maitre** de 0 à 64 K avec l'**interface** 64 K Standard
— Une programmation en **mode esclave** de 0 à 1024 K avec l'**interface** 64 K Standard
— Une programmation en **mode maitre panaché** avec un nouvel interface pour exploiter des zones d'échange ou des buffers de travail de 0 à 1024 K
Cet interface est utilisé dans la programmation des systèmes.
— Une programmation en **mode maitre ou esclave** utilisant un nouvel interface d'accès à la **CDA** pour demander à FMS d'y gérer des zones d'échange ou des buffers de travail

L'interface de programmation de 0 à 1024 K est donc **compatible** de façon **ascendante** avec celui de 0 à 64 K, FMS fournit donc 3 formes de FCB :

— l'interface 64 K
— l'interface d'accès à la CDA
— l'interface mode maitre panaché

3.4.1 - L'interface 64 K

La description générale de cet interface est faite au paragraphe 3.1. La description détaillée de chaque requête est faite dans le chapitre OPEN CLOSE (nº 4) et les chapitres correspondant au différentes méthodes d'accès. Avec une mémoire centrale de 0 à 1024 K cet interface permet les types de programmation suivants :

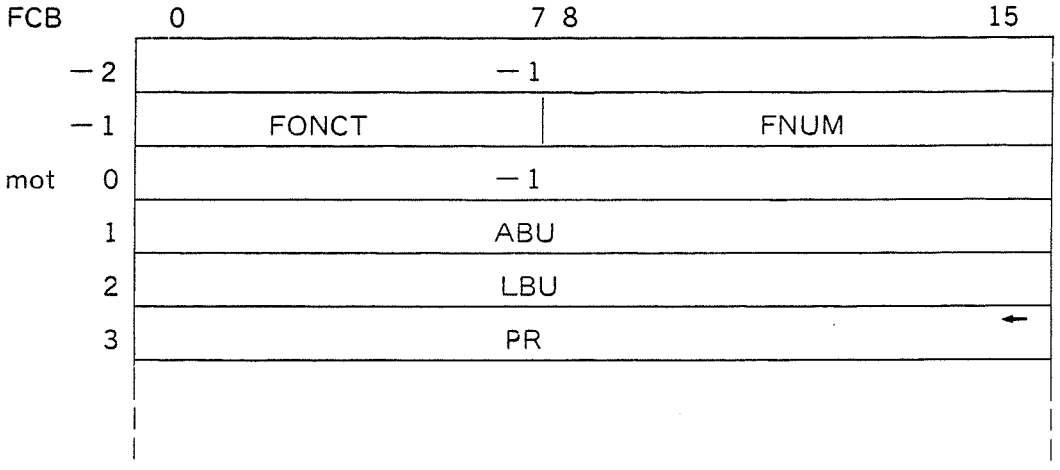
a) **En mode esclave pur** dans une partition implantée entre 0 et 1024 K et contenant les 5 zones (adresses relatives à SLO)

b) **En mode maitre pur** les 5 zones implantées entre 0 et 64 K (adresses absolues).

3.4.2 - L'interface d'accès à la CDA (FCB étendu)

Les usagers de FMS peuvent utiliser la CDA pour y implanter des buffers de travail ou des zones d'échange. Cela permet de libérer de la place dans les partitions esclaves et dans la zone de 0 à 64 K. Cela permet également une communication inter-tâche efficace : n taches peuvent utiliser les résultats d'un échange réalisé par l'une d'entre-elles.

Appel : RA := @ FCB ; SVC FMS x ;



@FCB	:	est l'adresse du mot 0 (adresse 16 bits)
FMSx	:	est le numéro de SVC standard de l'interface 64 K
mot — 2	:	— 1 signifie échange avec la CDA
mot — 1	:	correspond au mot 0 de l'interface 64 K
mot 0	:	— 1 signifie FCB étendu (par le haut)
ABU	:	adresse relative au début de la CDA dont l'adresse est stockée dans la mémoire débanalisée `18

Le reste du FCB est inchangé par rapport à l'interface 64 K

Cet interface utilisable pour toutes les requêtes à FMS permet les types de programmation suivante :

a) **En mode esclave** dans une partition implantée de 0 à 1024 K avec :

- **dans la CDA** (adresses relatives à la CDA) : la zone d'échange ou le buffer de travail
- **dans la partition** (adresse relative à SLO) : le code le FCB, la Kstore, la table des codes d'arrêt.

b) **En mode maitre** avec

- **dans la CDA** (adresses relatives à la CDA : la zone d'échange ou le buffer de travail
- **entre 0 et 64 K** (adresses absolues, indirection impossible) le FCB, la Kstore, la table des codes d'arrêt.

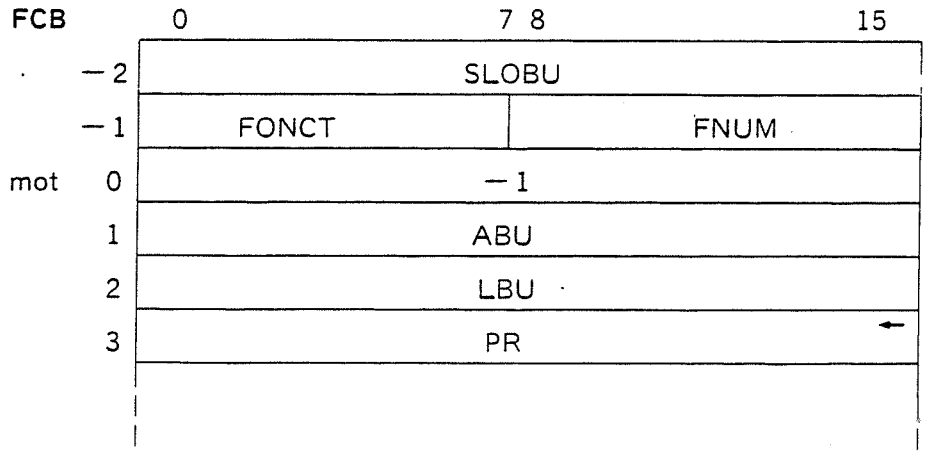
Remarque :

La Kstore étant gérée par la machine son adresse est relative au SLO du contexte en cours. Par exemple, si le 1er appelant est une tache esclave (enchainement de SVC) la Kstore doit être dans la partition esclave (adresse relative à SLO).

3.4.3 - L'interface mode maitre panaché (FCB étendu)

Cet interface est la forme générale permettant à une tache en mode maitre de réaliser des échanges pour le compte d'une partition esclave implantée entre 0 et 1024 K.

Appel : RA : =@FCB ; SVC FMSx



FCB : est l'adresse du mot 0 (adresse absolue 16 bits)
FMSx : est le numéro de SVC standard de l'interface 64 K
mot — 2 : est l'adresse de début de partition esclave
mot — 1 : correspond au mot 0 de l'interface 64 K
mot 0 : — 1 signifie FCB étendu (par le haut)
ABU : adresse relative à SLO

Le reste du FCB est inchangé par rapport à l'interface 64 K

Ce mode d'appel est utilisé par les superviseurs en particulier pour le chargement de taches esclaves et pour l'exécution des requêtes BCHLR.

— **dans la partition esclave** (adresse relatives à SLOBU) la zone d'échange ou le buffer de travail
— **entre 0 et 64 K** (adresse absolues, indirection impossible) le FCB, la table des codes d'arrêt.

3.5 - INTERFACE FAU PUBLIQUES

Remarques :
Voir définition au chapitre 2.5

Les informations qui suivent décrivent le fonctionnement des requêtes utilisant les FAU publiques et le mécanisme de synchronisation. L'interface de programmation des FAU Privées reste inchangé.

3.5.1 - Spécifications générales

L'interface de programmation des requêtes des niveaux Article et Portion d'Article n'est modifié qu'en un seul point : le compte-rendu 601F : FAU en cours, ne sera plus rendu par FMS, qui synchronise automatiquement toutes les requêtes à l'aide du sémaphore associé à la FAU.

L'interface de programmation du niveau OPEN CLOSE est modifié comme suit :

— le compte-rendu 601F: FAU en cours, ne sera plus rendu par FMS
— les requêtes DELET ALTER et RENAM ne seront acceptées que si la FAU publique est utilisée par un seul usager.
Si les requêtes sont refusées pour cette raison, FMS fournit le compte-rendu 601E : Fichier occupé.
— la requête RENUM est interdite avec les FAU publique. Si la valeur des paramètres FNUM ou NFNUM appartient à la plage des FNUM Publics, FMS fournit le compte-rendu 6018 incompatibilité primitive fichier.
— le paragraphe 5.3.5 explique le fonctionnement de la nouvelle requête ATADET : Attachement et Détachement d'une FAU publique
— les requêtes OPEN OLD et CLOSE sont modifiées comme suit :
Le 1er OPEN OLD sur une FAU publique c'est-à-dire celui qui crée la FAU fonctionne comme un OPEN OLD normal (sauf 601E).
C'est lui qui si nécessaire, fourni le buffer de travail dont aura besoin la FAU publique.
Le buffer de travail devra être exploitable par FMS pour chaque requête pendant toute la durée de vie de la FAU publique c'est-à-dire jusqu'au dernier Close.
— une option de la requête EOJ (USR) permet de détruire les FAU identifiées par un numéro d'usager public.

Le dernier CLOSE sur une FAU publique c'est-à-dire celui qui détruit la FAU, fonctionne comme un CLOSE normal (sauf 601E).

L'interface de programmation d'un n^{ième} OPEN OLD ou CLOSE est expliqué aux paragraphes suivants, ainsi que pour EOJ (USRP).

3.5.2 - Le service d'allocation de buffers

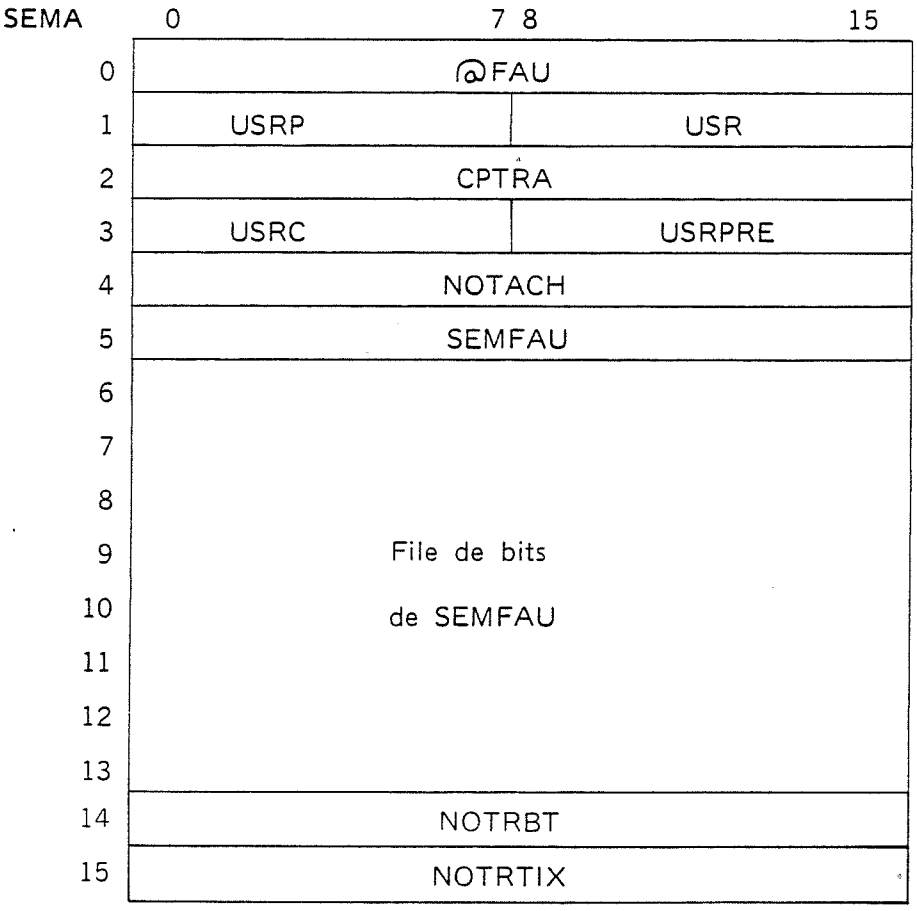
Lorsqu'une FAU Publique nécessite pour son fonctionnement un buffer de travail (OPEN NEW/OLD, CREAT, IRTIX), celui-ci constitue une ressource partagée par un ensemble de tâches. Ce buffer doit être disponible à tout moment pour les tâches qui utilisent la FAU, donc résident et implanté à adresse fixe en mémoire.

Cette règle n'est pas respectée lorsque le superviseur déplace les tâches esclaves à l'aide du DRPS. Dans ce cas et pour les FAU publiques, FMS demande un buffer au superviseur, il n'exploite donc pas celui fourni par l'utilisateur.

L'allocateur du superviseur gère 3 zones de buffers n° 8, 9, 10 pour lesquelles le nombre et la taille des buffers sont définis par l'utilisateur à la génération du système. FMS choisit l'une des 3 zones en fonction de la longueur spécifiée dans le FCB (LBU) : la plus petite taille supérieure ou égale à la longueur demandée.

Afin de permettre à une tâche esclave d'accéder au buffer de travail d'une FAU Publique lorsque celui-ci est fourni par le système, FMS met à sa disposition une requête du niveau OPEN CLOSE du type move : FMS MOVE dont l'interface de programmation est spécifié au paragraphe 3.5.7.

3.5.3 - Sémaphore de FAU publique



Description des paramètres

@FAU : adresse de la FAU publique associée
USRP : numéro d'usager public
USR : numéro d'usager privée du créateur de la FAU
CPTRA : compteur d'OPEN OLD exécutés sur la FAU
USRC : numéro d'usager courant
USRPRE : numéro d'usager précédent
NOTACH : numéro de la tâche qui bloque la FAU (ATADET). Lorsque la FAU est libre NOTACH = 128.
NOTRBT : numéro de la zone de pavés dans laquelle a été alloué le buffer de travail
NOTRTIX : numéro de la zone de pavés dans laquelle a été alloué le buffer pour IRTIX.

Remarque :

Au début de l'exécution de toute requête sur la FAU publique FMS réalise :
USRPRE : = USRC
USRC : = USR de la requête en cours

3.5.4 - N^ième OPEN OLD

Nom	**N^ième OPEN OLD / FAU Publique**
But	Accès à une FAU publique

Appel RA : =@ FCB SVC (FMS) ; où FMS = `38 .

FCB

	0					8	15
0	∀		2			FNUM	
1	∀						
2	∀						
3	PR						
4	FNAM						
5							
6							
7	PUBW						
8	∀	S	W	∀	∀	SU/FU	
9	∀						
10	∀						
11	∀						
12	∀						

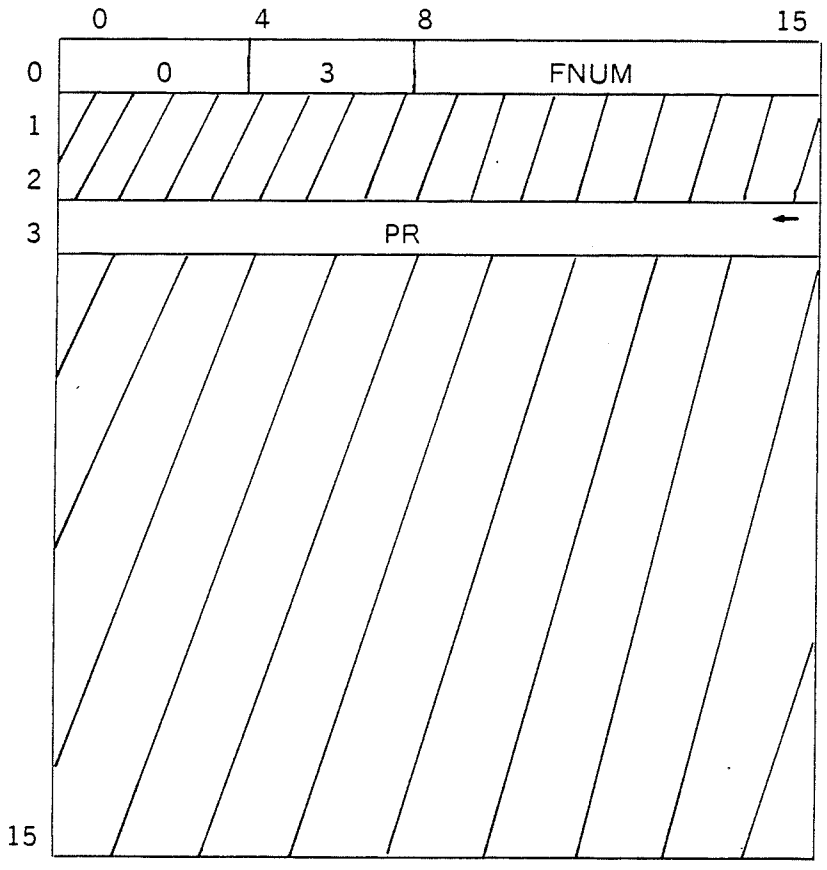
Compte rendu	Valeur de PR	Signification
	0	**Primitive correctement exécutée**
	`600B	**FAU inexistante :** la FAU publique ne travaille pas sur le fichier désigné
	`6014	**Protection écriture :** la FAU Publique est en lecture seulement la requête est ineffective
	`6015	**Permanent de nature différente**
	`601A	**Erreur d'enchainement :** le nombre d'accès demandé sur cette FAU publique dépasse 32 767 la requête est ineffective
Erreurs filtrées	`6028	**Erreur de syntaxe :** (@ FCB, FONCT, PR, FNAM, PUBW, SU/FU, S, W)
	`6029	**Adresse de FCB invalide**
	`602A	**SU ou FU non gérée par FMS**

3.5.5 - N^ième CLOSE

Nom	**N^ième CLOSE / FAU Publique**
But	Libérer 1 accès à une FAU publique

Appel RA : = @ FCB SVC (FMS) ; ou FMS = `38

FCB



Compte rendu

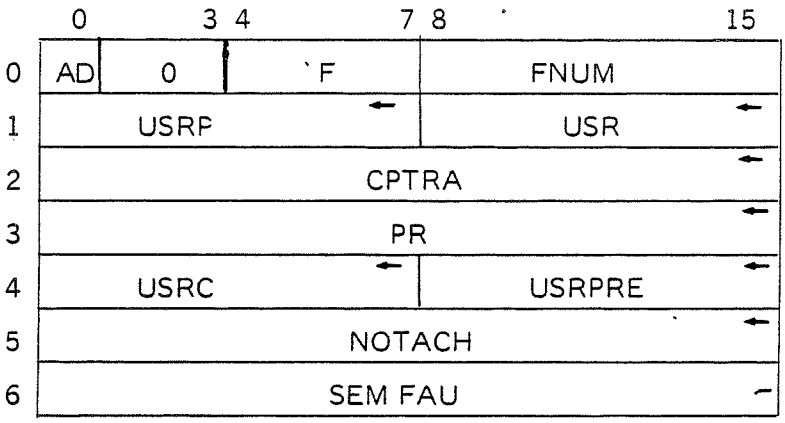
	Valeur de PR	Signification
	`600A	FAU inexistante
	`600B	FAU existante : primitive correctement exécutée
	`601F	FAU en cours : cas ou sy = 0
Erreurs filtrées	`6028	Erreur de syntaxe : @FCB, FONCT, PR
	`6029	Adresse de FCB invalide

3.5.6 - ATADET : Attach Detach

Nom	ATADET
But	Attachement ou Détachement d'une FAU publique

Appel RA : = @FCB SVC (FMS) ; << FMS = `38

FCB



Description des paramètres

AD : AD = 0 ⇒ Attach de la FAU
AD = 1 ⇒ Detach de la FAU

FMS fourni dans le FCB les informations associées au sémaphore de la FAU publique. Leur signification est décrite au paragraphe 3.5.2

Compte rendu

	Valeur de PR	Signification
	0	Primitive correctement exécutée
	`600A	FAU inexistante
	`601A	Erreur d'enchaînement : - 2 ATTACH de suite sur la même FU - si une tâche exécute la requête DETACH sans avoir exécuté ATTACH
Erreurs filtrées	`6028	Erreur de syntaxe : (@FCB, FONCT)
	`6029	Adresse de FCB invalide

3.5.7 - FMS MOVE : accès au buffer de travail

Nom	**FMS MOVE**
But	Transfert entre une zone d'échange et le buffer de travail associé à une FAU

Appel SVC (FMS) << FMS = `38

FCB

	0	1	4	7	8	15
0	BT	M	O	`E	FNUM	
1	ABU					
2	LBU					
3	PR					
4	ABT					

Description des paramètres

BT BT = 1 buffer de travail ; BT = 0 table d'index (IRTIX)

M Indicateur spécifiant le sens du MOVE

M = 0 BT → ZE
M = 1 ZE → BT

ABU LBU spécifications de la zone d'échange

ABT Adresse relative dans le buffer de travail

$0 \leqslant ABT < long\ BT$

Valeurs du PR	Signification
0	Requête correctement exécutée
`600A	FAU inexistante
`6014	Protection écriture : si M = 1 et FAU en lecture
`6018	Incompatibilité primitive fichier : il n'y a pas de buffer de travail
`601F	Primitive en cours
`6028	Erreur de syntaxe (ABU, LBU, ABT)
`6029	Adresse de FCB invalide

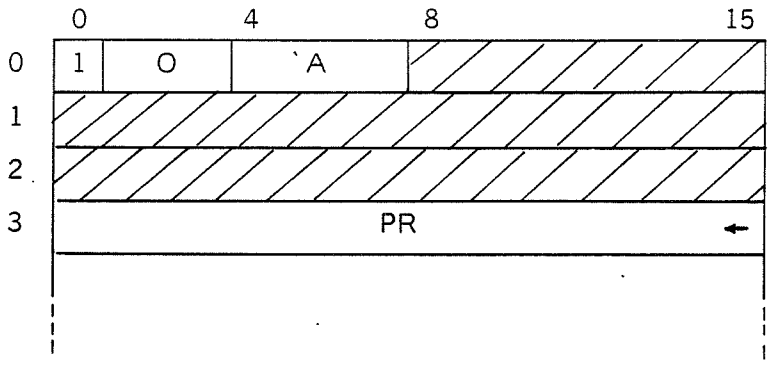
3.5.8 - EOJ (USRP) : EOJ usager public

Nom EOJ (USRP)

But Détruire les FAU identifiées par un numéro d'usager public. Détruire les Temporaires. Fermer les permanents.
Les FAU publiques sont détruites quel que soit la valeur du compteur d'OPEN (CPTRA) dans le pavé du sémaphore : SEMA.

Appel RA : = @ FCB ; SVC (FMS) ; << FMS = `38

FCB



Compte rendu Idem EOJ (USR) (4.2.10).

4 — L'ENTITÉ FICHIER : OPEN, CLOSE

4 — L'ENTITÉ FICHIER : OPEN, CLOSE

4.1 - UTILISATION DES TEMPORAIRES ET PERMANENTS

Le lecteur trouvera principalement dans ce paragraphe des exemples décrivant les différents enchaînements de requêtes, selon que les fichiers sont temporaires ou permanents.

4.1.1 - La création d'un fichier

FMS fournit deux requêtes de création de fichiers :
— OPEN NEW : pour créer un fichier Temporaire
— CREAT : pour créer un fichier Permanent.
Elles fonctionnent globalement de façon semblable et utilisent un ensemble de parametres fournis par l'utilisateur et spécifiant les attributs et caractéristiques propres au fichier.

L'Identification du fichier

— **FNAM :** nom du fichier (6 caractères)
— **PUBW** (seulement pour un Permanent) : mot de passe public, ou nom de catalogue (2 caractères). (Voir chapitre 2.1.1 § c) le Catalogue).
— **SU/FU :** désignation du support sur lequel FMS devra créer le fichier (1 octet).
FU : le support est désigné par une Unité Fonctionnelle. Espace d'adressage disque défini à la génération du système. (Voir chapitre 2.1.1 § d) : Unité Fonctionnelle et Unité Physique).
SU : Le support est désigné par une Unité Symbolique. L'affectation est indépendante de la configuration physique des Unités Fonctionnelles. (Voir chapitre 2.4.2 § c) : Utilisation des Unités Symboliques).

Caractéristiques du fichier

— **EMA/SID :** le numéro (4 bits) définissant quelle méthode d'accès devra créer le fichier et plus exactement quelle sera l'organisation logique du Fichier (Séquentiel, Indexé, ...).
— **S et W** (seulement pour un Permanent, 2 bits) :
S : indique quelle sera la nature du Permanent.
S = 0 : NON Simultané en principe réservé aux Tâches Temps Réel (Foreground).
S = 1 : Simultané en principe réservé au Background BOS16
(Voir chapitre 2.4.4 § c) et d)).
W : indique si le fichier sera accessible en écriture ou non. Remarque, cette caractéristique ne prend effet qu'à partir du CLOSE de la phase de création du fichier, et ne peut être modifiée que par la requête ALTER.
W = 1 ⟺ Fichier accessible en écriture.
— **De 0 à 4 paramètres** spécifiques de chaque méthode d'accès et qui fournissent des informations définissant en taille et nombre la structure en article du fichier. Ces informations permettront à FMS d'allouer la place initiale ou totale nécessaire au fichier, et à la méthode d'accès d'initialiser éventuellement des informations système lui permettant de gérer l'organisation logique du fichier. (Voir pour chaque méthode d'accès, le paragraphe : Fonctions Logiques : Le niveau Fichier).

Utilisation du Fichier en création

Lorsqu'un utilisateur crée un fichier, FMS crée également une **Unité d'Accès** à ce fichier (FAU). Ceci implique que le créateur du fichier peut, lorsque ce dernier est Temporaire, l'utiliser (écrire, lire), et lorsqu'il est Permanent, réaliser une phase de création du contenu initial du fichier.

On remarquera que pour un fichier **Permanent** la phase de création délimitée par [CREAT CLOSE] est **privilégiée** pour le créateur du fichier.
— Il peut écrire sur le fichier quelle que soit la valeur qu'il ait donné au bit de protection écriture W.
— Il sera seul à utiliser le fichier au sens où ce dernier sera :
  — inaccessible aux autres usagers (pas de partage simultané).
  — inaccessible par un autre chemin d'accès du même usager. Donc en Mono-Accès.
L'usager fournira donc des paramètres liés à l'utilisation du fichier.

Identification de la FAU

— **FNUM :** Numéro de l'Unité d'Accès (FAU) au fichier (1 octet). Ce numéro sera référencié par toute requête accédant au fichier. (Voir chapitre 2.4.3 § d) : La notion de chemin d'accès).

Paramètres de fonctionnement spécifiques de la FAU

— **ABU, LBU, BUF :** Ces trois paramètres sont spécifiques de la méthode d'accès utilisée, ils permettent à un usager de fournir un buffer de Travail à FMS
  ABU : Adresse du buffer (16 bits d'adresse, pas de bit index)
  LBU : Longueur du buffer en octets
  BUF : Booleen confirmant le fait que l'utilisateur a fourni un Buffer de Travail et donc que les informations ABU, LBU sont à prendre en compte.
  Remarque : Ces 3 paramètres sont parfois inutiles, parfois optionnels, parfois obligatoires selon la méthode d'accès. (Voir le chapitre décrivant la méthode d'accès correspondante).
— **ADR, NLC :** Des booleens paramétrant le fonctionnement des entrées-sorties exécutées par cette FAU. (Voir chapitre 3.3 : Les options de performance).

4.1.2 - Le Fichier Temporaire

a) **Création, utilisation et destruction**

Exemple : Soit un programme sous BOS16 s'exécutant avec le numéro d'usager BG (Background).

1°) Ce programme crée un fichier Temporaire Séquentiel de nom TEMPOR sur le support adressé par la FU D3, et à l'aide de l'unité d'accès numéro 1.

```
OPEN NEW   1, TEMPOR, S, D3
```

Remarque : A partir de maintenant ce programme ne peut plus créer de FAU de numéro 1 ni de fichier Temporaire de nom TEMPOR sur la FU-Support D3, ce qui est possible pour un autre usager en particulier pour une tâche Foreground (numéro d'usager différent).

2°) Utilisation : écriture en séquentiel puis relecture du fichier à l'aide d'une zone d'échange définie par ABU, LBU (adresse Longueur).

```
WRITE      1, ABU, LBU  ↺  n fois

REWIND     1

READ       1, ABU, LBU  ↺  jusqu'à fin de fichier :
                           PR = '6001
```

Remarque : L'allocation de la place nécessaire à un fichier Séquentiel est totalement dynamique. Elle est réalisée pendant la phase d'écriture du fichier.

3°) Destruction de la FAU et du fichier

```
CLOSE   1        ou        DELET   1
```

Remarque : La place disque occupée par le fichier sur la FU-Support D3 est alors automatiquement récupérée pour les autres fichiers qui seront créés ou agrandis sur la FU-Support D3, comme pour toute destruction de fichier. A partir de maintenant l'usager peut de nouveau créer une unité d'accès de numéro 1 et/ou un fichier Temporaire de nom TEMPOR sur la FU-Support D3.

b) **Transformation en Permanent**

Exemple : Soit un programme sous BOS16 s'exécutant avec le numéro d'usager BG (Background).

1°) Ce programme crée un fichier Temporaire de nom TEMPOR sur la FU-Support D3 avec l'unité d'accès de numéro 12. Le fichier est Direct (100 articles de 40 mots). Il est rempli en séquentiel à partir d'une zone d'échange de 256 octets (1 secteur).

```
OPEN NEW 12, TEMPOR, D (100, 80), D3
```

WRITE 12, ABU, LBU = 256  jusqu'à fin de fichier
PR = '6001

2°) Le fichier Temporaire est transformé en Permanent **NON Simultané, protégé en écriture,** à fin que les tâches du Foreground puissent l'utiliser. Le nom du nouveau fichier Permanent est DIRECT-C1. Il est bien sûr toujours stocké sur la FU-Support D3.

```
CATAL 12, DIRECT-C1, S = 0, W = 0
```

Remarque : A partir de maintenant le fichier est Permanent. Il est toujours accessible au programme du Background par la FAU n° 12, même en écriture. Le fichier se trouve donc dans une phase identique à la phase de création de tout fichier Permanent : Mono-accès et inaccessible aux autres usagers en particulier à ceux du Foreground.
Le programme créateur peut donc par exemple modifier le 1er mot du fichier.

```
REWIND 12
WRITE  12, ABU, LBU = 2
```

3°) Destruction de la FAU n° 12 et fermeture du fichier Permanent

```
CLOSE  12
```

Remarque : A partir de maintenant le fichier Permanent NON Simultané est accessible à tout usager par une requête d'ouverture en lecture seulement.

```
OPEN OLD   1, DIRECT-C1, D3, S = 0, W = 0
```

Il est accessible en lecture par exemple au niveau article : lecture de l'article n° 100.

```
DREAD      1, 100, ABU, LBU = 80
```

Destruction de la FAU n° 1 et fermeture du fichier.

```
CLOSE      1
```

4.1.3 - Le Fichier Permanent

a) **Création, initialisation, fermeture**

Exemple : Soit un programme sous BOS16 s'exécutant avec le numéro d'usager BG (Background).

1°) Ce programme crée un fichier Permanent Simultané, accessible **en lecture et en écriture.** Il sera Indexé 63 articles au maximum, et créé sur la FU-Support D2 avec le nom FMSSTD- : S

```
CREAT  4, FMSSTD- : S, I (63), D2, S = 1, W = 1
```

2°) Les articles de ce fichier sont ensuite créés, création de l'article de nom IDPSTD.

```
IWRITE 4, ABU, LBU, IDPSTD
etc...
```

Remarque : Pendant la phase de création du fichier (création initialisation fermeture) le fichier est en mono-accès et inaccessible aux autres usagers.

3°) Destruction de la FAU n° 4 et fermeture du fichier

```
CLOSE  4
```

b) **Ouverture, utilisation, fermeture**

Exemple : Soit un programme sous BOS16 s'exécutant avec le numéro d'usager BG (Background).

1°) Ouverture **en lecture seulement** du fichier créé dans l'exemple précédent.

```
OPEN OLD  22, FMSSTD- : S, D2, S= 1, W = 0
```

2°) Utilisation : lecture des articles en séquentiel par portions d'article.
Exemple : lecture de l'article IDPSTD

```
IREAD     22, ABU, LBU = 0, IDPSTD

READ      22, ABU, LBU = 80   jusqu'à fin d'article
                              PR = `6001
```

Remarque : Pendant cette phase d'utilisation, le fichier pourrait être accessible en lecture par d'autres usagers, puisque la FAU n° 22 a été créée en lecture seulement, ou utilisée par le même usager en multi-accès lecture.
Toute demande d'accès au fichier en écriture sera rejetée quelqu'en soit l'usager.

3º) Destruction de la FAU nº 22 et fermeture du fichier FMSSTD- : S/D2

```
CLOSE    22
```

Remarque : A partir de maintenant le fichier est de nouveau accessible en écriture par un quelconque usager.

c) **Ouverture, utilisation, destruction**

Exemple : Soit un fichier Permanent Séquentiel NON Simultané Protégé en écriture, identification du fichier CARAM-EL/D3. Le fichier est utilisé en multi-accès lecture, par 3 tâches Temps Réel T1, T2, T3 (même usager).

1º) **Ouverture** par les 3 tâches en lecture

```
T1  :  OPEN OLD  5, CARAM-EL, D3, S = 0, W = 0, RWK = 1
T2  :  OPEN OLD  6, CARAM-EL, D3, S = 0, W = 0, RWK = 1
T3  :  OPEN OLD  7, CARAM-EL, D3, S = 0, W = 0, RWK = 1
```

2º) **Utilisation**

```
Tâche T1
                READ    5, ABU, LBU
Tâche T2
                READ    6, ABU, LBU
Tâche T3
                READ    7, ABU, LBU
```

Remarque : Si l'une quelconque des trois tâches désire détruire le fichier, cela lui est impossible pour 3 raisons.
— l'**unité d'accès** au fichier qu'elle a créée est en **lecture seulement**
— le fichier est **protégé** en **écriture**
— le fichier **n'est pas** en **mono-accès.**
Supposons DELET 6, FMS fournit alors le compte rendu d'erreur PR = `6014 : Protection écriture.

Remarque : Si le fichier avait été Direct
— NON Protégé en écriture
— utilisé par ces trois tâches en multi-accès écriture.
Supposons DELET 6, FMS fournit alors le compte-rendu d'erreur PR = `601E fichier occupé.

3º) **Fermeture** du fichier par les 3 tâches :

```
T1  :  CLOSE 5
T2  :  CLOSE 6
T3  :  CLOSE 7
```

Remarque importante : l'une quelconque des trois tâches a la possibilité de détruire les 3 unités d'accès à la fois, par exemple :

T1 exécute
```
CLOSE 5
CLOSE 6
CLOSE 7
```

Cela lui est possible puisque toutes les tâches appartiennent au même usager. Il faut cependant synchroniser les tâches pour ne pas obtenir le compte-rendu `601F : Requête en cours sur la même FAU. Cela est dû au fait que ce qui correspond à 1 usager pour FMS peut être constitué de plusieurs tâches, plusieurs contextes différents activables par la machine. (Identification de FAU : USR + FNUM).

4o) Destruction du fichier par la Tâche T1 (ou par un autre usager). Demande d'accès au fichier en lecture, et en mono-accès.

```
OPEN OLD   5, CARAM-EL, D3, S = 0, W = 0, RWK = 0
```

A l'aide de la requête ALTER l'usager demande à la fois de ne plus protéger le fichier en écriture et d'autoriser les écritures par la FAU nº 5.

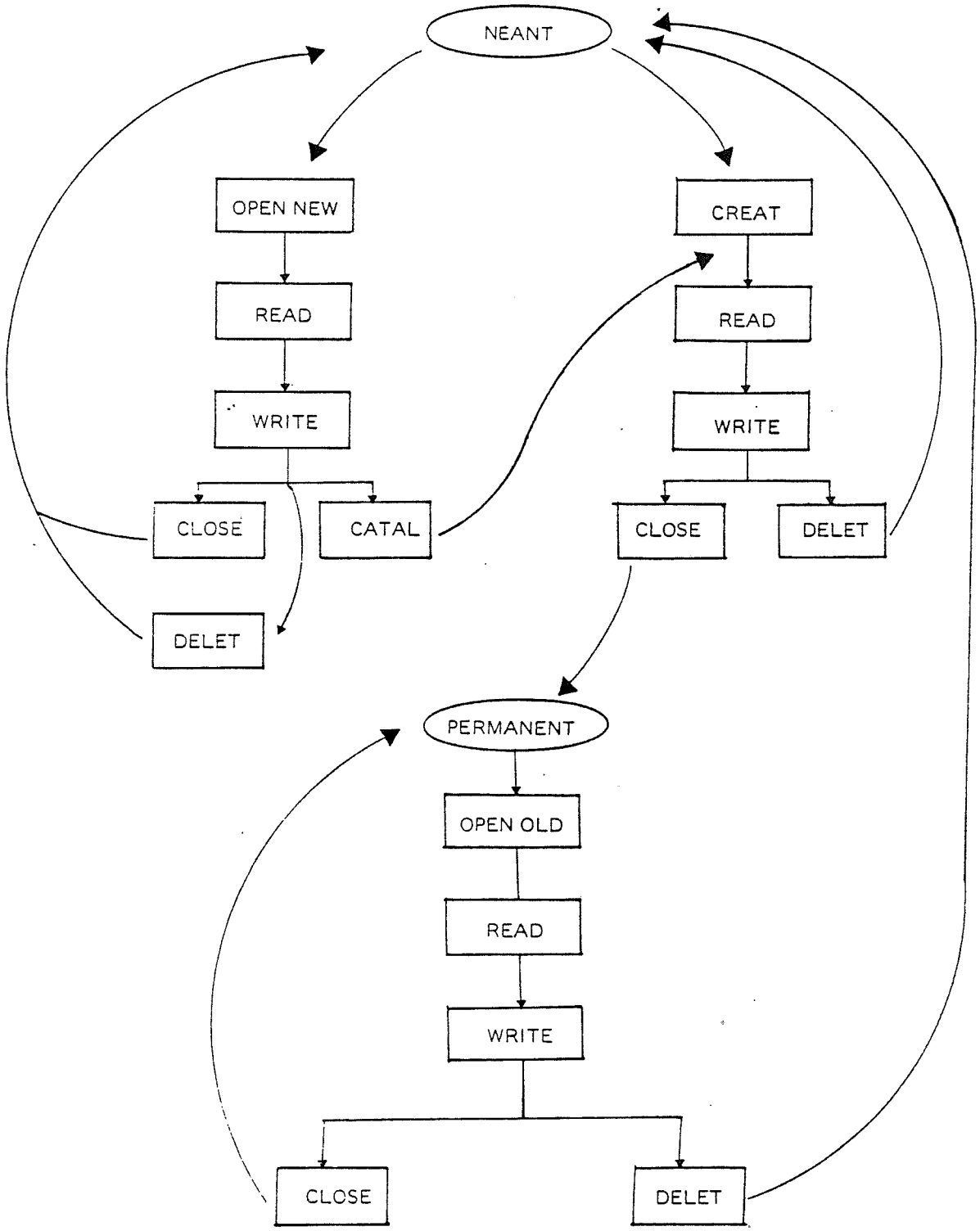
```
ALTER      5, FU = D3, S = 0, W = 1, RWK = 0
```

Remarque : Pour une requête ALTER il faut repréciser le support SU/FU et faire attention au fait que les valeurs de S et RWK sont automatiquement prises en compte. Ici le fichier sera donc encore NON Simultané et en mono-accès.

Destruction proprement dite du fichier et de la FAU nº 5.

```
DELET      5
```

4.1.4 - Schéma général d'enchainement



4.2 - FONCTIONS LOGIQUES

Généralités

Le module OPEN CLOSE fournit aux usagers 13 requêtes d'accès au niveau fichier spécifiees a l'aide du paramètre FONCT. Ce module fournit également 3 autres requêtes exclusivement réservées aux superviseurs des différents systèmes d'exploitation. FMS en interdit l'accès à tout programme ou tâche s'exécutant en mode esclave. En mode maître il est donc impérativement demandé de respecter la règle suivante sur la valeur du paramètre FONCT de chaque requête.

FONCT = [0 à 10, 14, 15]

Le non respect de cette règle détériore gravement le fonctionnement du système.

Parmi les 10 requêtes fournies 3 d'entre elles permettent de créer une Unité d'Accès (FAU) et son numéro d'identification FNUM.

OPEN NEW, OPEN OLD, CREAT

Toutes les autres requêtes (sauf EOJ) utilisent une FAU existante, et concernent le fichier accessible par celle-ci.

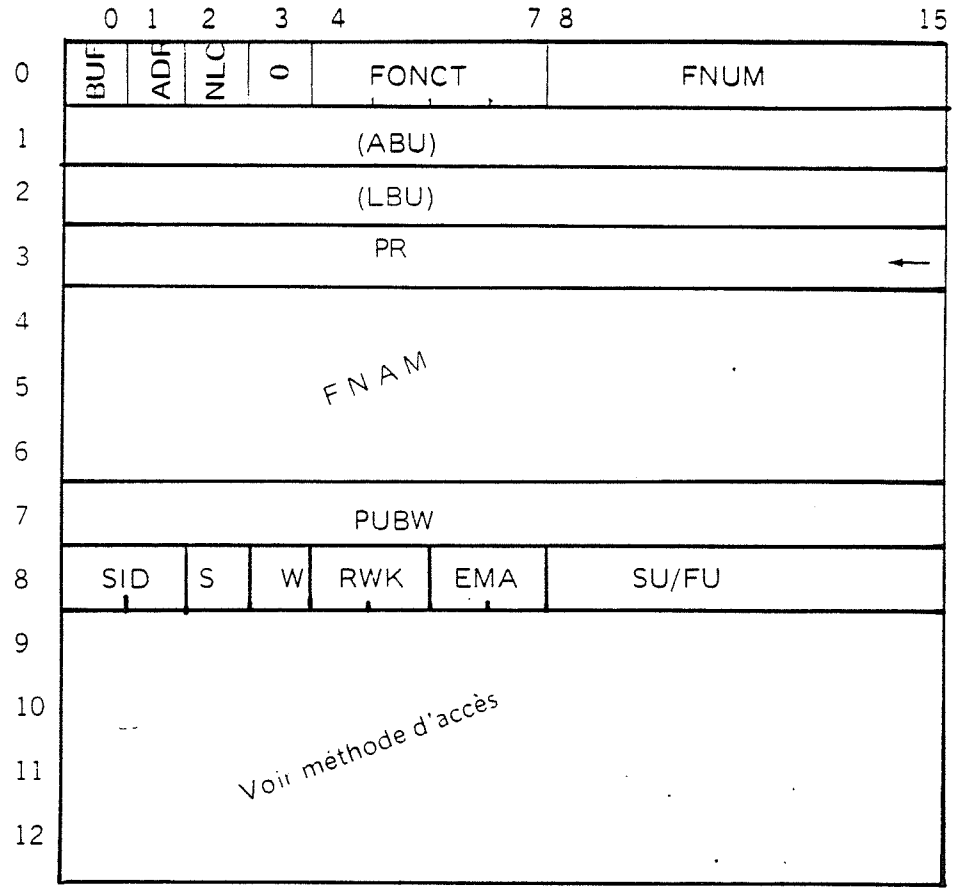
Le module OPEN CLOSE est non réentrant au sens ou lorsqu'il traite une requête pour le compte d'une tâche x , toute autre tâche faisant également appel à une requête de l'OPEN CLOSE est mise en attente jusqu'à ce que la requête de la Tâche x soit terminée.

L'interface d'appel d'une requête du niveau Fichier, traitée par le module OPEN CLOSE est en PL1600.

```
RA : = @ FCB
SVC    (FMS) ;                    où FMS = '38
```

Profondeur de Kstore nécessaire 70 mots.

FCB :



Conventions

- paramètre de retour fourni par FMS
- 0 n bits à zéro
- hachures spécifiant zone non utilisée (au niveau de chaque requête).
- (ABU) paramètre optionnel (voir méthode d'accès)
- numéro d'organisation logique 4 bits : | EMA | SID |

Description générale des paramètres

BUF	=	1 l'utilisateur fournit un buffer de travail, à l'aide des paramètres ABU, LBU. Voir dans chaque méthode d'accès si l'utilisateur doit fournir un buffer de travail.
ADR	=	1 Option accès direct rapide demandeé pour cette FAU.
NLC	=	1 Option écriture sans relecture de contrôle et sans reécriture de correction. Valable pour toutes les écritures de la FAU.
FONCT	:	Numéro de 1 à 10 spécifiant la fonction de la requête. ex. OPEN NEW : FONCT = 1.
FNUM	:	Numéro d'identification de l'unité d'accès au fichier. En création voir les numéros autorisés par les superviseurs.
ABU	:	Adresse du buffer de travail sur 16 bits (pas de bit index).
LBU	:	Longueur du buffer de travail. Un nombre pair d'octets sur 16 bits, devant en plus vérifier d'autres regles selon la méthode d'accès.

PR : Compte-rendu fourni par FMS

FNAM : Nom du fichier. 6 caractères ASCII pairs à raison de 2 caractères par mot.
26 lettres A à Z
10 chiffres 0 à 9
4 caractères spéciaux
(:) (;) (<) et le caractère (Nul) en fin seulement.

PUBW : Nom de catalogue. 2 caractères ASCII pairs. Même règle que pour le FNAM mais le (Nul) peut être en position quelconque.

SU/FU : Numéro d'Unité Symbolique ou d'Unité Fonctionnelle

EMA/SID : 4 bits qui dans cet ordre spécifient le numéro d'organisation logique du fichier ex : Séquentiel = 0
$0 \leqslant EMA/SID \leqslant 10$
(Voir chapitre 3 : Description Générale).

S = 1 $\Longleftrightarrow$ Fichier Permanent Simultané.

W : booleen d'écriture
en création : W = 0 $\Longleftrightarrow$ le fichier sera protégé en écriture

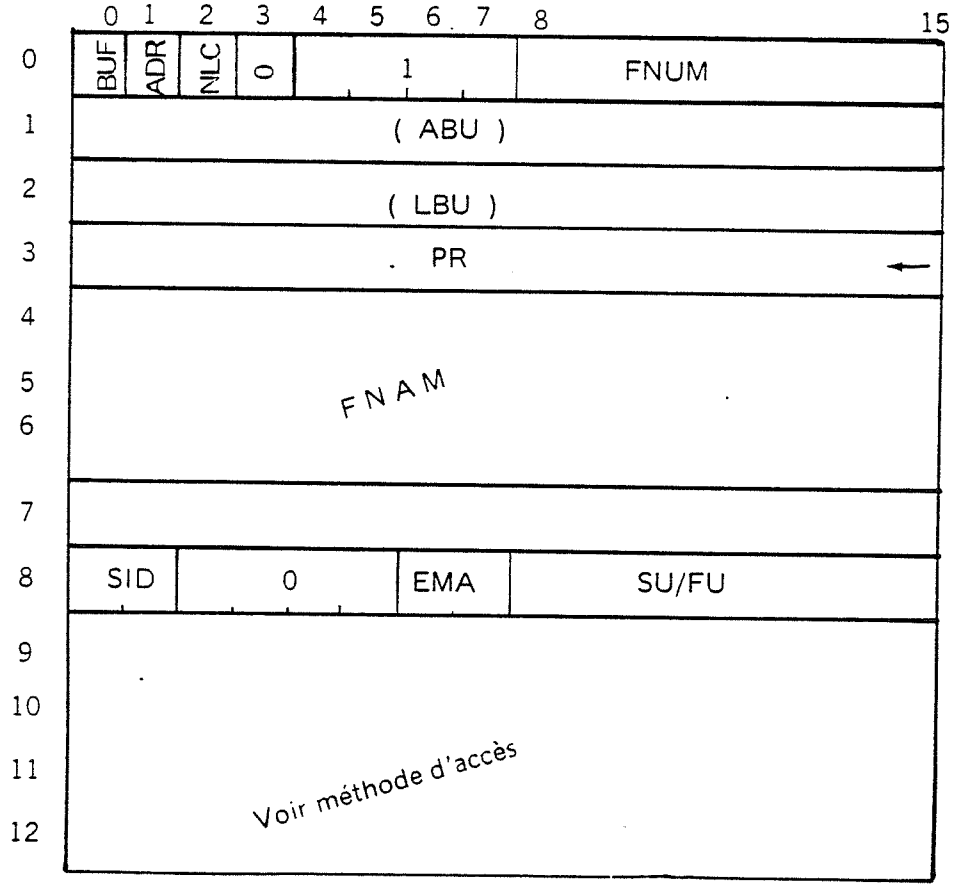
à l'ouverture : W = 1 $\Longleftrightarrow$ l'unité d'accès est demandée en lecture et en écriture

RWK : Clé d'accès à n'utiliser que pour l'ouverture d'un permanent NON Simultané (Voir OPEN OLD, et chapitre 2.4.4 § d) : Le fichier Permanent NON Simultané).

4.2.1 - OPEN NEW : Création d'un Temporaire

Nom	OPEN NEW
But	Créer un fichier Temporaire et créer une unité d'accès à ce fichier.

Appel : RA : = @FCB ; SVC (FMS) ; où FMS = `38



Compte-rendu

OPEN NEW

Valeur de PR	Signification
0	Primitive correctement exécutée
`600B	**FAU existante**
`600D	**Fichier existant**
`6017	**Fichier trop long** : Voir méthode d'accès
`601F	**Primitive en cours**
`6020	**Zone de Pavé saturée**
`6021	**FU saturée**
`6028	**Erreur de Syntaxe** : (@FCB, FONCT, [ABU, LBU] FNAM " PUBW ", EMA/SID, paramètres Méthode d'Accès)

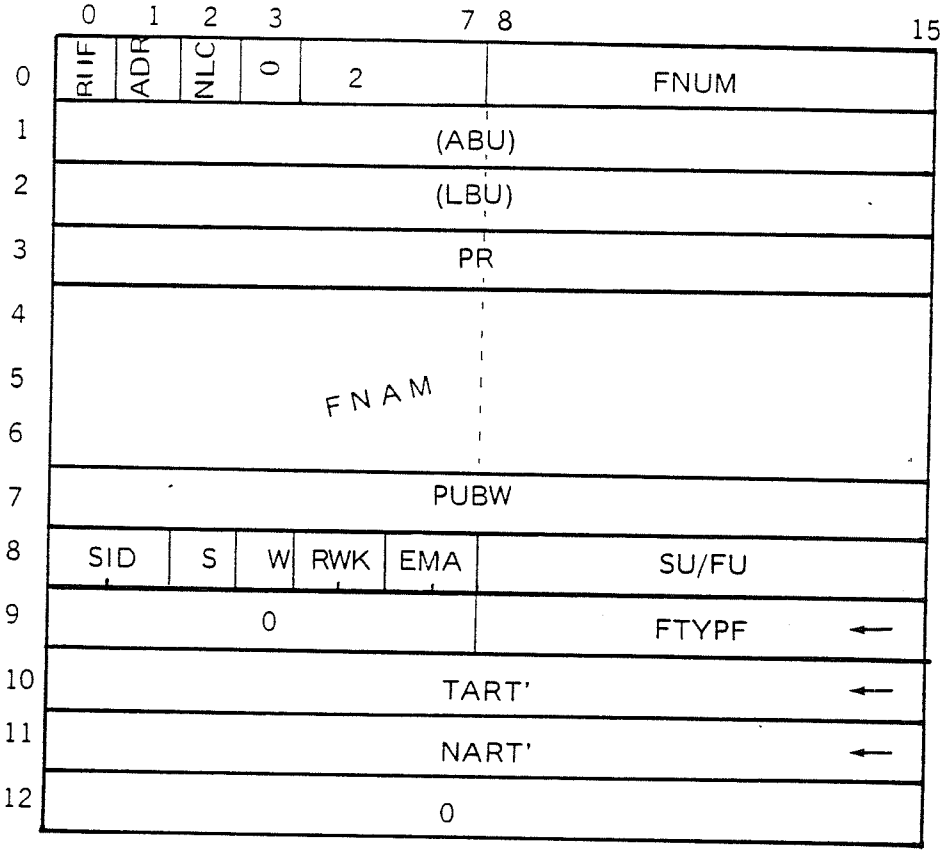
	`6029	Adresse de FCB invalide
	`602A	SU ou FU non gérée par FMS
	`602B	Méthode d'accès non gérée
Erreurs graves		
	`6032	Informations Système Invalide
	`6033	"
	`6034	"
	`6035	FU Verrouillée par IOCS
	`4...	Erreur hardware : `4000 + mot d'état PU

4.2.2 · OPEN OLD : Ouverture d'un Permanent

Nom	OPEN OLD	OPEN F
But	Ouvrir un fichier Permanent et créer une unité d'accès à ce fichier.	

Appel : RA : = @ FCB ; SVC (FMS) ; où FMS = '38

FCB



Il existe 3 formes d'OPEN OLD

a) Ouverture d'un Permanent Simultané

EMA-SID : 0 où le numéro d'organisation logique du fichier
S = 1 l'utilisateur doit répéter le fait que le fichier est Simultané
W : W = 1 l'usager désire lire et écrire sur le fichier.
W = 0 l'usager désire lire seulement sur le fichier.
RWK = 0 ·
Les mots 9, 10, 11 ne seront pas chargés par FMS.

b) Ouverture d'un Permanent NON Simultané

EMA-SID = 0 où le numéro d'organisation logique du fichier.
S = 0 l'utilisateur doit répéter le fait que le fichier est NON Simultané.
W : même chose que pour un Simultané
RWK : clé d'accès. (Voir chapitre 2.4.4).
RWK = 0 : l'usager désire utiliser le fichier en mono-accès.
RWK = 1 : l'usager désire utiliser le fichier en multi-accès lecture.
RWK = 3 : l'usager désire utiliser le fichier en multi-accès écriture.
Les mots 9, 10, 11 ne seront pas chargés par FMS.

Ouverture d'un permanent en mode séquentiel (OPEN F)

Le but de cet OPEN OLD est de réaliser un traitement séquentiel sur le fichier quelque soit son numéro d'organisation logique. L'utilisation se fait en séquentiel Pur statique pour tout fichier autre que séquentiel, en séquentiel pur dynamique pour le séquentiel.

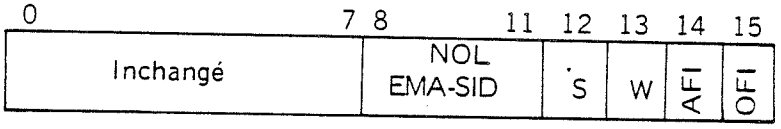
— L'utilisateur charge le FCB comme précédemment avec cependant :

BUF = 0 Si pas de buffer (valable pour toute méthode d'accès).

EMA-SID = 'F

— FMS rend à l'utilisateur dans le FCB :

FTYPF : Cet octet se trouve dans le descripteur de fichier, lorsque celui-ci est en mémoire centrale. (voir manuel d'utilisation paragraphe 2-2-3 a). L'octet gauche du FCB est inchangé.



TART' : Ces deux valeurs se trouvent dans le descripteur de fichier sur disque comme en mémoire
NART' : centrale. Attention, TART' est exprimé en mots. (voir MR Paragraphe 2-2-3, M.U Paragraphe 2-2-3 d).

LID : longueur de la clé pour un fichier séquentiel indexé (FCB12).

- **Ouverture d'un permanent en mode séquentiel et en FAU publique sans FNUM** (OPEN `E)

Le but de cet OPEN est identique à celui de l'OPEN F avec la possibilité supplémentaire de pouvoir ne pas préciser de FNUM. Celui-ci sera géré dynamiquement par FMS et rendu à l'utilisateur en fin de primitive.

Par ailleurs, il est possible de préciser le numéro de FNUM à partir duquel FMS commence sa recherche.

— Le FCB est initialisé comme pour l'OPEN OLD avec :
  . EMA-SID = `E
  . FNUM = 0 ou le numéro à partir duquel FMS recherche le FNUM.

— FMS rend à l'utilisateur les mêmes informations que l'OPEN `E avec en plus le n° de FNUMP alloué à l'accès au fichier.

- **Récupération des caractéristiques d'un fichier** (OPEN `D)

— L'OPEN `D permet de récupérer les FTYPF, TART', NART' et LID (LID si OPEN `D d'un fichier séquentiel indexé).

— De façon imagée, l'OPEN `D effectue un OPEN `E suivi d'un CLOSE et ceci en une seule primitive.

— Le FCB est initialisé de façon identique à l'OPEN OLD avec

EMA-SI D = `D
FNUM = 0

OPEN OLD

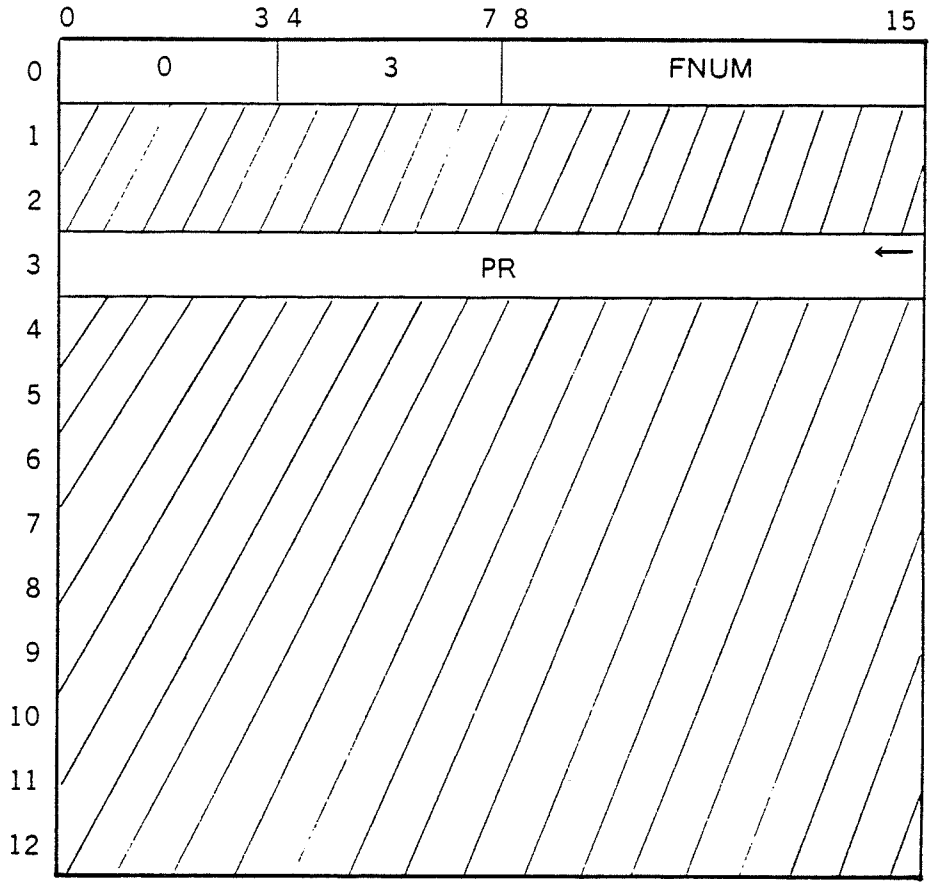
Compte-rendu	Valeur de PR	Signification
	0	**Primitive correctement exécutée** : le pointeur courant est positionné en début de fichier.
	'600B	**FAU existante**
	'600C	**Fichier inexistant**
	'6014	**Protection écriture** : 2 cas d'erreur 1°) le fichier est protégé en écriture 2°) la méthode d'accès ne permet pas de multi-accès écriture, ou lecture avec la 1ère FAU en écriture. la primitive est ineffective.
	'6015	**Permanent de nature différente**
	'601E	**Fichier occupé :** **Simultané :** l'accès est demandé en écriture et le fichier est déjà utilisé en lecture, ou inversement. **NON simultané :** le fichier est déjà utilisé par un autre usager, ou le fichier est bien utilisé par le même usager mais dans un autre contexte de multi-accès (RWK différentes). La primitive est ineffective.
	'601F	**Primitive en cours**
	'6020	**Zone de Pavé saturée :**
	'6028	**Erreur de syntaxe :** (FCB, FONCT, (ABU, LBU) FNAM, PUBW, RWK).
	'6029	**Adresse de FCB invalide**
	'602A	**SU ou FU non gérée par FMS**
	'602B	**Méthode d'accès non gérée**
Erreurs graves	'6032	**Informations système invalides**
	'6034	" " "
	'6035	**FU verrouillée par IOCS**
	'4...	**Erreur hardware** : '4000 + mot d'état PU.

4.2.3 - CLOSE : fermeture

Nom	CLOSE
But	Détruire une unité d'accès à un fichier (FAU) et, détruire un fichier Temporaire, ou fermer un fichier Permanent. Remarque : le fichier ne sera effectivement fermé que s'il n'est utilisé par aucune autre FAU.

Appel RA : = @FCB ; SVC (FMS) ; où FMS = `38

FCB



Compte-rendu CLOSE

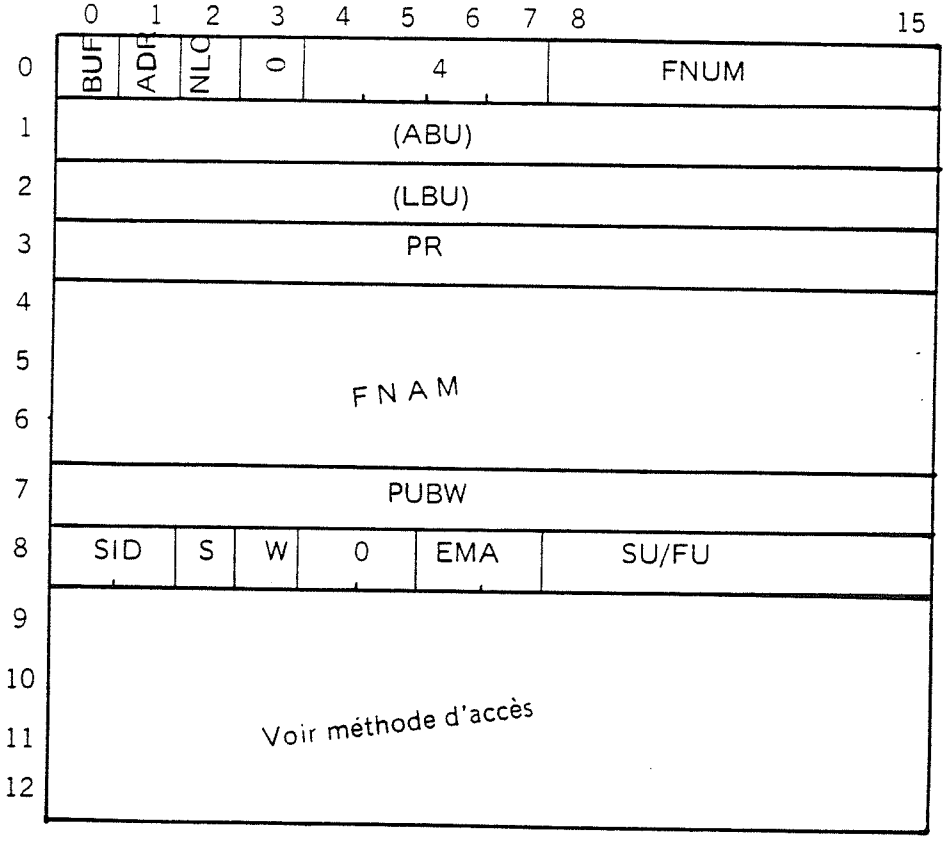
Valeur de PR	Signification
0	Primitive correctement exécutée
`600A	FAU inexistante
`601F	Primitive en cours
`6028	Erreur de Syntaxe :
`6029	Adresse de FCB invalide
'602B	Mèthode d'accès non gérée

Erreurs graves	`6032	Informations Systèmes Invalides
	`6033	"
	`6034	"
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU.

4.2.4 - CREAT : Créer un fichier Permanent

Nom	CREAT
But	Créer un fichier Permanent et une Unité d'Accès au fichier.
Appel	RA : = @ FCB ; SVC (FMS) ; où FMS = `38

FCB



Remarque : RWK = 0 sauf pour certaines Méthodes d'Accès.
W = 0⇔fichier protégé en écriture (après close).

Compte-rendu

CREAT

Valeur de PR	Signification
0	Primitive correctement exécutée
`600B	FAU existante
`600D	Fichier existant
`6017	Fichier trop long :
`601F	Primitive en cours
`6020	Zone de Pavé saturée
`6021	FU saturée
`6022	Table des fichiers saturée

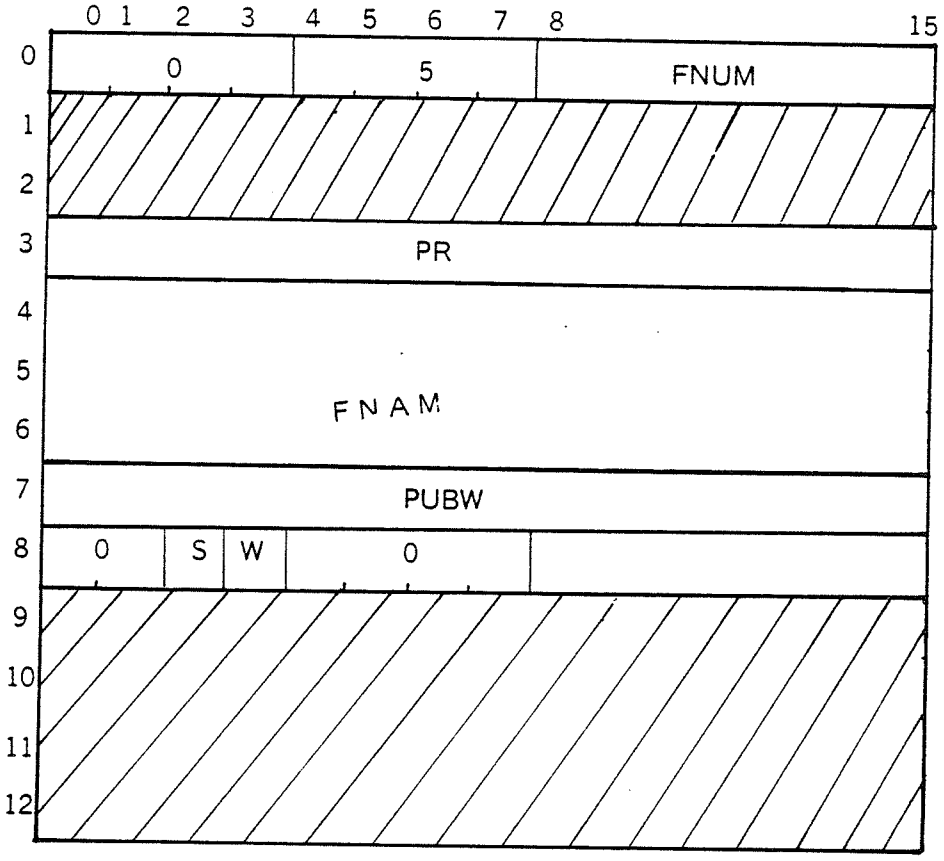
	`6028	Erreur de Syntaxe : (@ FCB, FONCT, [ABU, LBU] FNAM, PUBW, EMA/SID, paramètres Méthode d'Accès).
	`6029	Adresse de FCB invalide
	`602A	SU ou FU non gérée par FMS
	`602B	Méthode d'Accès non gérée
Erreurs graves	`6032	Informations Systèmes Invalides
	`6033	"
	`6034	"
	`6035	FU verrouillée par IOCS
	`4...	Erreur hardware : `4000 + mot d'état PU.

4.2.5 · CATAL : Transformation d'un Temporaire en Permanent

Nom	CATAL
But	Transformer un fichier Temporaire en fichier Permanent. Au point de vue utilisation l'unité d'accès se trouve dans les mêmes conditions qu'après un CREAT.

Appel RA : = ⌀ FCB ; SVC (FMS) , où FMS = `38

FCB



Compte-rendu CATAL

Valeur de PR	Signification
0	Primitive correctement exécutée
`600A	FAU inexistante
`600D	Fichier existant
`6018	Incompatibilité Primitive-Fichier : le fichier accessible par cette FAU n'est pas Temporaire. La primitive est ineffective.
`601F	Primitive en cours
`6022	Table des fichiers saturée
`6028	Erreur de Syntaxe : (⌀ FCB, FONCT, FNAM, PUBW).
`6029	Adresse de FCB invalide

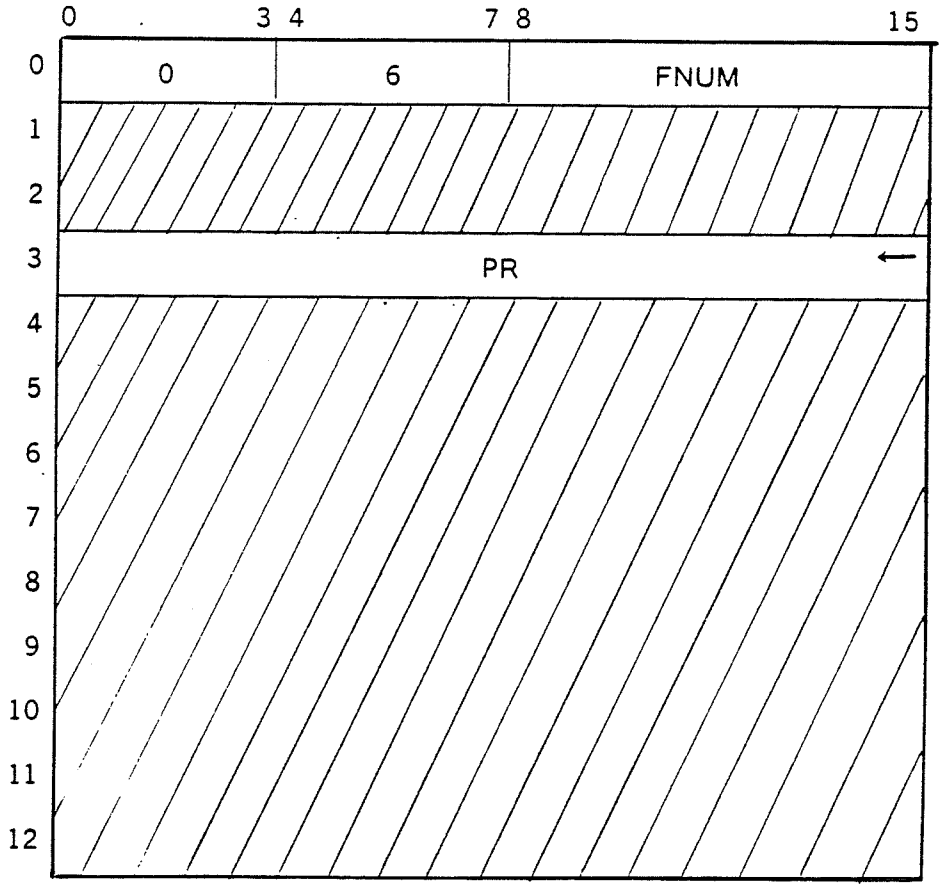
Erreurs graves	`6032	Informations Systèmes Invalides
	`6034	"
	`6035	FU verrouillée par IOCS
	`4...	Erreur hardware : `4000 + mot d'état PU.

4.2.6 - DELET : Détruire un fichier

Nom	DELET
But	Détruire un fichier Temporaire ou Permanent et détruire l'unité d'accès correspondante. **Remarque** : Voir conditions d'exécution au niveau des comptes-rendus `6014 et `601E

Appel RA : = @ FCB ; SVC (FMS) ; où FMS = `38

FCB



Compte-rendu

DELET

Valeur de PR	Signification
0	**Primitive correctement exécutée**
`600A	**FAU inexistante**
`6014	**Protection écriture** : Le DELET n'est pas effectué lorsque la FAU est en lecture seulement. La primitive est ineffective.
`601E	**Fichier occupé** : Le DELET n'est effectué que si le fichier est utilisé par un seul usager et à l'aide d'une seule FAU à l'instant où la requête DELET est exécutée. La requête est ineffective.

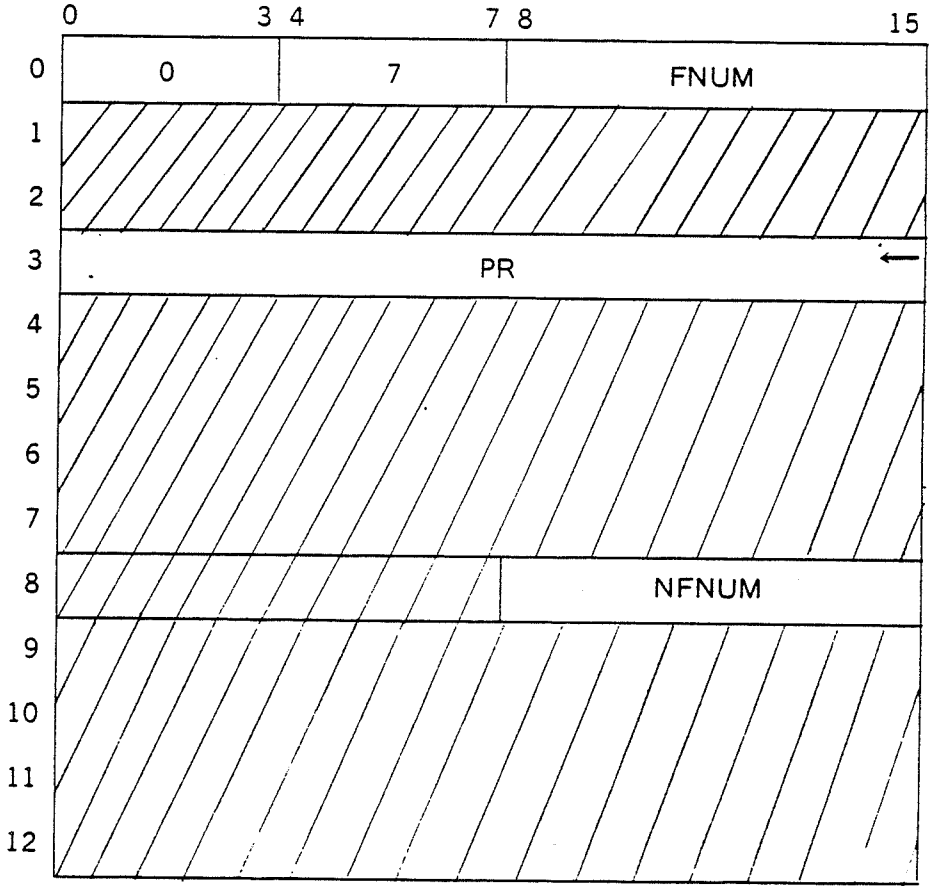
	`601F	Primitive en cours
	`6028	Erreur de Syntaxe : (@ FCB, FONCT)
	`6029	Adresse de FCB invalide
Erreurs graves	`6032	Informations Systèmes Invalides
	`6033	"
	`6034	"
	`6035	FU verrouillée par IOCS
	`4...	Erreur hardware : `4000 + mot d'état PU.

4.2.7 - RENUM : Renuméroter une FAU (FNUM)

Nom	RENUM
But	Changer le numéro FNUM d'une Unité d'Accès à un fichier Temporaire ou Permanent.

Appel RA : = @ FCB ; SVC (FMS) ; où FMS = `38

FCB



NFNUM : nouveau numéro de la FAU

Compte-rendu

RENUM

Valeur de PR	Signification
0	**Primitive correctement exécutée**
`600A	**FAU inexistante**
`600B	**FAU existante** : Cet usager possède déjà une FAU de ce numéro (NFNUM). La primitive est ineffective.
`601F	**Primitive en cours**
`6028	**Erreur de Syntaxe** : (@ FCB, FONCT)
`6029	**Adresse de FCB invalide.**

4.2.8. ALTER : Modifier les attributs d'un fichier

La requête ALTER permet de modifier les attributs d'un fichier permanent.

a) Si A = 0 modification de S, W, RWK : ALTER SIMPLE.
Modification des informations statiques
— la nature du permanent
SI S = 0 le fichier devient NON simultané
SI S = I le fichier devient simultané
— la protection écriture
SI W = 0 le fichier devient protégé en écriture
SI W = I le fichier devient accessible en écriture
Modification des informations dynamiques
— L'accès en écriture
SI W = 0 la FAU n'est plus autorisée en écriture
SI W = I la FAU devient autorisée en écriture
— les conditions de multi-accès et.ou de partage
SI RWK = 0 tout nouvel OPEN OLD sur le fichier sera refusé
SI RWK = 1 seuls les OPEN OLD en lecture seront acceptés
SI RWK = 3 tout OPEN OLD sera accepté en accord cependant avec le booleur de protection écriture.

Attention :
La requête ALTER SIMPLE (A=0) modifie les trois paramètres à la fois, la valeur de chaque paramètre est prise en compte comme étant la nouvelle valeur. Autrement dit pour ne pas changer un paramètre il faut répéter l'ancienne valeur.
Le boolen W modifie à la fois, la protection écriture du fichier et l'utilisation de la FAU en écriture.
La requête ALTER permet de transgresser les règles normales de partage,
L'utilisateur est donc responsable du bon fonctionnement.
— du partage inter-usager pour un fichier permanent simultané selon sa méthode d'accès:
— du multi-accès pour un fichier permanent NON simultané selon sa méthode d'accès.
La requête ALTER permet en particulier a des usagers différents de se partager en écriture et en lecture un fichier statique permanent simultané. Le fichier doit être ouvert par tout usager en lecture seulement et ceux qui désirent écrire utilisent la requête ALTER (A=0, S inchangé, W=1, RWK=1)

b) Si A = 1 modification totale (NOL, TART', NART', OFI, S, W, RWK) : ALTER TOTAL.
Modification de S, W, RWK (même traitement que pour A = 0)
Modification de NOL, TART', NART', OFI
— Le numéro d'organisation logique (NOL = EMA-SID)
EMA-SID contient le nouveau numéro d'organisation logique
— TART',NART', (voir définition paragraphe 2-2-3) sont les nouvelles valeur à mettre dans le DF sur disque en accord avec le numéro d'organisation logique. (TART' doit être exprimé en **mots**)
—OFI numéro d'organisation physique
OFI = 0 organisation séquentielle OFI sera remis à zéro
OFI = 1 organisation directe. Le traitement effectué sera équivalent à celui de FAST. Création d'un TLGet OFI:= 1)

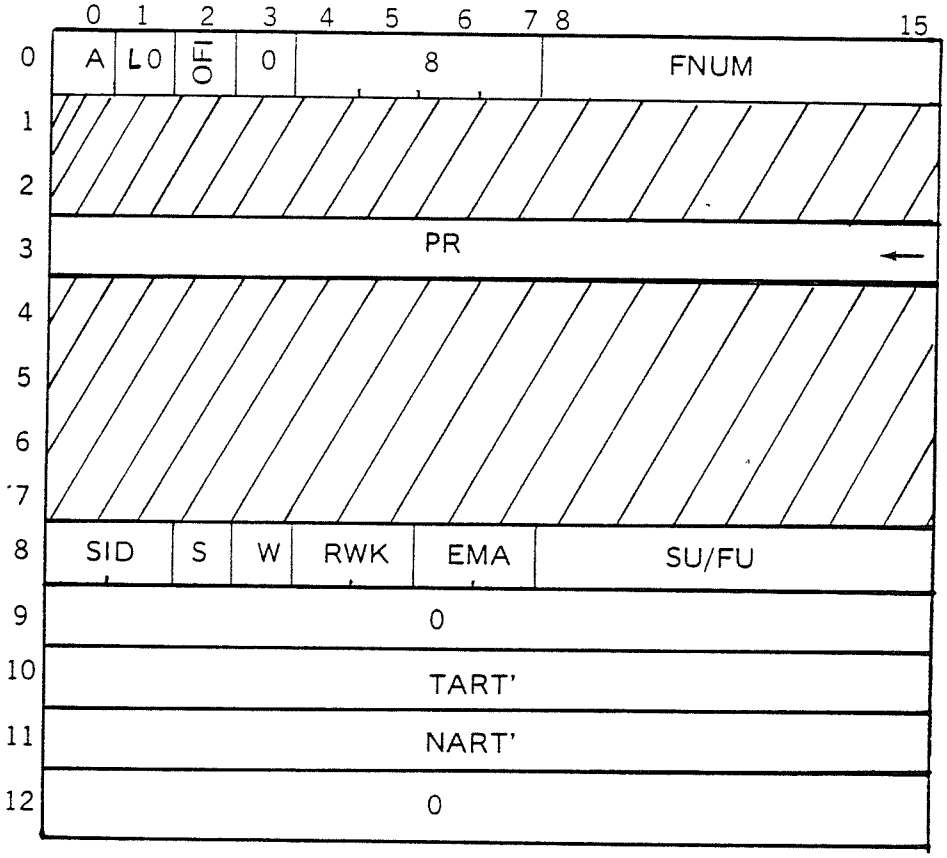
Attention :
Le ALTER total ne fonctionne que si le fichier est en mono-accès, et la RWK = 3 n'est possible que pour les fichiers statiques. Pour le ALTER total tous les paramètres seront pris en compte comme étant les nouvelles valeurs à mettre à jour. Pour le ALTER simple (A= 0) seuls S, W, RWK, doivent être chargés dans le FCB.
Il est vivement conseillé de **fermer** le fichier après un **ALTER TOTAL.**

b) Interface de programmation

Nom	ALTER
But	Modifier S, W d'un fichier et W, RWK d'une FAU.

Appel RA := @FCB ; SVC (FMS) ; où FMS = '38

FCB



Attention : Selon A toutes les valeurs sont prises en compte à la fois comme étant les nouvelles valeurs.

Paramètre d'entrée :

LO

— Si LO = 0 la requête ALTER est effectuée sur disque normalement.
— Si LO = 1 la requête ALTER est dite logique.
Les modifications ne sont faites qu'en mémoire centrale. Cela permet par exemple de modifier temporairement en mémoire le numéro de méthode d'accès afin d'accéder physiquement au contenu du fichier par la requête DSEL du Direct et les requêtes du séquentiel.

Compte-rendu

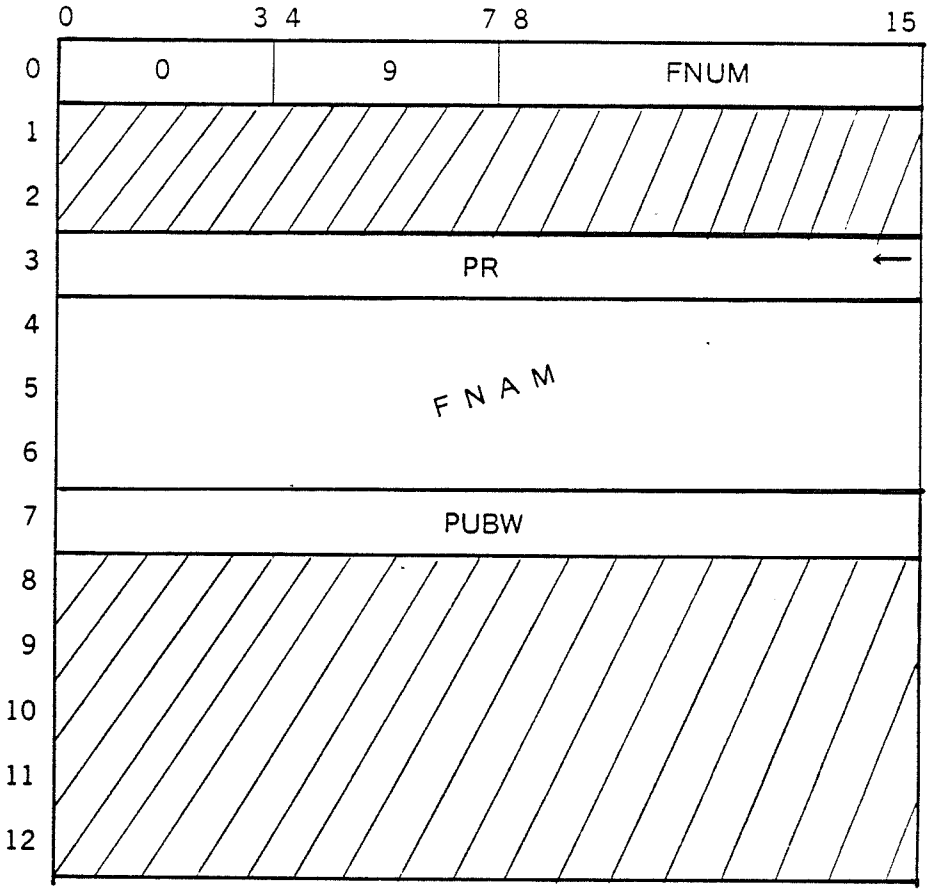
ALTER

Valeur de PR	Signification
0	**Primitive correctement exécutée**
`600A	**FAU inexistante**
`6018	**Incompatibilité primitive-fichier :** Le fichier n'est pas permanent ; la requête est ineffective.
`6014	**Protection écriture :** La requête ineffective. ALTER Total (A = 1) : la FAU n'est pas en écriture, ou le fichier est ouvert avec RWK/=0, ou les nouvelles RWK (du FCB) ne sont pas nulles.
`601E	**Fichier occupé :** La requête est ineffective. ALTER Total (A = 1), le fichier doit être en mono-accès.
`6017	**Fichier trop long** . ALTER Total OF1 = 1, le fichier fait plus de 128 granules.
`601F	**Primitive en cours**
`6028	**Erreur de syntaxe : (@ FCB, FONCT) (EMA SID).**
`6029	**Adresse de FCB invalide**
`602A	**SU ou FU non gérée par FMS**
ERREURS GRAVES	
`6034	**Informations systèmes invalides**
`6035	**FU verrouillée par IOCS**
`4.....	**Erreur hardware :** `4000 + mot d'état PU.

4.2.9 - RENAM : Renommer un fichier Permanent

Nom	RENAM
But	Changer le nom FNAM ⊕ PUBW d'un fichier Permanent.
Appel	RA : = @ FCB ; SVC (FMS) ; où FMS = `38

FCB



Attention : les deux valeurs FNAM et PUBW sont prises en compte à la fois.

Compte-rendu

RENAM

Valeur de PR	Signification
0	**Primitive correctement exécutée**
`600A	**FAU inexistante**
`600D	**Fichier existant** : Sur la FU-Support qui contient déjà le fichier il existe un fichier Permanent du nom spécifié dans le FCB : FNAM ⊕ PUBW. La requête est ineffective.
`6014	**Protection écriture**
`6018	**Incompatibilité primitive-fichier** : Le fichier n'est pas Permanent. La requête est ineffective.

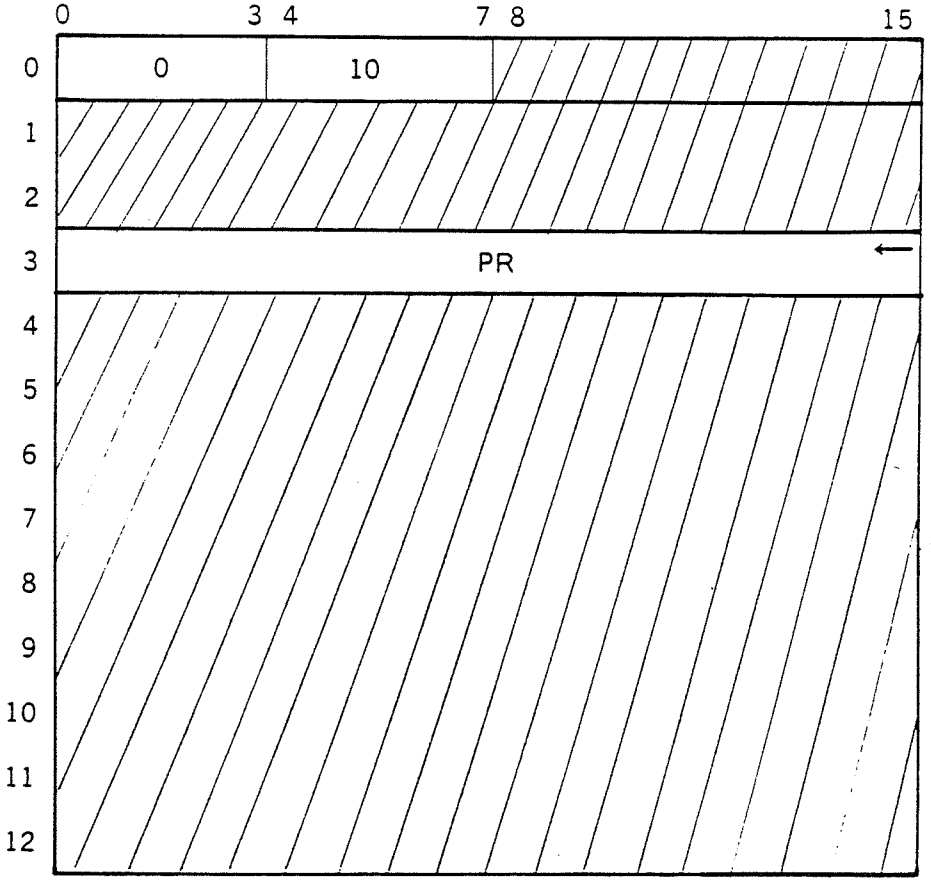
	`601E	**Fichier occupé.** La requête RENAM n'est exécutée que si le fichier est utilisé par un seul usager et à l'aide d'une seule FAU au moment où la requête RENAM est exécutée. La requête est ineffective.
	`601F	**Primitive en cours**
	`6028	**Erreur de Syntaxe :** (ω FCB, FONCT, FNAM, PUBW)
	`6029	**Adresse de FCB invalide**
Erreurs graves	`6032	**Informations Systèmes Invalides**
	`6034	"
	`6035	**FU verrouillée par IOCS**
	`4...	**Erreur hardware :** `4000 + mot d'état PU.

4.2.10 - EOJ : Fin de Travail d'un usager

Nom :	EOJ (USR)
But	Détruire tous les fichiers Temporaires créés par un usager et les FAU correspondantes, Fermer Tous les Fichiers Permanents ouverts par un usager et détruire les FAU correspondantes de cet usager. L'information USR est fournie par le superviseur.

Appel : RA : = @FCB ; SVC (FMS) ; où FMS = `38

FCB



Compte-rendu

EOJ (USR)

Valeur de PR	Signification
0	Primitive correctement exécutée
`6028	Erreur de Syntaxe : (@FCB, FONCT)
`6029	Adresse de FCB invalide

Erreurs graves	`6032	Informations Systèmes Invalides
	`6033	"
	`6034	"
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU.

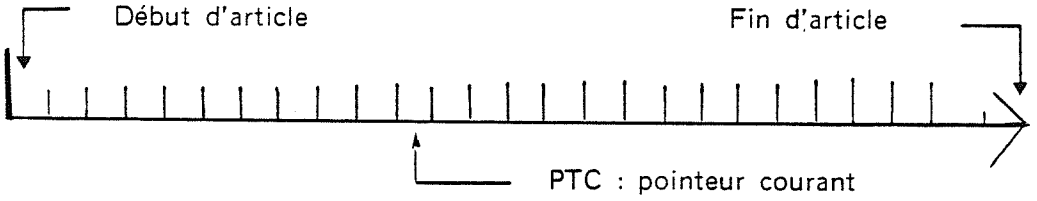
5 — LE FICHIER DE TYPE SEQUENTIEL

5 — LE FICHIER DE TYPE SEQUENTIEL

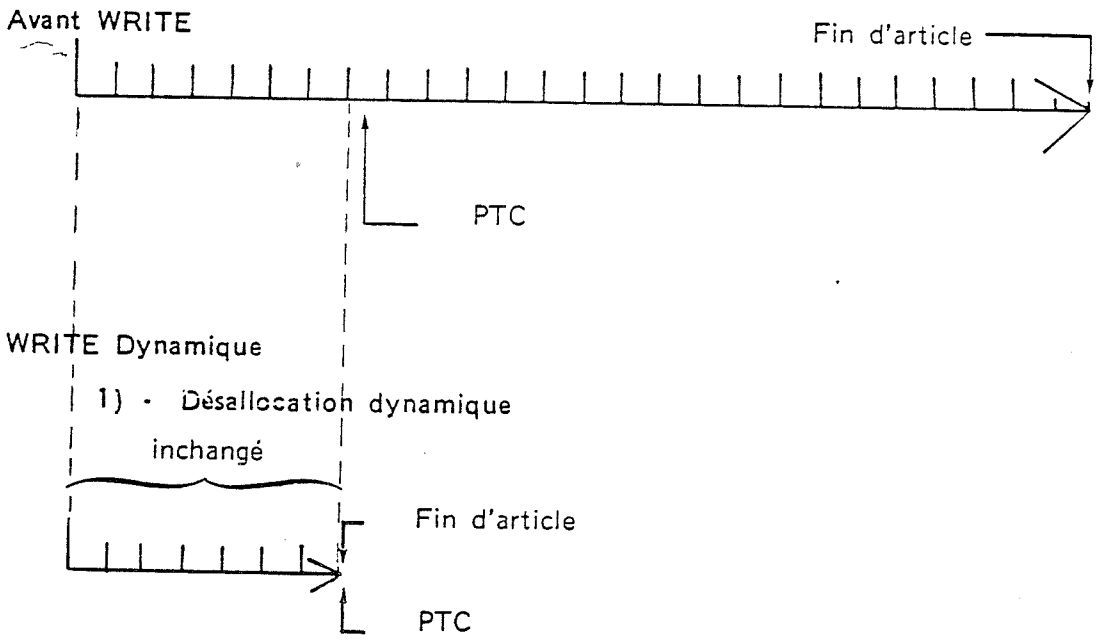
5.1 - ORGANISATION LOGIQUE

L'organisation logique d'un fichier de type séquentiel est définie à la création du fichier, elle possède le **numéro : 0.**

Organisation logique : **Séquentiel.**

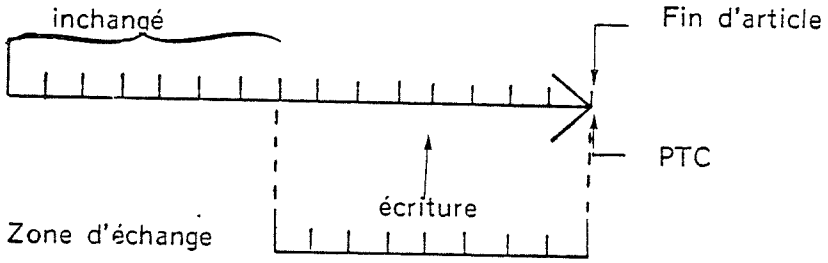


Un fichier séquentiel est constitué d'**un seul** article. Cet article est un **vecteur dynamique** de mots. A la création du fichier il est de longueur nulle. C'est la primitive d'écriture (WRITE) qui permet d'agrandir l'article, et d'allouer dynamiquement la place nécessaire à la quantité d'information apportée par l'ordre d'écriture.
La taille d'un fichier séquentiel n'est limitée que par la place disponible sur le support. FMS gère un pointeur courant (PTC) qui se déplace dans l'article (donc dans le fichier) et pointe toujours sur le prochain **mot** à lire ou à écrire.
Le fichier de type séquentiel simule le fonctionnement d'une **bande magnétique.** C'est la primitive (WRITE) qui permet de détruire la fin de l'article lorsque le pointeur courant ne désigne pas la fin de l'article.



La primitive WRITE permet de désallouer la place physique occupée par la fin de l'article PTC, Fin d'article , et ainsi libérer des granules pour les fichiers du même support.

2) - Allocation Dynamique



La primitive WRITE permet d'agrandir l'article et d'allouer dynamiquement la place nécessaire à l'information qu'il apporte.

5.2 - METHODES D'ACCES

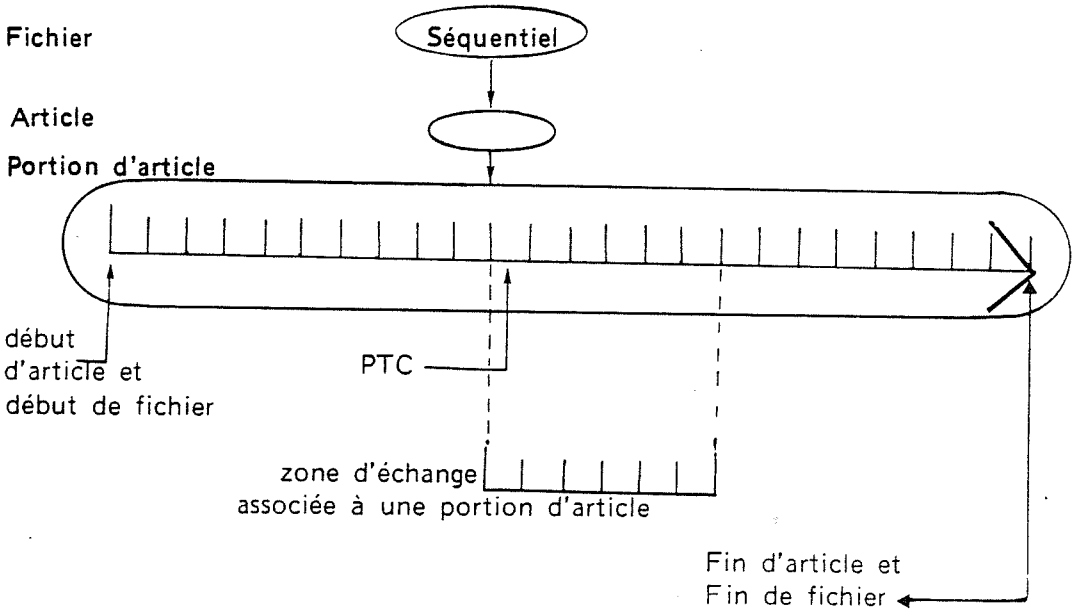
5.2.1 - Le Séquentiel

Le séquentiel est une méthode d'accès capable de gérer des unités d'accès à des articles appartenant à des fichiers d'une quelconque organisation logique. Il est cependant principalement utilisé pour gérer les fichiers séquentiels (**numéro : 0**).
Le séquentiel est **rentrant**. La réentrance est opérationnelle pour des tâches différentes utilisant les primitives d'accès séquentiel par des unités d'accès différentes.
Le séquentiel fournit **7 requêtes** d'accès à une portion d'article : READ, WRITE, SKIPB, SKIPF, REWIND, SKEOA, WERE.
Le séquentiel est capable de fonctionner selon **4 modes différents** :

- Séquentiel Pur Dynamique
- Séquentiel Pur Statique
- Portion d'Article Statique
- Portion d'Article Dynamique.

a) **Séquentiel Pur Dynamique**
Le séquentiel fonctionne de façon autonome par rapport aux autres méthodes d'accès, en séquentiel pur dynamique pour les fichiers séquentiels (**numéro : 0**).

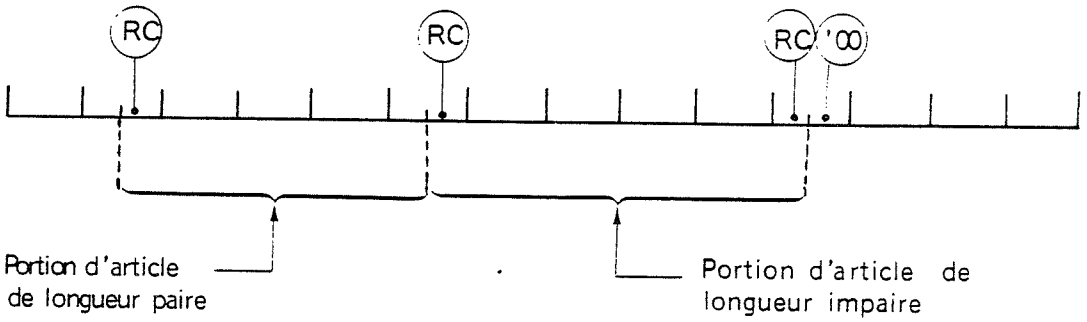
Séquentiel Pur Dynamique.



Le séquentiel gère le pointeur courant (**PTC**) non pas au niveau de l'octet, mais au niveau du **MOT**.
Par convention un fichier séquentiel ne possède qu'**un seul** article.
Définition d'une portion d'article.
A l'intérieur d'un article l'information n'est pas structurée par FMS
Une portion d'article est définie au moment ou l'utilisateur adresse à FMS une requête **de** transfert d'information (READ, WRITE) ou de saut relatif au pointeur courant (SKIPF, SKIPB).

1) - **Compte d'octets.**
Une portion d'article peut être définie à l'aide d'un compte d'octets. La structure de l'article est alors totalement libre et à la charge de l'utilisateur. Les portions d'articles peuvent être de tailles quelconques et **indépendantes** les unes des autres. Cependant l'unité de transfert entre la mémoire centrale et les mémoires secondaires est **le mot.** Un compte d'octet impair est automatiquement apairé par FMS
Plus précisément, FMS traite à l'aide de nombre **entiers** de mots, les zones d'échanges associées aux primitives de transfert d'information (READ, WRITE), et les sauts relatifs au pointeur courant (SKIPF, SKIPB).

2) - **Codes d'arrêt.**
L'utilisateur peut cependant structurer l'information d'un article à l'aide de **codes d'arrêt** de son choix. Une portion d'article étant alors définie selon le schéma suivant.
exemple :



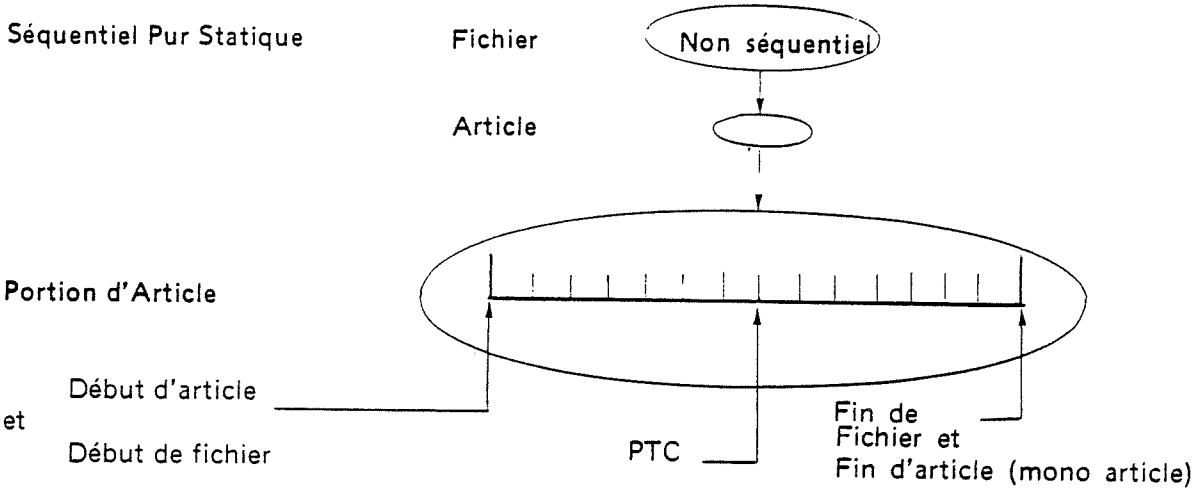
Remarques :
- Une portion d'article de longueur impaire, écrite par code d'arrêt est automatiquement complétée par un octet nul.
- En lecture par code d'arrêt l'octet nul est chargé dans la zone d'échange.
- La longueur d'une portion d'article définie par codes d'arrêt est limitée à 80 octets.

3) - Il est recommandé de définir toutes les portions d'un **même** article de la **même** façon : soit compte d'octets, soit codes d'arrêt.
De plus, lorsque la longueur des portions d'article est impaire, il est recommandé de relire ou sauter les portions d'articles de la même façon qu'elles ont été écrites.

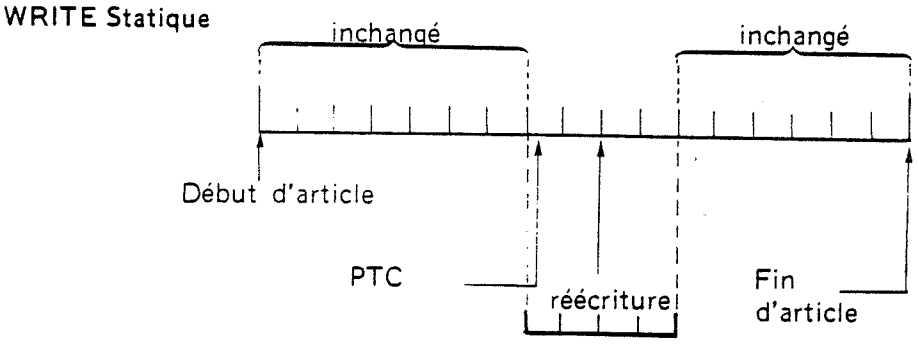
Particularités du Séquentiel Pur Dynamique.
- Les termes Fin d'Article et Fin de Fichier sont équivalents, ainsi que les termes Début d'Article et Début de Fichier.
- Seule la requête WRITE dont le fonctionnement est décrit au chapitre 5.1 Organisation Logique, permet de modifier la longueur d'un fichier séquentiel, détruire la fin de l'article puis agrandir l'article. On parlera de **WRITE Dynamique** ou encore d'**écriture** au sens accès physique (par rapport à réécriture).
- Le mode Séquentiel Pur Dynamique simule donc le fonctionnement de l'accès à une bande magnétique.

b) **Séquentiel Pur Statique**

Le Séquentiel fonctionne en accord avec toutes les autres méthodes d'accès de FMS en séquentiel pur statique pour les fichiers dont l'organisation logique n'est pas séquentielle.



- Toutes les méthodes d'accès autres que le séquentiel permettent d'accéder au contenu de tout le fichier, en ignorant la structure en articles et en considérant qu'il est constitué d'un seul article de taille bornée.
- Par rapport au séquentiel pur dynamique seule la requête WRITE fonctionne de façon différente. On parlera de **WRITE statique** borné par le début et la fin du fichier ou encore de **réécriture** au sens accès physique.

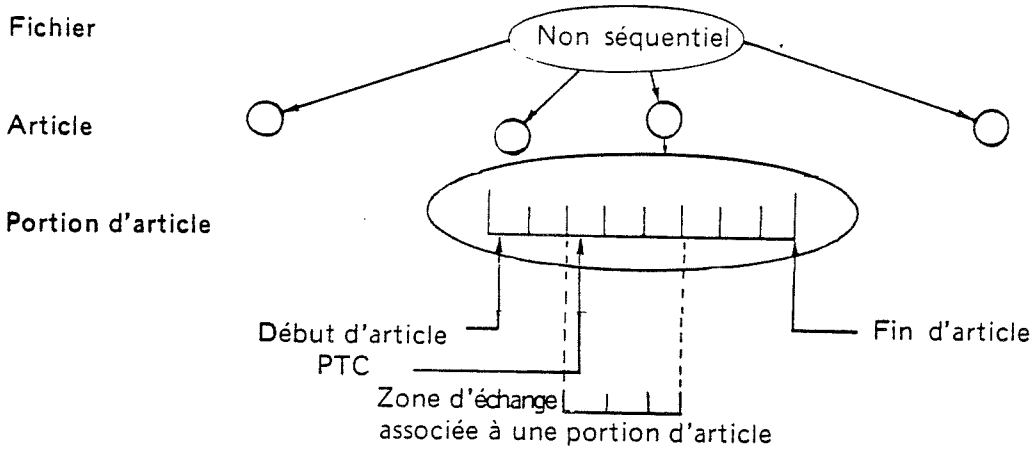


Zone d'échange associée à une portion d'article.

- Le Séquentiel Pur Statique est autorisé après la création d'une Unité d'Accès à un fichier non séquentiel et avant toute requête du niveau article de la méthode d'accès correspondante.
- On peut donc ainsi en accès séquentiel archiver ou reconstituer le contenu d'un fichier de type quelconque.

c) **Portion d'Article Statique.**
Lorsqu'une méthode d'accès non séquentielle a réalisé la sélection d'un article existant elle peut commander au séquentiel le fonctionnement selon le mode Portion d'Article Statique.

Accès Séquentiel à une Portion d'Article Statique.

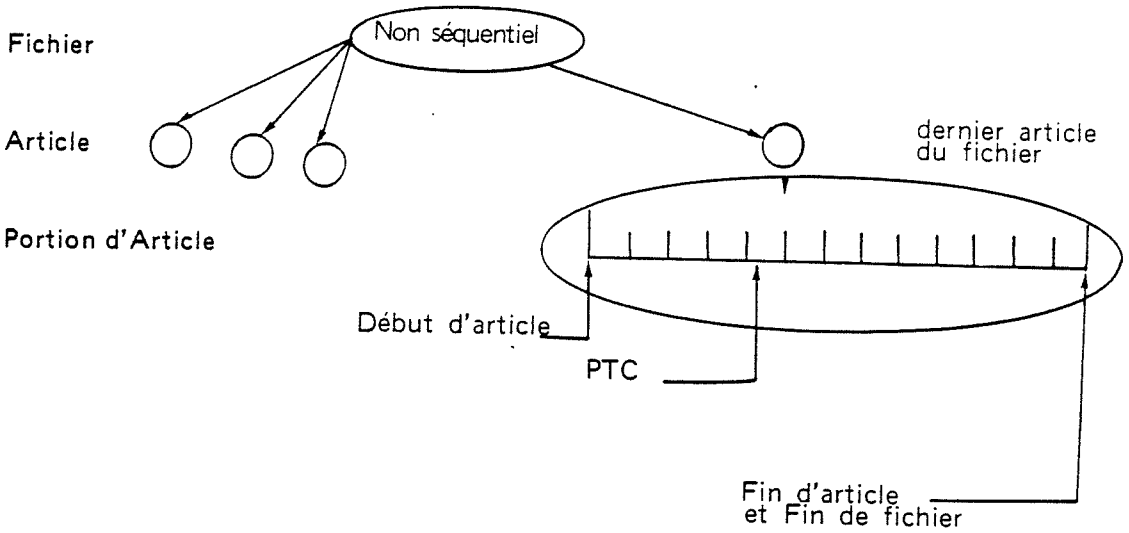


— Pour pouvoir sélectionner un tel article il doit être constitué d'un vecteur de mots **contigus** dans le fichier, et être de taille inférieure à 64 K mots. La méthode d'accès positionne le pointeur courant (PTC) de l'unité d'accès au fichier, **dans** l'article, et indique également dans le descripteur de l'unité d'accès (FAU) à l'aide de deux comptes de mots, la position du pointeur courant par rapport au début et à la fin de l'article.
— Toutes les requêtes d'accès séquentiel sur cette unité d'accès sont alors **bornées** par le début et la fin de l'article.
— L'ordre WRITE est un **WRITE Statique.** Il ne modifie pas l'allocation du fichier.
— L'ordre WRITE est une **réécriture.** Il laisse inchangé, les mots de l'article non concernés par le transfert de la zone d'échange.
— Un article existant peut donc être traité comme un fichier «séquentiel» **borné.**

d) **Portion d'Article Dynamique.**

Une méthode d'accès permet de créer un article à l'aide du Séquentiel. Plus précisément, elle permet de créer le **contenu** d'un article en mode séquentiel, avec allocation et désallocation dynamique de la place nécessaire à l'article.

Portion d'Article Dynamique.



— La création séquentielle dynamique du contenu d'un article n'est possible que pour le dernier article du fichier (physiquement). En effet ce mode de fonctionnement (P.A.D.) et le mode Séquentiel Pur Dynamique (SPD) utilisent le même outil d'allocation physique. Cet outil réalise l'**allocation dynamique** de la place au niveau **des fichiers** et non des articles. Il modifie le pointeur de Fin de Fichier, puisqu'il ne permet que de désallouer et allouer par rapport à la fin du fichier.

— A l'aide d'une méthode d'accès telle que l'Indexé par exemple, l'usager peut créer un nouvel article (IWRITE) au sens de créer son nom et le début de son contenu. La méthode d'accès réalise alors la sélection de l'article à créer, et commande au Séquentiel le fonctionnement selon le mode Portion d'Article Dynamique. L'usager peut alors créer le contenu de son article à l'aide de toutes les primitives du Séquentiel, dont le fonctionnement est identique à celui qui a été décrit pour le Séquentiel Pur Dynamique.

— L'ordre WRITE est un **WRITE Dynamique** ou encore une **écriture.**

— L'analogie avec le Séquentiel Pur Dynamique est limitée par une exception : la Taille d'un article écrit selon le mode Portion d'Article Dynamique doit être inférieure à 32 K mots.

— Ce dernier article du fichier, se comporte comme un fichier Séquentiel. Il simule le fonctionnement bande magnétique.

5.2.2 - Le Séquentiel Bufférisé

Le **Séquentiel Bufférisé** est une méthode d'accès capable de gérer. :

— avec bufférisation des unités d'accès à des articles appartenant à des fichiers : **Séquentiel** et **Indexé.** (BUF = 1)
— sans bufférisation la même chose que le Séquentiel (BUF = 0).

Le Séquentiel Bufférisé est réentrant, et réalise lorsque sa bufférisation est active, des échanges avec retour immédiat. Il fournit les 7 requêtes du séquentiel avec le même interface. (niveau SVC, FCB).

Le Séquentiel Bufférisé fonctionne avec bufférisation de façon autonome selon le mode Séquentiel Pur Statique pour un fichier Séquentiel.
Le Séquentiel Bufférisé fonctionne en accord avec l'Indexé Bufférisé selon les modes :

— Portion d'Article Statique en lecture seulement pour un article existant
— Portion d'Article Dynamique en lecture et écriture pour un article à créer.

Voir description détaillée au chapitre 3. Description Générale 3.3.1 la bufférisation

Remarque : Le fonctionnement du Séquentiel Bufférisé est totalement compatible avec celui du Séquentiel.
En particulier lorsque l'utilisateur commande de la Portion d'Article Statique en écriture (WRITE $\Longrightarrow$ réécriture) dans un article existant d'un fichier Indexé, le Séquentiel Bufférisé ne réalise pas de bufférisation mais un traitement identique à celui du Séquentiel.

5.3 - FONCTIONS LOGIQUES

5.3.1 - Le Niveau Fichier

a) **Généralités**

Les requêtes du niveau fichier pour les fichiers de Type Séquentiel numéro 0, sont semblables à celles des autres méthodes d'accès. Leur fonctionnement est décrit dans le chapitre 4. L'entité Fichier. Il n'est décrit ici que les particularités liées au Séquentiel Pur Dynamique. Elles ne concernent que les requêtes : CREAT, OPEN NEW.

La seule différence entre le séquentiel et le séquentiel bufférisé réside dans le fait que le Séquentiel Bufférisé est capable de prendre en compte un buffer de travail fourni par l'utilisateur de façon optionnelle lors des requêtes : CREAT, OPEN NEW, OPEN OLD.

L'interface d'appel d'une requête du niveau fichier est en PL 1600.

```
RA :=@FCB ;
SVC (FMS) ;        ou   FMS= '38
```

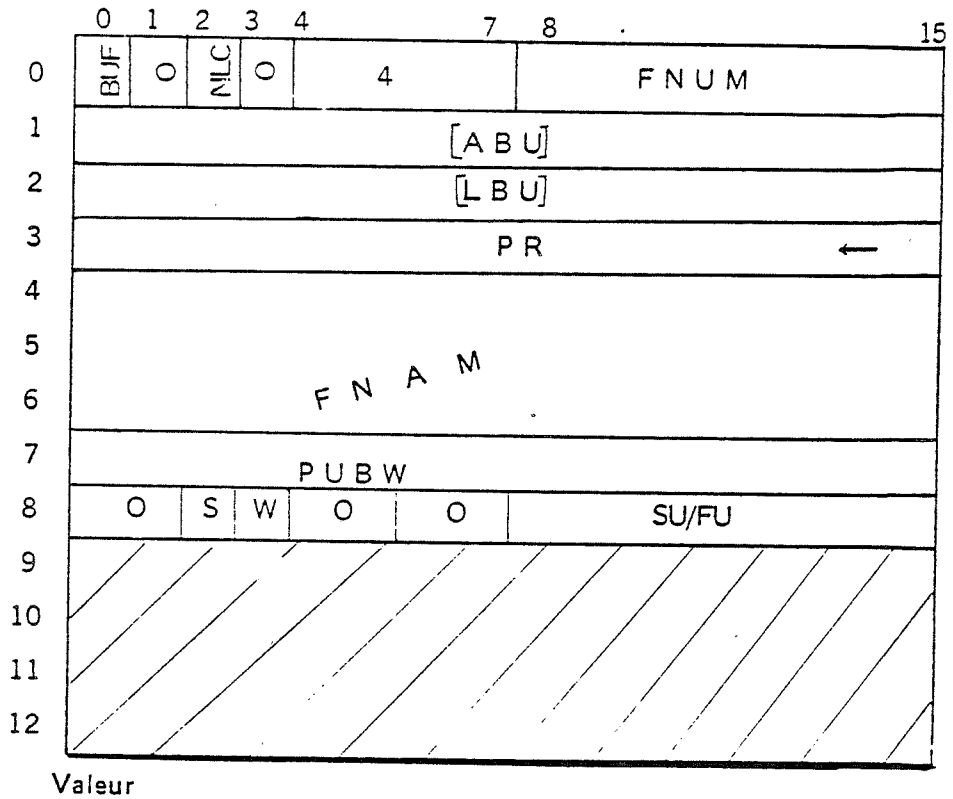
b) CREAT, OPEN NEW : **Création d'un fichier Séquentiel.**

Le seul paramètre spécifique de la création d'un fichier séquentiel est son numéro d'organisation logique : 0.
Les paramètres de création taille et nombre d'articles sont définis par FMS. Le fichier ne contient qu'un seul article, de taille nulle.
FMS alloue 1 granule pour créer physiquement un fichier séquentiel. Un fichier séquentiel est créé avec une organisation physique séquentielle.

Nom	CREAT	Séquentiel
But	Créer un fichier Permanent Séquentiel et créer une unité d'accès à ce fichier.	

Appel RA := @FCB ; SVC (FMS) ; <<FMS = `38

FCB



Comptes rendus

Valeur de PR	Signification
0	Primitive exécutée correctement.
`600B	FAU existante.
`600D	Fichier existant
`601F	Primitive en cours
`6020	Zone de Pavés saturée
`6021	FU saturée : la FU ne contient plus un seul granule de libre. La primitive est ineffective.
`6022	Table des Fichiers saturée.
`6028	Erreur de Syntaxe : @FCB, FONCT, [ABU, LBU,] FNAM, PUBW, FTYP)
`6029	Adresse de FCB incorrecte.
`602A	SU ou FU non gérée par FMS.
`602B	Méthode d'accès non gérée.

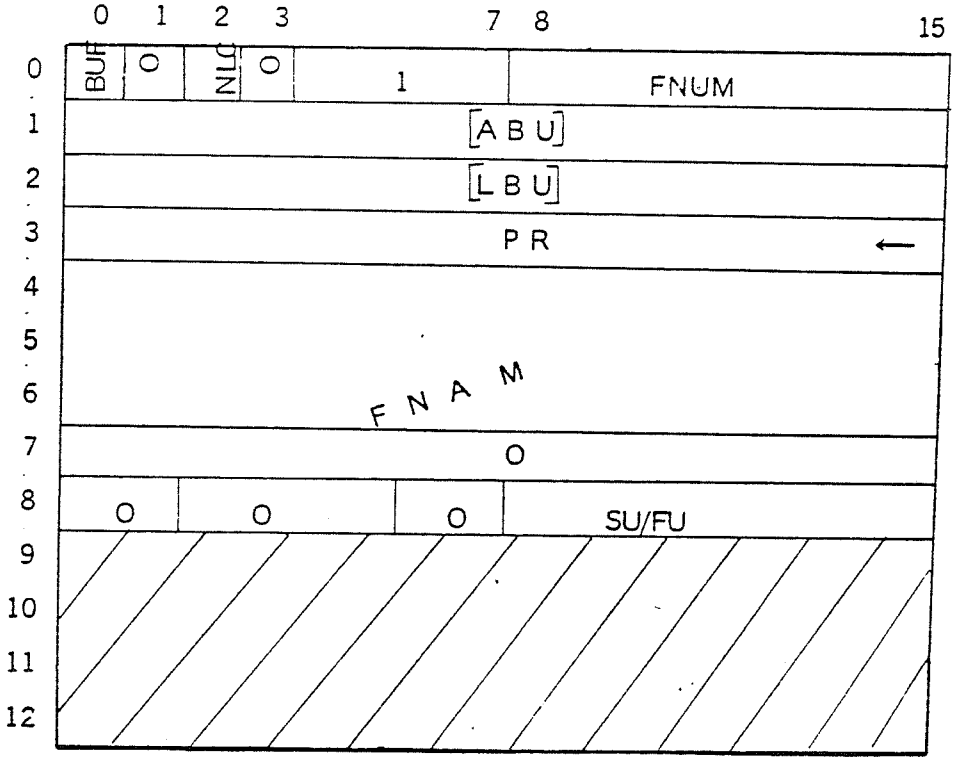
Erreurs graves

`6032, `6033, `6034	Informations Système Invalide.
`6035	FU verrouillée par IOCS.
`4.....	Erreur hardware : `4000 + mot d'état PU.

Nom	OPEN NEW	Séquentiel
But	Créer un fichier Temporaire Séquentiel et créer une Unité d'Accès à ce fichier.	

Appel : RA := @FCB ; SVC (FMS) ; ou FMS = `38

FCB



Comptes rendus	Valeur de PR	Signification
	0	Primitive correctement exécutée.
	`600B	FAU existante
	`600D	Fichier existant
	`601F	Primitive en cours
	`6020	Zone de Pavés saturée
	`6021	FU saturée : La FU ne contient plus un seul granule libre. La primitive est ineffective.
	`6028	Erreur de syntaxe : (@FCB, FONCT, [ABU, LBU,] FNAM, FTYP,).
	`6029	Adresse de FCB invalide
	`602A	SU ou FU non gérée par FMS.
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032 `6033 `6034	Informations Système Invalide.
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU.

5.3.2 - Le Niveau Portion d'Article

a) Généralités

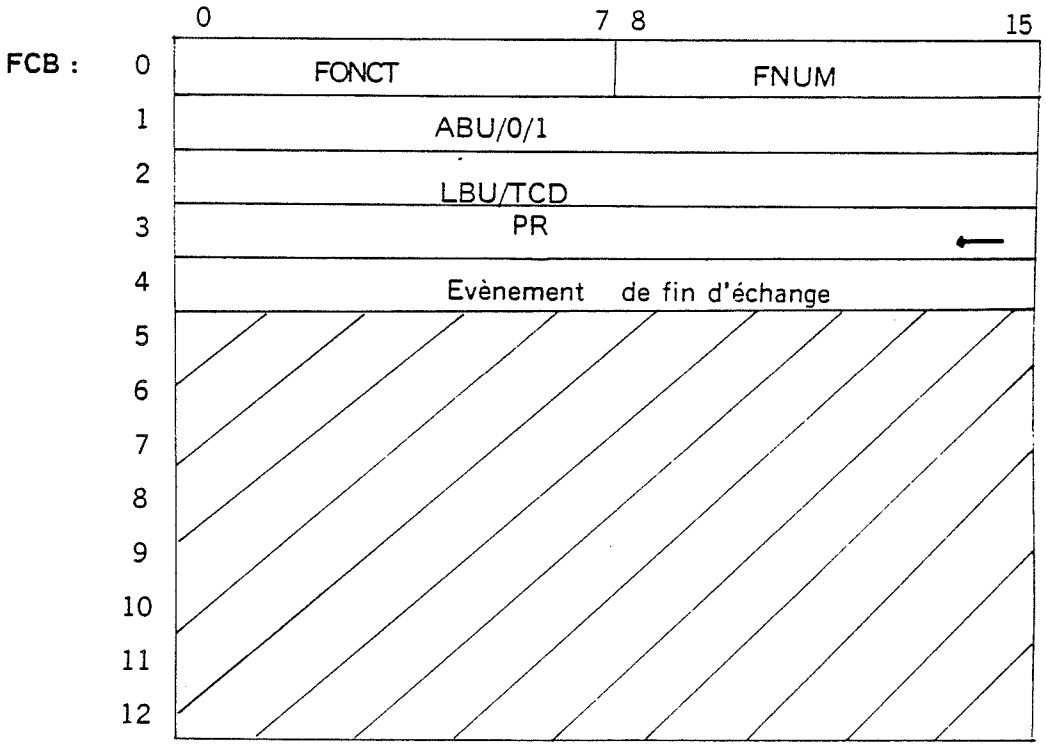
Le Séquentiel fournit 7 primitives d'accès séquentiel à un article d'un fichier de type quelconque et selon le mode défini par la méthode d'accès correspondant au type du fichier :
READ, WRITE, SKIPB, SKIPF, REWIND, SKEOA, WERE.

L'interface de programmation du Séquentiel Bufferisé est entièrement compatible avec celui du Séquentiel.
L'interface d'appel d'une requête du niveau portion d'article (du Séquentiel) est en PL 1600 :

```
RA := @FCB ;
SVC   (FMSS) ;          où FMSS = '39
```

Profondeur de Kstore nécessaire : 70 mots.

Description générale des FCB Séquentiel



Conventions

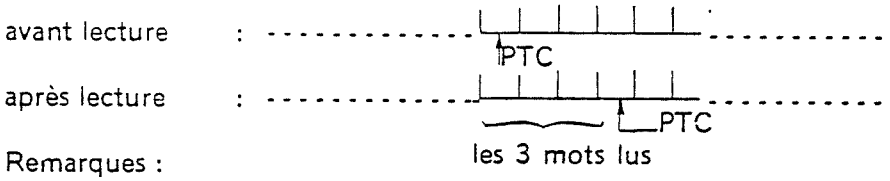
- Les hachures spécifient : zone non utilisée par FMS.
- ⟵ paramètres de retour fournit par FMS.
- ABU / 0 / 1 un paramètre au choix doit être fournit par l'utilisateur.
- L'évènement de fin d'échange peut être utilisé par l'usager à l'aide de la requête WEIO, par souci de compatibilité avec l'accès (séquentiel) offert par IOCS.
- Dans les autres cas le paramètre doit être fournit par l'utilisateur.

Toute requête du niveau portion d'article s'adresse à l'unité d'accès spécifiée par l'utilisateur (FNUM) et concerne le fichier accessible par celle-ci. Le choix d'une primitive parmi celles offertes par la méthode d'accès est fait par l'usager à l'aide de l'octet de fonction : FONCT.

b) READ : **lecture des n mots suivants**

Cette primitive permet de lire à partir du pointeur courant, le nombre d'octets précisé, ou lire jusqu'à la rencontre d'un des codes d'arrêt donnés (les codes d'arrêt sont situés dans une table organisée comme dans IOCS).

En fin de lecture, PTC pointe sur le mot suivant le dernier mot lu.
Exemple : READ 5 octets (c'est à dire 3 mots)



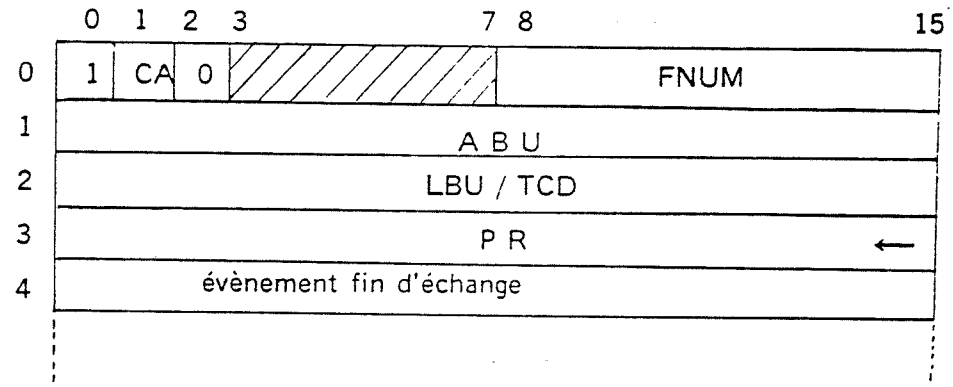
Remarques :

- le nombre d'octets effectivement lus est rendu à l'utilisateur (dans l'exemple ci-dessus : PR = 6 octets).
- on s'arrête en fin d'article, même si tous les octets demandés n'ont pas été lus.
- lors d'une lecture avec codes d'arrêt, on s'arrête après avoir lu un nombre maximum d'octets même si aucun code d'arrêt n'a été rencontré. Cette limite supérieure est celle d'IOCS (80 octets).

Nom	READ
But	Lire un nombre d'octets ou lire jusqu'à la rencontre d'un code d'arrêt.

Appel : RA := @FCB ; SVC (FMSS) ; où FMSS = '39

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.
CA : type de l'arrêt
CA = 0 indique une demande d'arrêt sur un compte d'octets. alors le mot 2 du FCB contient LBU, nombre d'octets à lire. 0 ⩽ LBU ⩽ '3FFE
CA = 1 indique une demande d'arrêt sur code. Alors le mot 2 du FCB contient TCD, adresse de la table contenant les codes d'arrêt.
ABU : adresse de la zone d'échange de l'utilisateur qui recevra les octets lus.
Évènement fin d'échange : (voir description paragraphe (a)).
PR : paramètre de retour.

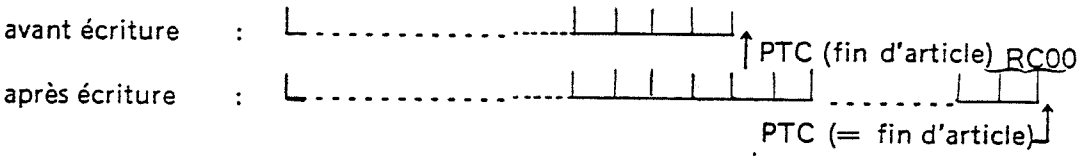
Comptes rendus	Valeur de PR	Signification READ
	0 ⩽ PR ⩽ '3FFE	Primitive correctement exécutée : PR = compte d'octets effectivement lus.
	`6001	Fin d'article : A l'exécution de cette primitive, le pointeur courant désignait déjà la fin de l'article. La primitive est ineffective.
	`600A	FAU inexistante.
	`601A	Erreur d'enchainement : Le fichier n'est pas de type séquentiel et aucun article n'a été sélectionné. La primitive est ineffective.
	`601F	Primitive en cours.
	`6028	Erreur de syntaxe. (∅ FCB, FONCT, LBU, TCD).
	`6029	Adresse de FCB invalide.
	`602B	Méthode d'accès non gérée.
Erreurs Graves	`6032 `6033 `6034	Informations Systèmes Invalides
	`6035	FU verrouillée par IOCS.
	`4.....	Erreur hardware : '4000 mot d'état PU.

c) WRITE : **Ecriture ou réécriture des n mots suivants.**
Cette primitive permet d'écrire, à partir du pointeur courant, le nombre d'octets précisé, ou écrire jusqu'à la rencontre d'un des codes d'arrêt donnés (Les codes d'arrêt sont situés dans une table organisée comme dans IOCS).

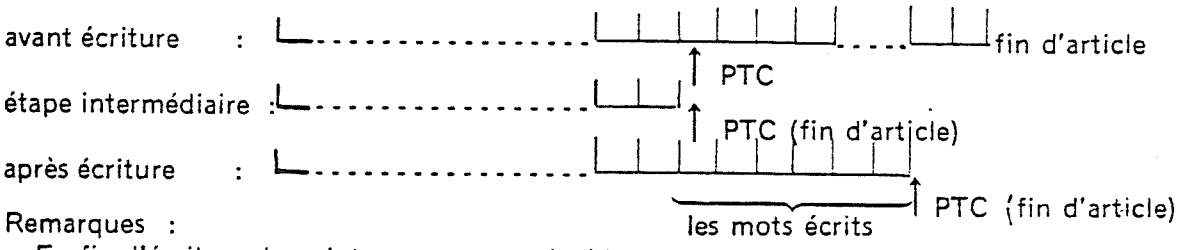
Cette primitive fonctionne selon 2 modes :
- WRITE Dynamique : écriture
- WRITE Statique : réécriture

WRITE Dynamique : écriture
Deux cas se présentent :

1) **PTC pointe sur la fin de l'article**, les octets précisés sont écrits en fin d'article.
Exemple : WRITE jusqu'au code «Retour chariot» (RC)



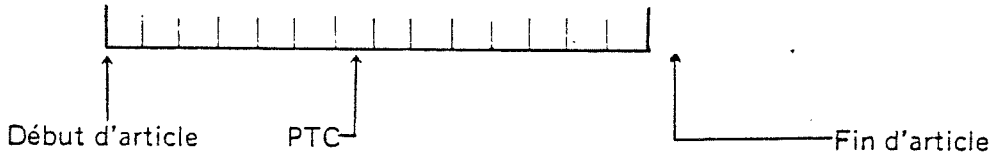
2) **PTC ne pointe pas sur la fin de l'article**
Il y a d'abord destruction de l'information comprise entre PTC et la fin de l'article puis les octets précisés sont écrits à partir du pointeur courant.
Exemple : WRITE 12 octets (c'est à dire 6 mots)



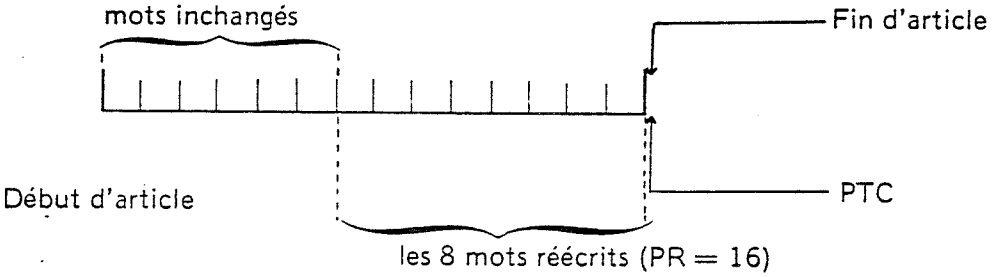
Remarques :
- En fin d'écriture, le pointeur courant coincide toujours avec la fin d'article.
- Cette primitive est donc éventuellement destructive. C'est la seule primitive qui diminue ou augmente la taille d'un fichier séquentiel ou d'un article créé en Portion d'Article Dynamique.
- Le nombre d'octets effectivement écrits est rendu à l'utilisateur. (exemple , dans le 1er cas cité précédemment : PR est impair)
- Lors d'une écriture avec code d'arrêt, on s'arrête après avoir écrit un nombre maximum d'octets, même si aucun code d'arrêt n'a été rencontré. Cette limite supérieure est celle d'IOCS (80 octets).

WRITE Statique : réécriture
Les octets précisés sont réécrits à partir du pointeur courant sans modifier les mots adjacents à la portion d'article statique réécrite.

Exemple : WRITE 39 octets (20 mots)
Avant WRITE :



Après WRITE :

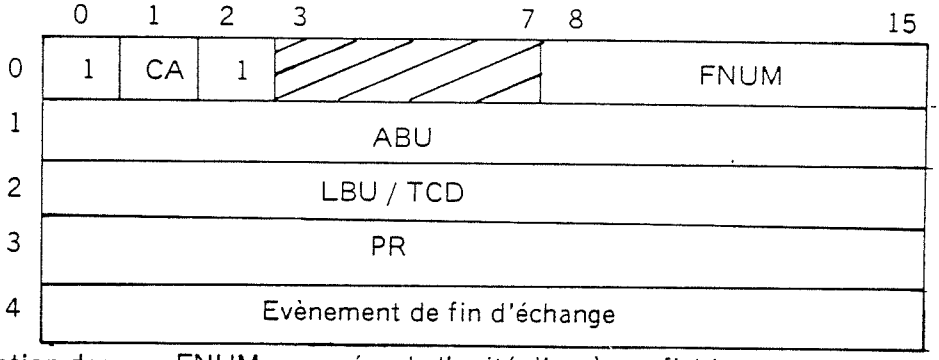


Remarques :
— le nombre d'octets effectivement réécrits est rendu à l'utilisateur. Il est limité par la fin de l'article, il se peut donc que tous les octets demandés ne soient pas réécrits.
— Lors d'une réécriture avec codes d'arrêt, la taille de la portion d'article est limitée à 80 octets (IOCS).

Nom	WRITE
But	Ecrire un nombre d'octets ou écrire jusqu'à la rencontre d'un code d'arrêt.

Appel RA := FCB ; SVC(FMSS) ; où FMSS = '39

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier
CA : type de l'arrêt
CA = 0, indique une demande d'arrêt sur compte d'octets. Alors le mot 2 du FCB contient LBU, nombre d'octets à écrire. 0 ⩽ LBU ⩽ '3FFE
CA = 1, indique une demande d'arrêt sur code. Alors le mot 2 du FCB contient TCD, adresse de la table contenant les codes d'arrêt.
ABU : adresse de la zone d'échange de l'utilisateur qui contient les octets à écrire, ou réécrire.
Evènement fin d'échange : (voir description paragraphe (a)).
PR : paramètre de retour.

Comptes-rendus

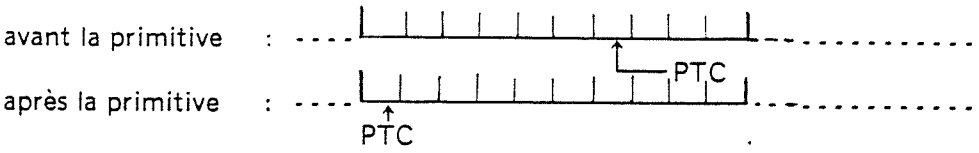
Valeur de PR	Signification
0 ⩽ PR ⩽ 3FFE	**Primitive correctement exécutée :** PR = le compte d'octets effectivement écrits ou réécrits. Pour un WRITE dynamique, si 0 < PR < LBU alors la saturation de la FU est atteinte mais le fichier est utilisable. La prochaine requête WRITE fournira le PR = '6017.
'6001	**Fin d'article :** (WRITE statique) A l'exécution de cette primitive, le pointeur courant désignait déjà la fin de l'article. La primitive est ineffective.
'600A	**FAU inexistante.**
'6014	**Protection écriture**
'6017	**Fichier trop long.** (WRITE Dynamique) La taille de la portion d'article dynamique doit être inférieure 0 32 Kmots - 1 ou la taille du fichier est trop grande (Indexé 64 Kmots). La création de l'article est anormalement terminée. Ou encore la FU SUPPORT est saturée. La requête est ineffective. La prochaine requête WRITE fournira le compte-rendu : 6021.

	`601A	**Erreur d'enchaînement** : Le fichier n'est pas de type séquentie et aucun article n'a été sélectionné. Ou encore le fichier est exploité en écriture dynamique et le fichier n'est pas en mono-accès. Ceci est probablement dû à une mauvaise utilisation des requêtes ALTER, WERE. La primitive est ineffective.
	`601F	**Primitive en cours**
	`6021	**FU saturée** (WRITE Dynamique), il n'y a plus de place sur le support le fichier (permanent) est utilisable dans la mesure ou le pointeur de fin de fichier a été mis à jour sur disque (CLOSE ou ALTER).
	`6028	**Erreur de syntaxe** : (∂FCB, FONCT, ABU, LBU, TCD)
	`6029	**Adresse de FCB invalide**
	`602B	**Méthode d'accès non gérée**
Erreurs Graves	`6032 `6033 `6034	**Informations systèmes invalides**
	`6035	**FU verrouillée par IOCS**
	`4.....	**Erreur hardware** : `4000 + mot d'état PU.

d) SKIPB : **Saut arrière de n mots**

Saut arrière du nombre d'octets précisé.

exemple : SKIPB 12 octets (c'est à dire 6 mots)



Remarques :

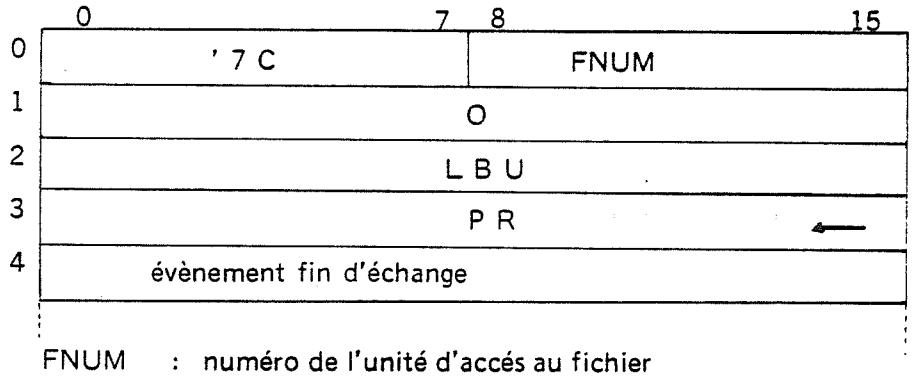
. Le nombre d'octets effectivement sautés est rendu à l'utilisateur.

. On s'arrête sur le début d'article même si tous les octets demandés n'ont pas été sautés.

Nom	**SKIPB**
But	Saut arrière du nombre d'octets précisés.

Appel RA :=@FCB ; SVC (FMSS) ; où FMSS = '39

FCB



Description des paramètres

FNUM : numéro de l'unité d'accés au fichier
LBU : nombre d'octets à sauter
Évènement fin d'échange : (voir description paragraphe (a)).
PR : paramètre de retour

Comptes rendus

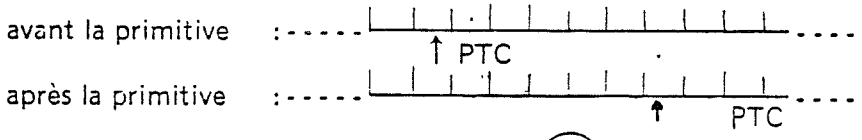
Valeur de PR	Signification
0≤ PR≤'3FFE	**Primitive correctement exécutée** : PR = compte d'octets effectivement sautés.
'6002	**Début d'article** : A l'exécution de cette primitive, le pointeur courant désignait déjà le début de l'article. La primitive est ineffective.
'600A	**FAU inexistante.**
'601A	**Erreur d'enchaînement** : Le fichier n'est pas de type séquentiel et aucun article n'a été sélectionné. La primitive est ineffective.
'601F	**Primitive en cours.**
'6028	**Erreur de syntaxe** : (@FCB, FONCT, LBU)
'6029	**Adresse de FCB invalide.**
'602B	**Méthode d'accés non gérée.**
Erreurs graves '6032, '6034	**Informations Systèmes Invalides**
'6035	**FU verrouillée par IOCS**
'4 . . .	**Erreur hardware** : '4000 + mot d'état PU.

e) SKIPF : Saut avant de n mots

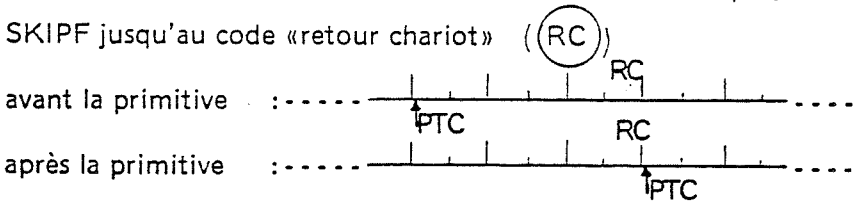
Saut avant du nombre d'octets précisé, ou saut en avant jusqu'à la rencontre d'un des codes d'arrêt donnés. (Les codes d'arrêts sont situés dans une table organisée comme dans IOCS).

Exemple :

1) SKIPF 12 octets (c'est à dire 6 mots)



2) SKIPF jusqu'au code «retour chariot» (RC)



en fin de saut PTC pointe sur le mot qui suit celui contenant le code d'arrêt.

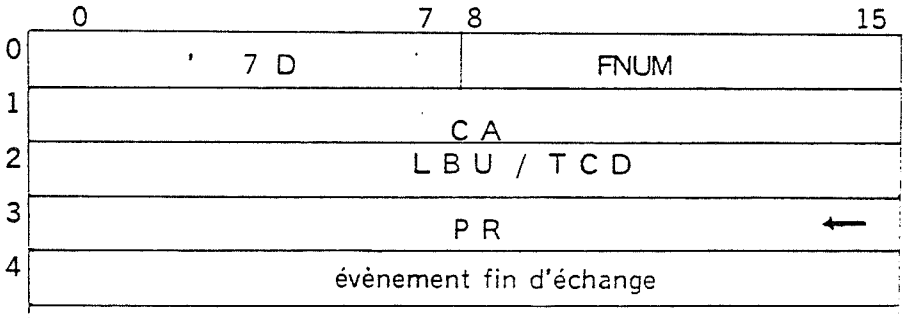
Remarques :

- Le nombre d'octets effectivement sautés est rendu à l'utilisateur.
- On s'arrête sur la fin d'article, même si tous les octets demandés n'ont pas été sautés.
- Lors d'un saut avec codes d'arrêt, on s'arrête après avoir sauté un nombre maximum d'octets, même si aucun code d'arrêt n'a été rencontré. Cette limite supérieure est celle d'IOCS (80 octets).

Nom	SKIPF
But	Saut avant d'un nombre d'octets ou sauter en avant jusqu'à la rencontre d'un code d'arrêt.

Appel : RA : = @FCB ; SVC (FMSS) ; où FMSS = '39

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.

CA : type de l'arrêt

CA = 0 indique une demande d'arrêt sur un compte d'octets. Alors le mot 2 du FCB contient LBU nombre d'octets à sauter.
0 ⩽ LBU ⩽ '3FFE

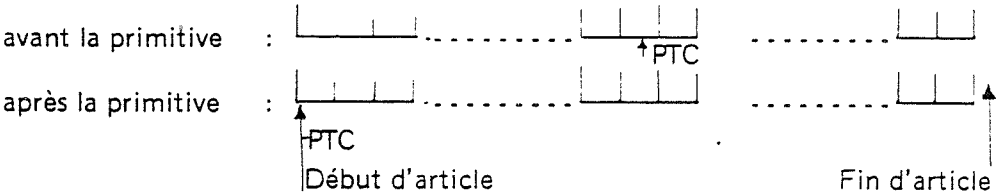
CA = 1 indique une demande d'arrêt sur code. Alors le mot 2 du FCB contient TCD, adresse de la table contenant les codes d'arrêt.

Evènement fin d'échange (voir description paragraphe (a)).

PR : paramètre de retour.

Comptes rendus	Valeur de PR	Signification	SKIPF
	0 ⩽ PR ⩽ `3FFE	**Primitive correctement exécutée** : PR = le compte d'octets effectivement sautés.	
	`6001	**Fin d'article** : A l'exécution de cette primitive, le pointeur courant désignait déjà la fin de l'article. La primitive est ineffective.	
	`600A	**FAU inexistante.**	
	`601A	**Erreur d'enchaînement** : Le fichier n'est pas de type séquentiel et aucun article n'a été sélectionné. La primitive est ineffective.	
	`601F	**Primitive en cours.**	
	`6028	**Erreur de Syntaxe,** (@FCB , FONCT , ABU , LBU , TCD)	
	`6029	**Adresse de FCB invalide.**	
	`602B	**Méthode d'accès non gérée.**	
Erreurs Graves	`6032 `6034	**Informations Systèmes Invalides**	
	`6034		
	`6035	**FU verrouillée par IOCS.**	
	`4.....	**Erreur hardware** : `4000 + mot d'état PU.	

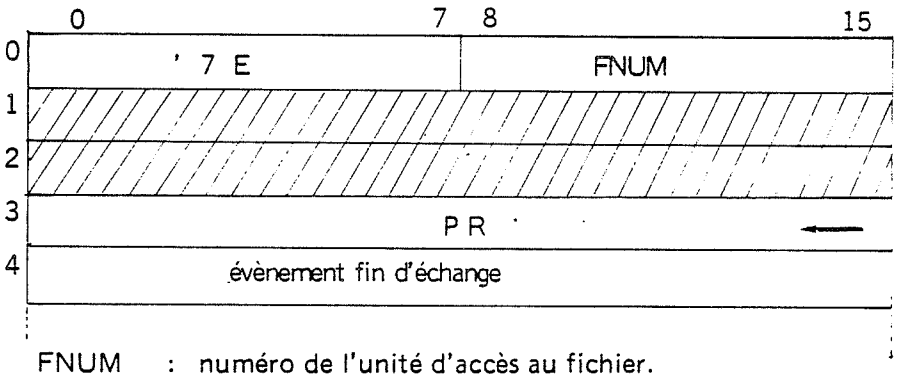
f) REWIND : Mettre le pointeur au début de l'article.
En fin de primitive, PTC pointe sur le premier mot du fichier.
Exemple : REWIND



Nom	**REWIND**
But	Se positionner au début de l'article.

Appel RA : =@FCB ; SVC (FMSS) ; où FMSS = '39

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.
Évènement fin d'échange : (voir description pragraphe (a)).
PR : paramètre de retour.

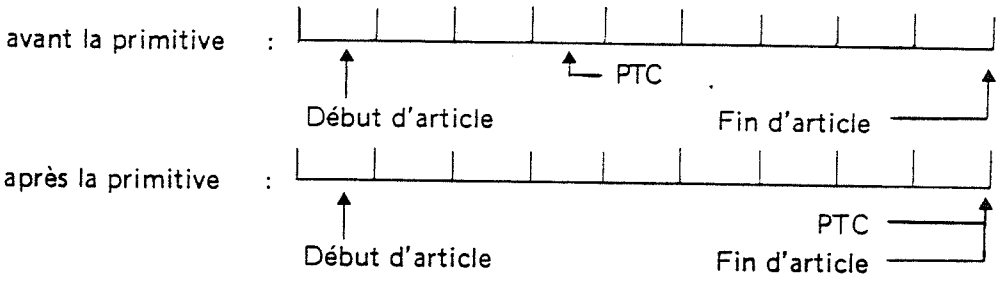
Comptes rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée.**
'600A	**FAU inexistante.**
'601A	**Erreur d'enchaînement** : Le fichier n'est pas de type séquentiel et aucun article n'a été sélectionné. La primitive est ineffective.
'601F	**Primitive en cours.**
'6028	**Erreur de syntaxe,** (@FCB, FONCT)
'6029	**Adresse de FCB invalide.**
'602B	**Méthode d'accès non gérée.**

Erreurs Graves

'6032, '6034	**Informations Systèmes Invalides**
'6035	**FU verrouillée par IOCS**
'4.....	**Erreur hardware** : '4000 + mot d'état PU.

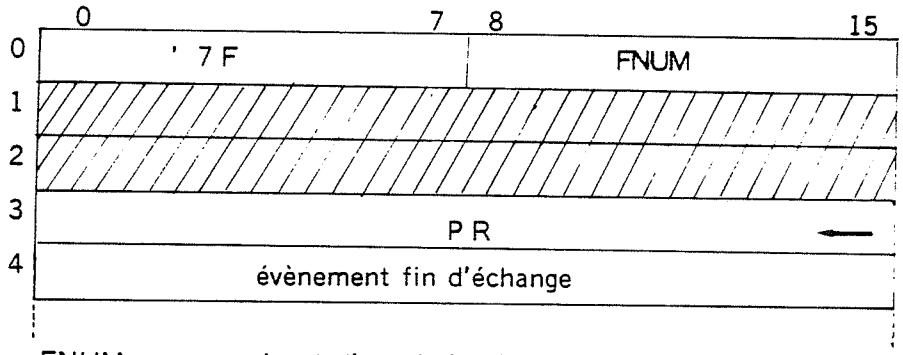
g) SKEOA : **Mettre le pointeur à la fin de l'article.**
En fin de primitive, le pointeur courant (PTC) coincide avec la fin de l'article.
Exemple : SKEOA



Nom	**SKEOA**
But	Sauter à la fin de l'article.

Appel RA : =@FCB ; SVC (FMSS) ; où FMSS = '39

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.
Évènement fin d'échange : (voir description paragraphe (a)).
PR : paramètre de retour.

Comptes rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée.**
'600A	FAU inexistante.
'601A	Erreur d'enchaînement : Le fichier n'est pas de type séquentiel et aucun article n'a été sélectionné. La primitive est ineffective.
'601F	Primitive en cours.
'6028	Erreur de syntaxe, (@FCB, FONCT)
'6029	Adresse de FCB invalide
'602B	Méthode d'accès non gérée.

Erreurs Graves

'6032 '6034	Informations Systèmes Invalides.
'6035	FU verrouillée par IOCS.
'4.....	Erreur hardware : '4000 + mot d'état PU.

h) WERE : Commander un WRITE statique ou dynamique

La requête WERE, programmée avant une suite de requêtes WRITE, à pour effet de commander leur mode de fonctionnement :
— WRITE statique c'est à dire réécriture dans le fichier sans modifier le reste.
— WRITE dynamique c'est à dire écrire avec desallocation et allocation dynamique du fichier.
Les deux modes de fonctionnement de la requête WRITE sont expliqués au début de ce chapitre.
L'utilisation de la requête WERE permet en particulier de réécrire dans un fichier séquentiel ou d'agrandir un fichier statique (ex : direct). Elle ne doit pas être utilisée pour les fichiers dynamiques, sauf exception : pendant la phase de création du dernier article.

Nom	**WERE**
But	Commander le mode de fonctionnement de WRITE.

Appel RA = @FCB ; SVC(FMSS); où FMSS = '39

FCB

	0 — 7	8 — 15
0	'7B	FNUM
1	WERE	
2	0	
3	PR ←	
	Mot réservé	

Description des paramètres
FNUM : numéro de l'unité d'accès au fichier.
WERE : WERE = 0 ⟹ WRITE Dynamique
WERE : WERE = 1 ⟹ WRITE statique
PR : paramètre de retour

Comptes rendus	**Valeur de PR**	**Signification**
	0	Primitive correctement exécutée
	'600A	FAU inexistante
	'6014	Protection écriture
	'601A	Erreur d'enchainement
	'601F	FAU en cours
	'6028	Erreur de syntaxe : FCB, FONCT
	'6029	Adresse de FCB invalide
	'602B	Méthode d'accès non gérée

i) La portion d'article et les comptes-rendus

Le tableau suivant indique les exceptions du fonctionnement des requêtes du module séquentiel lorsqu'il travaille selon les deux modes Portion d'Article Statique ou Dynamique.

OUI : La requête est correctement exécutée. La portion d'article est toujours sélectionnée et peut continuer normalement. Les informations systèmes sont cohérentes.
NON : La requête est totalement ineffective. La portion d'article peut continuer.
INV : Le fonctionnement de la requête est anormal. La portion d'article n'est plus sélectionnée ; il faut sélectionner à nouveau. Dans le cas Portion d'Article Dynamique le fichier doit être rendu cohérent à l'aide de FUP.

Le tableau suivant indique également la liste des comptes rendus de l'ensemble des requêtes du module séquentiel.

Séquentielle Valeur de PR	Signification	Portion d'Article utilisée
0⩽ PR⩽ '3FFE	Primitive correctement exécutée	OUI
'6001	Fin d'article	OUI
'6002	Début d'article	OUI
'600A	FAU inexistante	NON
'6014	Protection écriture	NON
'6017	Fichier trop long	INV
'601A	Erreur d'enchaînement	NON
'601F	Primitive en cours	NON
'6021	FU saturée	INV
'6028	Erreur de syntaxe @ FCB	NON
	FONCT	NON
	ABU	NON
	LBU	NON
	TCD	NON
'6029	Adresse de FCB invalide	NON
'602B	Méthode d'accès non gérée	NON
'6032	Informations Système Invalides	INV
'6033	Informations Système Invalides	INV
'6034	Informations Système Invalides	INV
'6035	FU verrouillée par IOCS	INV
'4.....	Erreur hardware	INV

6 — LE FICHIER DE TYPE INDEXE

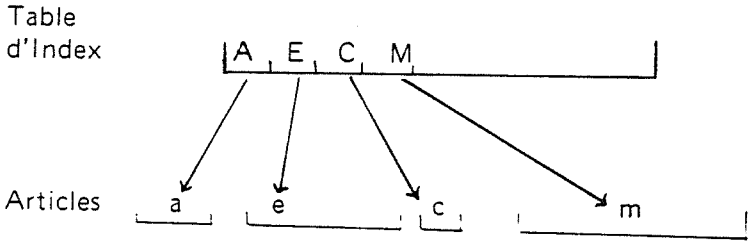
6 - LE FICHIER de TYPE INDEXE

6.1 - ORGANISATION LOGIQUE

6.1.1. - Présentation.

L'organisation logique d'un fichier de Type **Indexé** est définie à la création du fichier, elle possède le **numéro 1**.

Organisation logique : **Indexé.**



On distingue 2 parties dans un fichier indexé.
- **La table d'index** : contenant des informations système gérées par FMS.
- **Les articles** du Fichier : contenant l'information de l'utilisateur.

a) **Les articles** :
Un fichier indexé est composé d'un ou plusieurs **articles** identifiés par un **nom.** Le **nom** et la **taille** d'un article sont définis par l'utilisateur à la **création** de l'**article.**
Un nom d'article est composé de 8 caractères ASCII (2 caractères par mot). FMS accepte une plage de 40 caractères : les lettres [A ; Z], les chiffres [0 ; 9], :, ;, /, Nul (en fin seulement). FMS vérifie la non homonymie des noms d'articles. La taille des articles est **variable** d'un article à l'autre. Elle doit cependant être un nombre **pair** d'octets, inférieure à 64 K mots.

b) **La table d'index** :
La table d'index contient les noms des articles ainsi que l'adresse de chacun dans le fichier. C'est à la création du fichier que l'**utilisateur** définit le nombre maximum d'articles (NART) que le fichier pourra contenir. La taille de la table d'index est figée à la création du fichier par FMS en fonction des règles suivantes :

Règles :
- $1 \leqslant NART \leqslant 8159$
- Pour NART articles il faut : 4 (NART + 1) mots de table d'index
- Une table d'index occupe un nombre entier de secteurs.
- Une table d'index de n secteurs permet de gérer dans le fichier : (n x 32) - 1 article.

6.1.2 - Structure physique d'un fichier Indexé

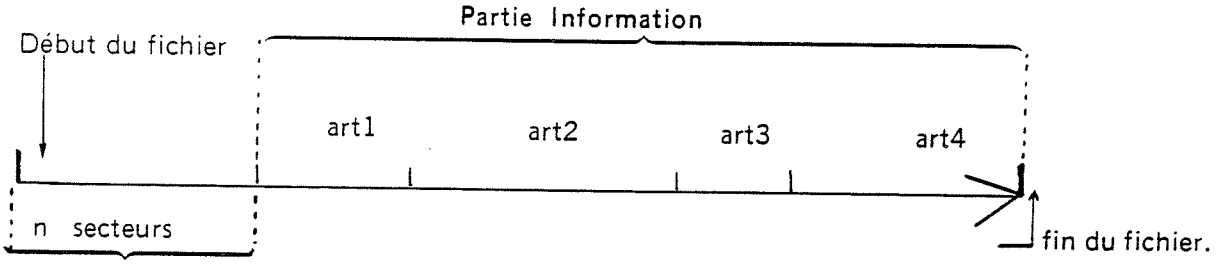


Table d'Index (TIX) :

0 mot 1	45 mot 2	15 mot 3	mot 4
ART1	Ad1	ART1	Ad 1
ART2	Ad2	ART2	Ad 2
ART3	Ad3	ART3	Ad 3
ART4	Ad4	ART4	Ad 4
0	Ad5	0	Ad 5
0	0	0	0
0	0	0	0
0	0	0	0

sens de recherche séquentielle dans la table d'index.

Le schéma précédent indique comment l'organisation logique Indexé est implantée physiquement dans le fichier.

a) **Table d'Index** :

La table d'index occupe les n premiers secteurs du fichier.
* 1 secteur pour 31 articles.
* 2 secteurs pour 63 articles, etc. . .

Le premier article est donc aligné à une frontière de secteur.

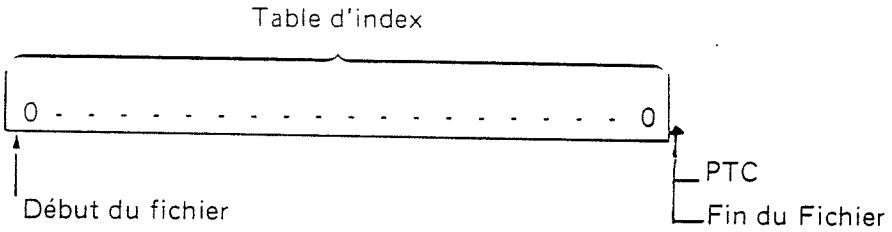
Dans la table d'index un nom d'article occupe 3 mots, très exactement 43 bits. Il est codé dans un système à base 40 (Radix 40). Une adresse occupe 1 mot et 5 bits. Elle représente l'adresse relative de l'article dans le fichier par rapport au début de la **partie information** : les bits 0-4 du mot 3 sont les poids forts de l'adresse relative

Ad 1 = 0
Ad 2 = longueur de ART1

Règles :
- $Ad\ n = \sum_{i=1}^{n-1}$ long ART i
- la taille de la partie information est limitée à **2 M mots.**
- On appelle élément de table d'index les 4 mots contenant le nom d'un article et son adresse dans le fichier.

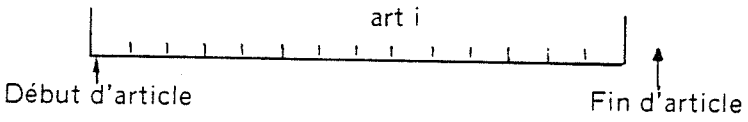
b) **La partie information :**
A sa création un fichier indexé ne contient aucun article, et la table d'index est mise à zéro par FMS.

Fichier indexé après création.

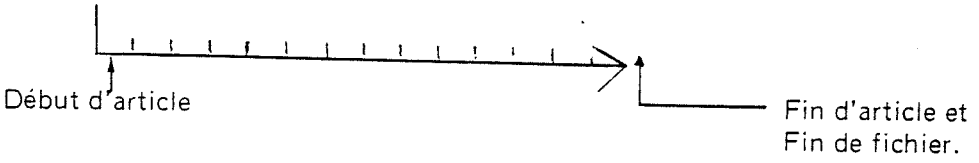


A la création de chaque article, la table d'index est mise à jour, et le fichier est agrandit dynamiquement à partir du pointeur de fin de fichier, du contenu de l'article.
Règles :
- Au nième article du fichier est associé de façon bijective le nième élément de table d'index.
- Les articles ne sont pas triés selon leur nom par exemple. Leur position dans le fichier correspond à la chronologie des traitements entrepris sur le fichier.

c) **Remarques générales :**
D'après le précédent schéma décrivant la structure physique d'un fichier Indexé on peut remarquer que :
1) Le fichier est composé d'articles indépendants, c'est à dire que chaque article est un vecteur borné de mots.
On peut considérer la table d'index comme le premier article du fichier (pseudo-article ou article système).

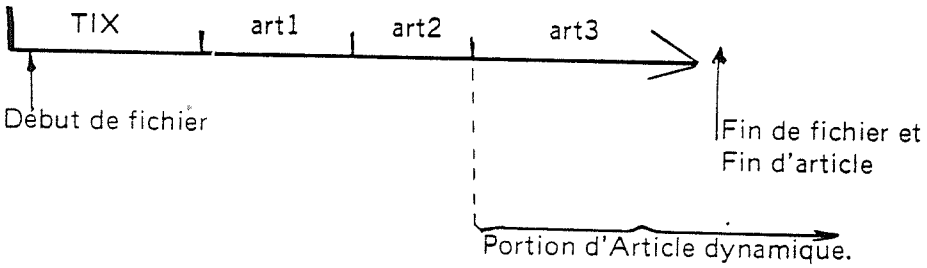


Le dernier article, pendant sa création est un vecteur dynamique de mots. Son contenu pourra être créé selon le mode Portion d'article dynamique.

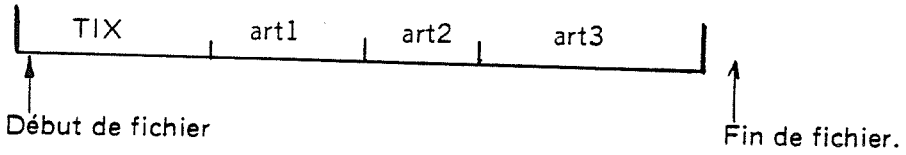


2) Le fichier lui-même est un vecteur de mots. C'est un vecteur dynamique au sens ou l'on peut l'agrandir par la création d'un nouvel article. C'est un vecteur borné de mots au sens ou, en dehors d'une phase de création d'article, un accès séquentiel au fichier selon le mode séquentiel Pur Statique est borné par le début et la fin du fichier.

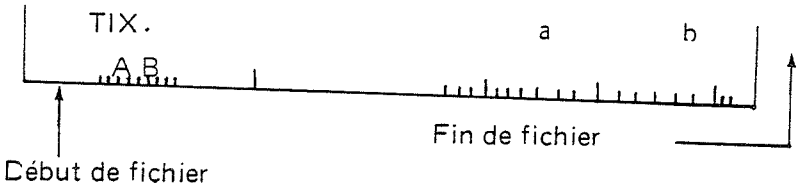
Un fichier Indexé, création d'un article :



Un fichier Indexé figé :



3) Deux articles adjacents, c'est à dire, dont les noms sont adjacents dans la table d'index sont physiquement adjacents (aux changements de granule près).



6.2 - LES METHODE D'ACCES

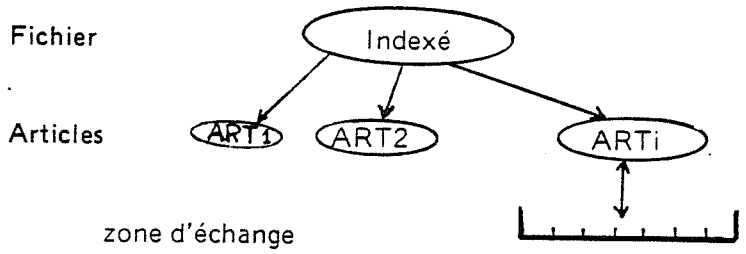
6.2.1 - L'Indexé

L'Indexé est une méthode d'accès capable de gérer des Unités d'Accès à des fichiers indexés **(Numéro 1).**

L'Indexé **fournit 6 requêtes** d'accès direct à l'article : IREAD, IWRITE, ISUP, IRWRITE, IRTIX. L'Indexé **permet** d'utiliser au niveau portion d'article les 6 requêtes de la méthode d'accès : Séquentiel.

L'Indexé permet pour 1 unité d'accès à un fichier Indexé 3 modes de fonctionnement.

a) **Accès direct aux articles**



L'utilisateur peut **lire** (IREAD) un article en le désignant à l'aide de son nom.
L'utilisateur peut **réécrire** (IRWRITE) le contenu d'un article en le désignant par son nom.
L'utilisateur peut **renommer** (IRNAM) l'article en cours. Par exemple l'article qui vient d'être lu.
L'utilisateur peut **supprimer** (ISUP) un article en le désignant par son nom. Cette primitive ne tasse ni la table d'index ni les articles du fichier, elle crée donc un «trou», qui pourra être éventuellement récupéré par la primitive de création d'un article IWRITE si le Trou est à la fin du fichier.
L'utilisateur peut **créer** (IWRITE) un article et définir sa taille et son nom. FMS vérifie qu'il n'existe pas d'article de même nom. L'article est créé à la fin du fichier. La primitive IWRITE permet de **récupérer** la place d'une suite d'articles supprimés en fin de fichier.
L'utilisateur peut demander à FMS (IRTIX) de **charger** en mémoire centrale **toute** la table d'index à fin d'accélérer l'accès direct aux articles, c'est à dire la lecture et/ou la mise à jour de la Table d'index.

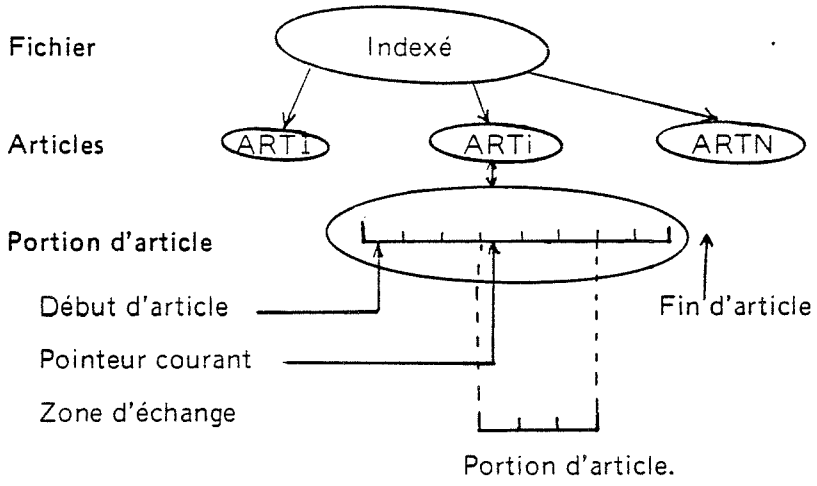
b) **Accès à une portion d'article.**

L'indexé permet un accès séquentiel :

— à un article existant selon le mode Portion d'Article Statique (P.A.S.).

— à un article à créer selon le mode Portion d'Article Dynamique (P.A.D.).

1) **Portion d'article statique.**



Portion d'article.

L'utilisateur

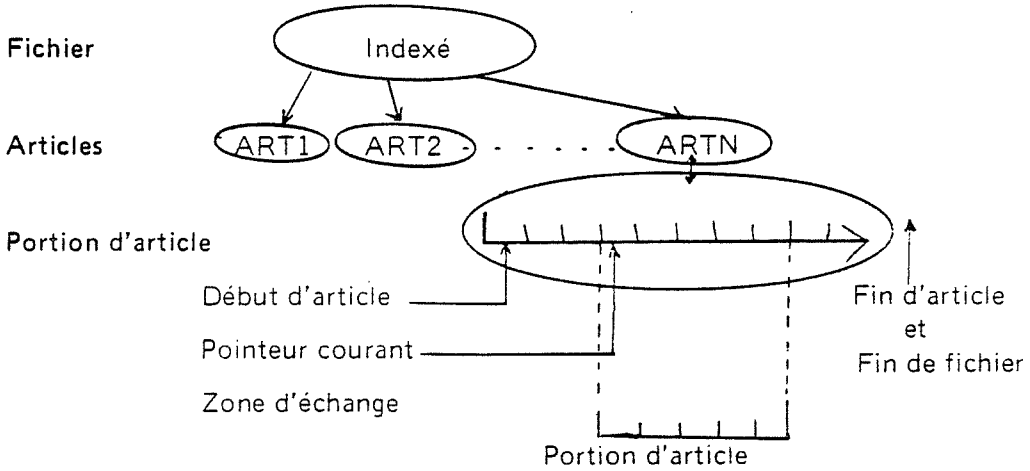
1 - sélectionne un article existant à l'aide d'une requête IREAD, ou IRWRITE.

2 - accède à l'article sélectionné à l'aide de toutes les requêtes de la méthode d'accès : Séquentiel, selon le mode P.A.S. (Voir description au chapitre 5.2)

Après la sélection, le pointeur courant est positionné par rapport au début de l'article, à une distance égale à la longueur de l'échange demandé par la sélection. Si cette longueur est nulle le pointeur courant adresse le premier mot de l'article. Pour accéder à un autre article, il faut sélectionner à nouveau.

Un fichier Indexé peut être considéré comme N «fichiers séquentiels» bornés et indépendants.

2) **Portion d'article dynamique.**



Portion d'article

L'utilisateur

1 - crée un nouvel article et réalise une sélection dynamique à l'aide de la primitive IWRITE. La requête IWRITE réalise une sélection selon le mode Portion d'Article Dynamique **seulement** pour le dernier article du fichier. Il faut donc prendre garde de ne pas «tomber dans un trou». Ce qui est effectif s'il existe un trou de même longueur que la zône d'échange associée à IWRITE.

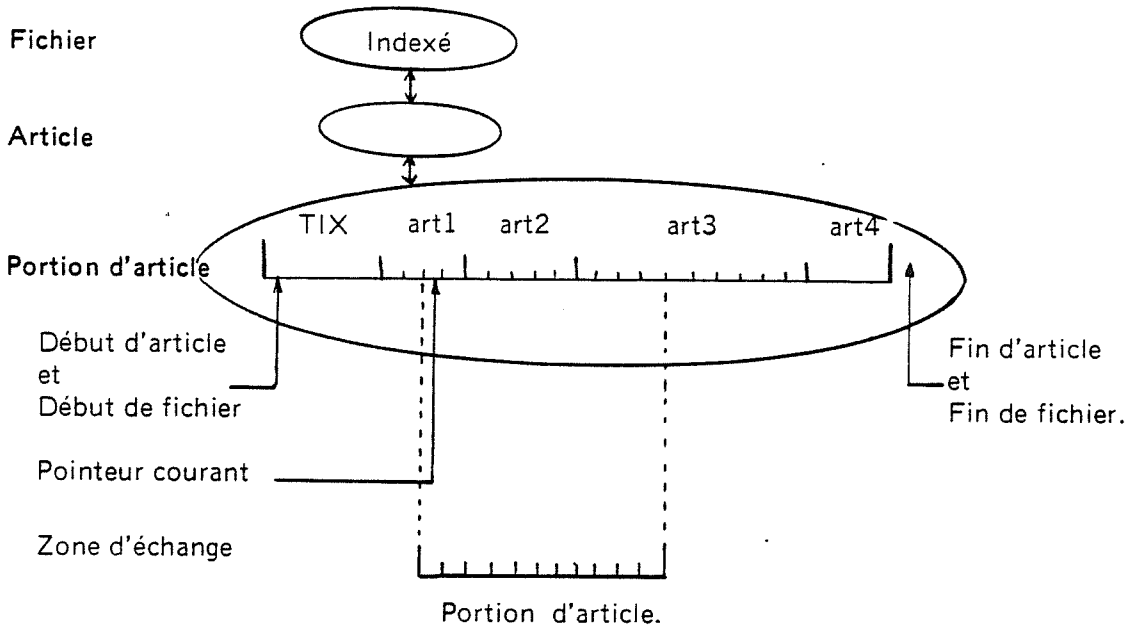
2 - accède à l'article sélectionné pour **créer son contenu** à l'aide de toutes les requêtes de la méthode d'accès. Séquentiel, selon le mode P.A.D. (Voir description chapitre 5.2).

La phase de création de l'article est terminée par toute requête du niveau article sur l'unité d'accès concernée, ou par une requête du niveau fichier de destruction de l'unité d'accès (CLOSE, DELET).

c) **Accès séquentiel pur statique.**

L'indexé permet un accès séquentiel sur tout le fichier.

Accès Séquentiel Pur Statique.



Après l'une des trois primitives de création d'une Unité d'Accès (CREAT, OPEN NEW, OPEN OLD) et **avant** toute requête du niveau article (ex : IREAD) l'indexé permet un accès Séquentiel Pur Statique. L'usager peut utiliser les requêtes de la méthode d'accès : Séquentiel. Les frontières d'articles sont alors ignorées par le séquentiel.
Après une création de fichier (CREAT, OPEN NEW), le pointeur courant adresse la fin du fichier c'est à dire la fin du pseudo-article table d'index. Après l'ouverture d'un fichier permanent indexé (OPEN OLD), le pointeur courant adresse le début du fichier.
Cet accès séquentiel permet d'archiver où restituer le contenu d'un fichier Indexé.

6.2.2 - L'Indexé Bufférisé

L'Indexé Bufférisé est une méthode d'accès capable de gérer avec bufférisation ou sans des Unités d'Accès à des fichiers Indexés (numéro 1).

Si l'utilisateur ne demande pas le fonctionnement de la bufférisation (BUF = 0), l'Indexe Bufférisé fonctionne comme l'Indexé.

Lorsque l'utilisateur demande le fonctionnement de la bufférisation (BUF = 1) pour l'accès séquentiel aux portions d'articles, l'Indexé Bufférisé réalise les traitements suivants, transparents à l'utilisateur :

- Initialisation de la gestion du buffer de travail pour CREAT, OPEN NEW, OPEN OLD et pour les sélections d'article réalisées par :
  - IREAD pour de la Portion d'Article Statique ou Dynamique
  - IWRITE pour de la Portion d'Article Dynamique
- Eventuellement écriture sur disque du buffer de travail, (bufférisation en écriture) lorsque l'utilisateur termine une Portion d'Article à l'aide de : CLOSE, DELET et de toute requête du niveau article de l'Indexé. Voir description détaillée au chapitre 3 ; description générale § 3.3.1, la bufférisation.

6.3 - FONCTIONS LOGIQUES.

6.3.1 - Le niveau Fichier.

a) **Généralités**

Les requêtes du niveau fichier dans les cas Indexé et Indexé Bufferisé sont semblables à celles des autres méthodes d'accès, leur fonctionnement est décrit dans le chapitre 4 : l'entité Fichier. Il n'est décrit ici que les particularités liées à la méthode d'accès Indexé. Elles ne concernent que les requêtes : CREAT, OPEN NEW.

La seule différence entre l'Indexé et l'Indexé Bufferisé réside dans le fait que seul l'Indexé Bufferisé est capable de prendre en compte un buffer de travail fournit par l'utilisateur de façon optionnelle lors des requêtes : CREAT, OPEN NEW, OPEN OLD.

L'interface d'appel d'une requête du niveau fichier est en PL 1600.

```
RA := @FCB ;
SVC (FMS) ;     où      FMS =` 38
```

b) CREAT, OPEN NEW : **Création d'un fichier Indexé**

Paramètres spécifiques de la création d'un fichier indexé :

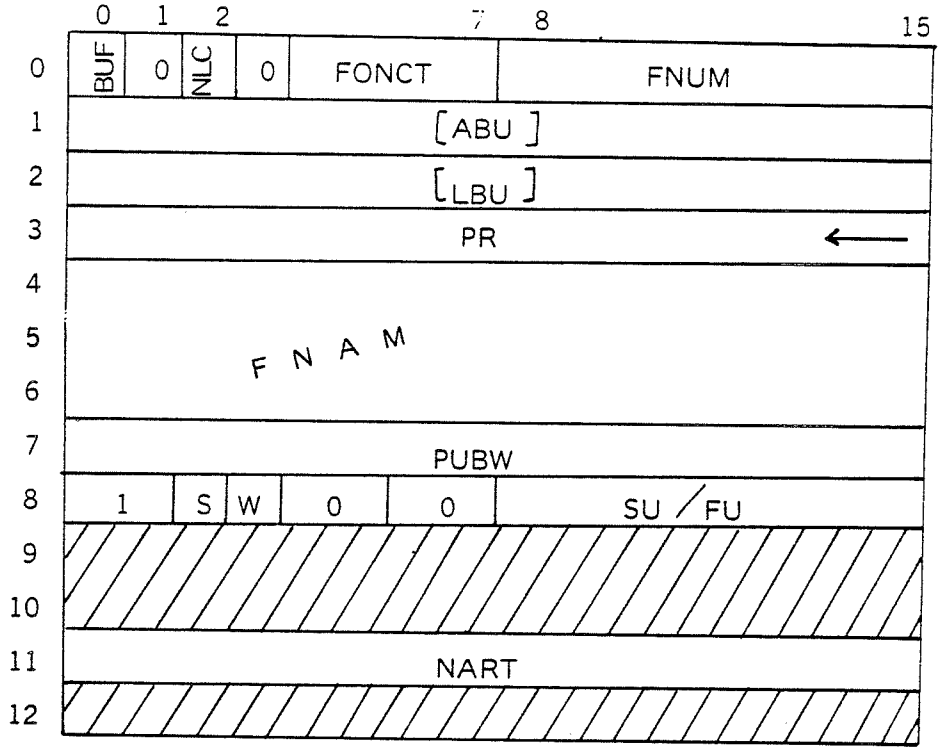
- Organisation logique numéro 1
- le nombre maximum d'articles (NART) que le fichier devra pouvoir contenir :
  $1 \leqslant NART \leqslant 8159$

La taille des articles est définie à la création des articles.
FMS alloue à la création du fichier la place nécessaire à la Table d'Index, et initialise celle-ci à zéro. Après la création un Fichier Indexé est constitué d'une table d'index nulle et le pointeur courant coincide avec la fin du fichier.
Un fichier Indexé est créé avec une organisation physique séquentielle.

Nom	CREAT	Indexé
But	Créer un fichier Permanent Indexé et créer une unité d'Accès à ce fichier.	

Appel RA : = @FCB ; SVC (FMS) ; ou FMS = '38

FCB



Description des paramètres

NART : Nombre d'articles tel que :
1 ⩽ NART ⩽ 8159

Comptes Rendus

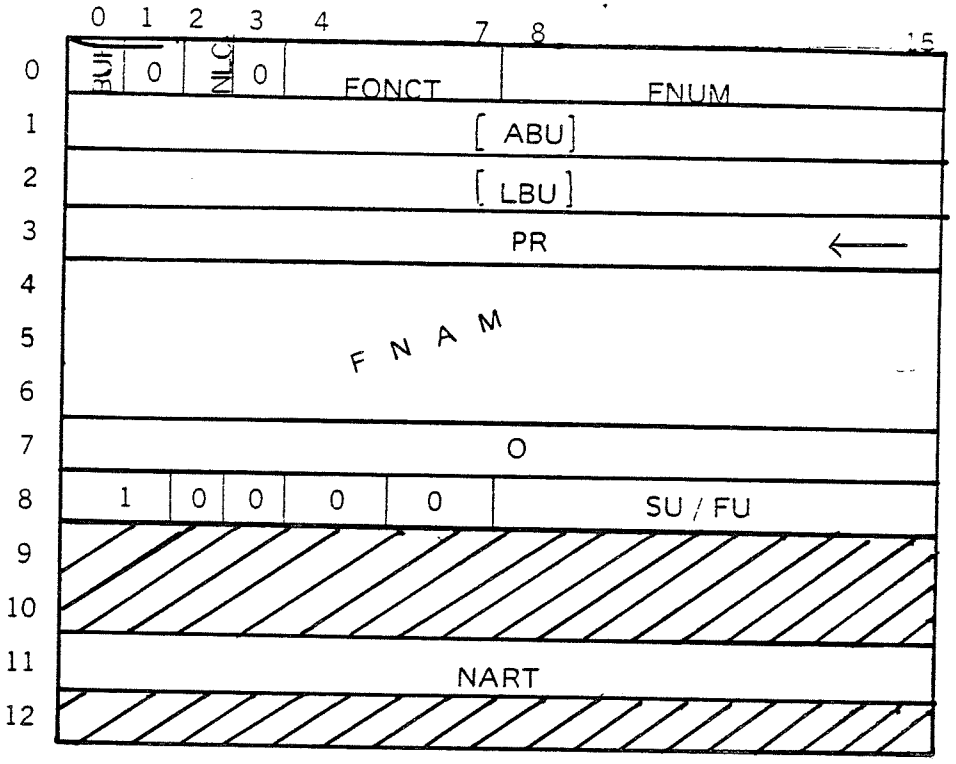
Valeur de PR	Signification
0	**Primitive correctement exécutée.**
' 600B	**FAU existante.**
' 600D	**Fichier existant.**
' 601F	**Primitive en cours.**
' 6020	**Zone de Pavés saturée.**
' 6021	**FU saturée :** la FU ne contient pas assez de granules libres pour allouer la place nécessaire à la Table d'index. La primitive est ineffective.
' 6022	**Table des Fichiers saturée.**
' 6028	**Erreur de Syntaxe :** (@FCB, FONCT, [ABU, LBU,] FNAM PUBW, FTYP, NART)
' 6029	**Adresse de FCB invalide.**
' 602A	**SU ou FU non gérée par FMS.**
' 602B	**Méthode d'accès non gérée.**

Erreurs Graves	`6032	Informations Système Invalides
	`6033	
	`6034	
	`6035	FU verrouillée par IOCS.
	`4.....	Erreur hardware : `4000 = mot d'état PU.

Nom	OPEN NEW	Indexé
But	Créer un fichier Temporaire indexé et créer une unité d'accès à ce fichier.	

Appel : RA := @FCB ; SVC (FMS) ; ou FMS = `38

FCB



Description des paramètres

NART : Nombre d'articles tel que :
1 ⩽ NART ⩽ 8159

Comptes rendus

Valeur de PR	Signification
0	**Primitive exécutée correctement**
`600B	**FAU existante**
`600D	**Fichier existant**
`601F	**Primitive en cours**
`6020	**Zone de Pavés saturée.**
`6021	**FU saturée :** La FU ne contient plus un seul granule de libre. La primitive est ineffective.
`6028	**Erreur de Syntaxe :** (FCB, FONCT, [, ABU, LBU], FNAM, FTYP, NART).
`6029	**Adresse de FCB invalide**
`602A	**SU ou FU non gérée par FMS**
`602B	**Méthode d'accès non gérée.**

Erreurs Graves	`6032 `6033 `6034	Informations Systèmes Invalides
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU.

6.3.2 - Le niveau Article.

a) **Généralités**

L'Indexé fournit 6 primitives d'accès direct à l'article :
IREAD, IWRITE, ISUP, IRNAM, IRWITE, IRTIX.

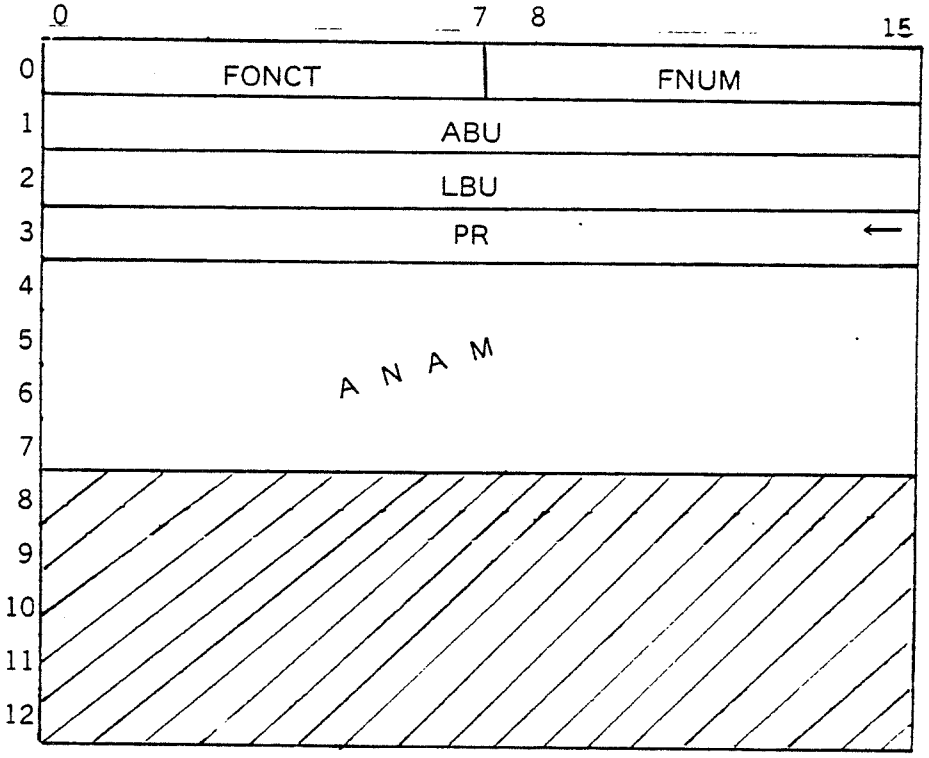
L'Interface d'appel d'une requête du niveau article pour la méthode d'accès : Indexé est en PL 1600.

RA : = @FCB ;
SVC (FMSI) ; où FMSI = '3A

L'Indexé et l'Indexé Bufferisé possèdent le même interface de programmation.

Description générale des FCB.

FCB



Conventions.

- les zones hachurées indiquent : zone non utilisée par FMS.
- ⟵ paramètre de retour fournit par FMS.
- dans les autres cas le paramètre doit être fournit par l'utilisateur.

Toute requête du niveau article s'adresse à l'unité d'accès spécifiée par l'utilisateur (FNUM) et concerne le fichier accessible par celle-ci. Toute requête du niveau article interdit l'accès séquentiel pur statique sur cette unité d'accès, et ceci jusqu'à sa destruction. Le choix d'une primitive parmi celles offertes par la méthode d'accès est fait par l'usager à l'aide de l'octet FONCT.

Le nom d'un article (ANAM) est composé de 8 caractères ASCII (2 caractères par mot) appartenant à l'ensemble des 40 caractères suivants :

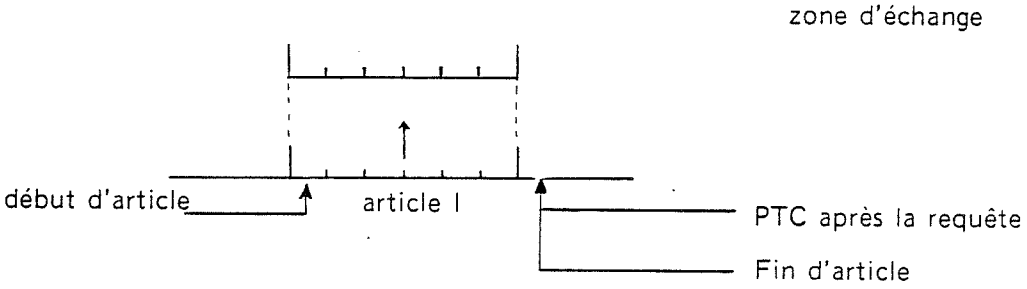
- les lettres [A ; Z]
- les chiffres [0 ; 9]
- :
- ;
- ∅
- Nul (en fin seulement).

b) IREAD : **Lecture d'article.**

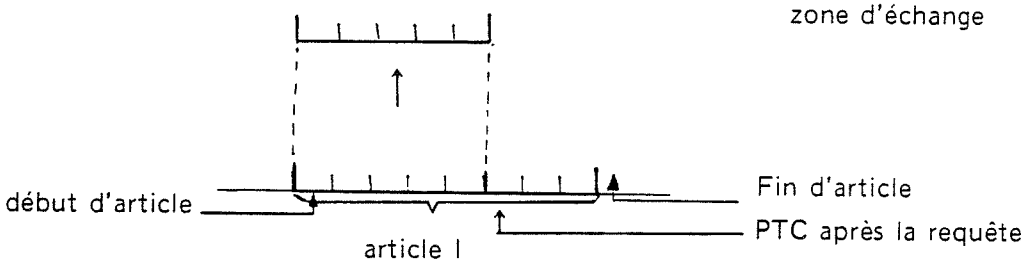
Cette primitive permet de lire un article désigné par son **nom** (ANAM) dans la zone d'échange spécifiée par l'utilisateur. L'article est cadré à gauche dans la zone d'échange. Cette primitive recherche séquentiellement le **nom** de l'article dans la TIX à fin de connaître son adresse relative dans le fichier.

La primitive IREAD sélectionne l'article dans les cas suivants, en vue de lui changer son nom (IRNAM), ou en vue d'un accès séquentiel selon le mode Portion d'Article Statique.

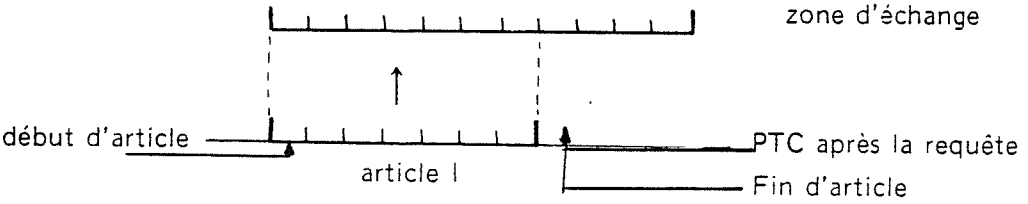
PR = 0 Primitive exécutée correctement.



PR = 6003 Article du fichier plus long que la zone d'échange.



PR = 6004 Article du fichier plus court que la zone d'échange.

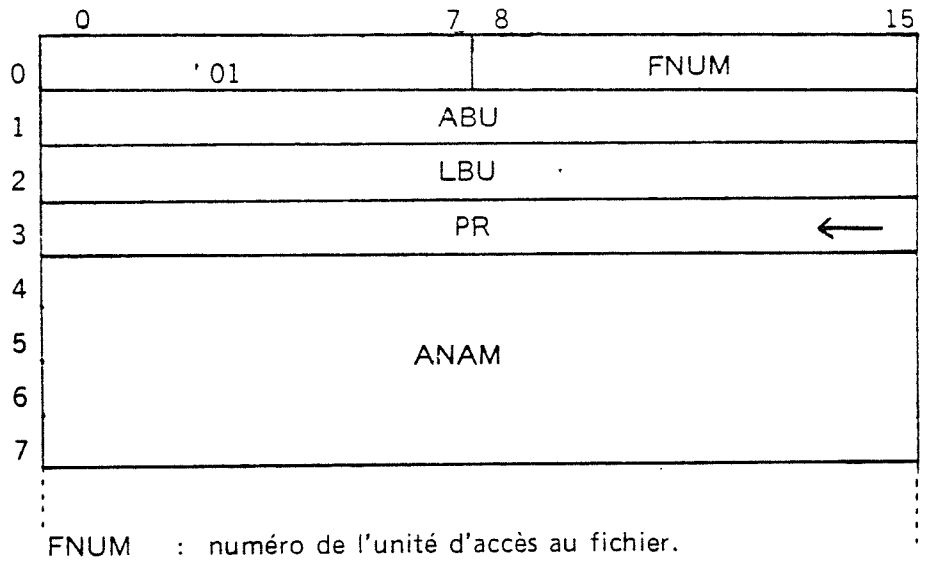


Pour les autres valeurs de PR la primitive IREAD invalide la sélection d'article, et interdit tout accès séquentiel sur cette unité d'accès. (Voir exceptions chapitre 6.3.3. (d)).

Nom	**IREAD**
But	**Lire un article d'un fichier indexé.**

Appel RA := @FCB ; SVC (FMSI) ou FMS I = `3A

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.
ABU : adresse de la zone d'échange de l'utilisateur, qui recevra les octets lus.
LBU : nombre d'octets à lire. $0 \leqslant LBU \leqslant$ `FFFE (apairé par FMS).
ANAM : nom de l'article à lire 8 caractères ASCII appartenant à la plage valide, chapitre 6.3.2, § (a).
PR : paramètres de retour.

Compte rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée.**
`6003	**Article du fichier plus long :** l'article du fichier est plus long que la zone d'échange spécifiée dans le FCB. La zone d'échange entière est remplie avec le début de l'article.
`6004	**Article du fichier plus court :** l'article du fichier est plus court que la zone d'échange spécifiée dans le FCB. Tout l'article est placé dans la zone d'échange. Il est cadré à gauche, le reste de la zone d'échange est inchangé.
`600A	**FAU inexistante.**
`600E	**Article inexistant.** La table d'Index ne contient pas le nom désigné par l'utilisateur dans le FCB. La primitive est ineffective du point de vue de la lecture de l'article. Voir chapitre 6.3.3. § (a).
`6018	**Incompatibilité primitive fichier.**
`601F	**Primitive en cours.**
`6028	**Erreur de syntaxe.** (@ FCB, FONCT, ABU, LBU, ANAM)
`6029	**Adresse de FCB invalide.**
`602B	**Méthode d'accès non gérée.**

Erreurs Graves	`6032 `6034	Informations Systèmes Invalides.
	`6035	FU verrouillée par IOCS.
	`4.....	Erreur hardware : `4000 + mot d'état PU.

c) IWRITE : **Création d'un nouvel article**

Cette primitive permet de créer un article désigné par son **nom** (ANAM). La zone d'échange contient le début de l'article.

FMS vérifie que le fichier ne contient pas d'article de même nom. Cette requête mettra à jour la table d'index en y indiquant le nom du nouvel article.

Cette primitive s'utilise pour créer un article selon 2 modes différents :

- création au niveau article seulement.
- création au niveau article et portion d'article.

1) **Création au niveau article seulement.**

La zone d'échange associée à la requête IWRITE contient toute l'information de l'article. Sa longueur (LBU) définit donc la longueur de l'article. Elle est spécifiée à l'aide d'un nombre d'octets, tel que :

$$0 \leqslant LBU \leqslant `FFFE$$

- Si le compte d'octet est impair, il sera automatiquement apairé par FMS.
- Il est déconseillé de créer des articles de longueur nulle.

La requête IWRITE alloue dynamiquement l'article à la fin du fichier.

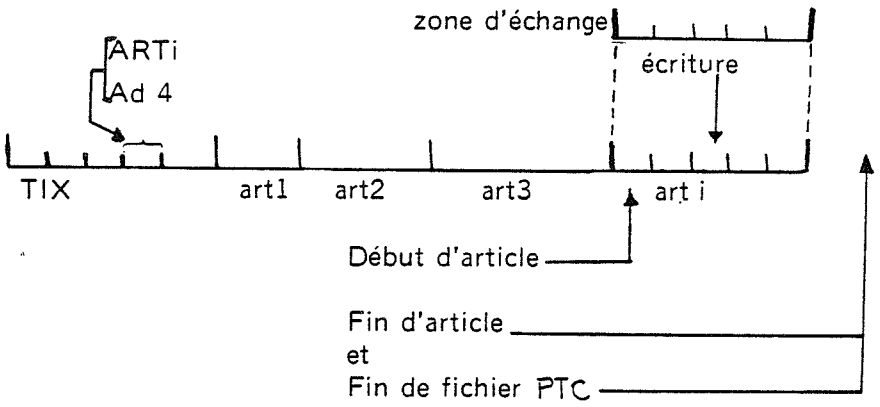
Remarque :

Cette méthode permet de créer des articles de 32 K mots - 1 maximum.

* **Allocation dynamique à la fin du fichier**
Cette création utilise un nouvel élément de table d'index, et agrandit le fichier de la longueur de l'article.

Dans ce cas la requête IWRITE
- fournit un compte rendu nul (PR = 0)
- réalise une sélection d'article pour :
  - * IRNAM
  - * un accès séquentiel à l'article selon le mode Portion d'Article Dynamique. Cette sélection sera utilisée dans le cas 2) décrit ci-après.

Exemple : IWRITE ARTi



2) **Création au niveau Article et portion d'article.**

- Réaliser les créations d'article : IWRITE (ARTI , avec LBU = x)
- Remplir les articles avec les requêtes WRITE Séquentiel.

Dans ce cas la longueur de l'article n'est pas définie par la requête IWRITE, mais par la taille de la portion d'article dynamique au moment où la création est terminée. (par une autre requête du niveau article ou par la destruction de l'unité d'accès au fichier (CLOSE, DELET).. Voir chapitre 6.3.3 § (d)).

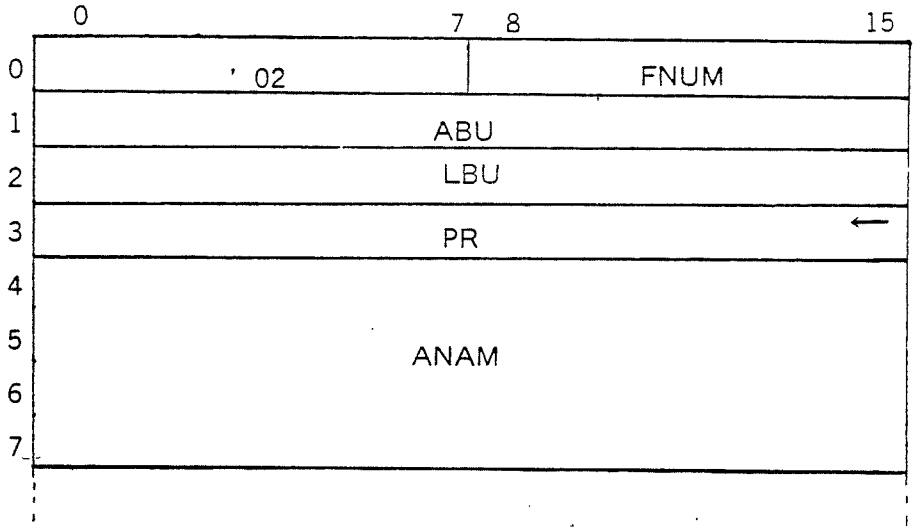
Remarque : Pendant la phase de création de l'article, la séquence REWIND + WRITE a pour effet de détruire puis recréer dynamiquement le contenu de l'article.

Remarque : Cette méthode permet de créer des articles de 64 K mots − 1. Ils devront être relus avec IREAD + READ séquentiel.

Nom	IWRITE
But	Créer un nouvel article dans un fichier indexé.

Appel RA : = @FCB ; SVC (FMSI) ; ou FMSI = `3A

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.
ABU : adresse de la zone d'échange de l'utilisateur qui contient l'article.
LBU : longueur en octets de l'article à créer : 0 ⩽ LBU ⩽ ` FFFE (apairée par FMS).
ANAM : nom de l'article à créer. 8 caractères ASCII (2 caractères par mot) appartenant à la plage valide, chapitre 6.3.2, § (a).

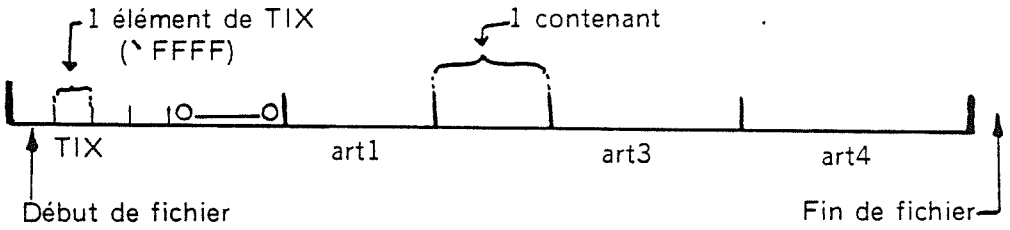
Comptes rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée.**
` 600A	**FAU inexistante.**
` 600F	**Article existant** .La table d'index contient le même nom que celui désigné dans le FCB. La primitive est ineffective du point de vue de la création du nouvel article. Cependant elle a éventuellement terminée la création d'un article selon le mode portion d'article dynamique. Elle a également annulé toute sélection.
` 6014	**Protection écriture.** La primitive est ineffective. Voir chapitre 6.3.3. § (d)..
` 6016	**Fichier saturé :** La table d'Index est saturée. Le nombre d'article demandé à la création du fichier est atteint. Utiliser FUP5 pour agrandir la Table d'Index. La requête est ineffective.
` 6017	**Fichier trop long.** La taille de l'article est telle que la partie information du fichier (les articles) dépasserait 64 K mots. La primitive est ineffective. Voir chapitre 6.3.3 § (d).

	`6018	**Incompatibilité primitive - Fichier :**
	`601F	**Primitive en cours**
	`6021	**FU saturée :** La création de l'article se fait à la fin du fichier avec allocation dynamique (LBU ≠ 0) et il n'y a pas assez de granules libres sur la FU. La primitive est ineffective. Voir chapitre 6.3.3 (d)
	`6028	**Erreur de syntaxe :** (FCB, FONCT, ABU, LBU, ANAM)
	`6029	**Adresse de FCB invalide**
	`602B	**Méthode d'accès non gérée.**
Erreurs Graves	`6032	**Informations Systèmes Invalides.**
	`6034	
	`6035	**FU verrouillée par IOCS**
	4 . . .	**Erreur hardware :** `4000 + mot d'état PU.

d) ISUP : **Suppression d'un article.**
Cette primitive permet de supprimer un article désigné par son **nom** (ANAM). Cette primitive a pour effet de rechercher l'article dans la Table d'Index et d'y indiquer que l'article est supprimé.

Une place libre : ISUP (ART2)



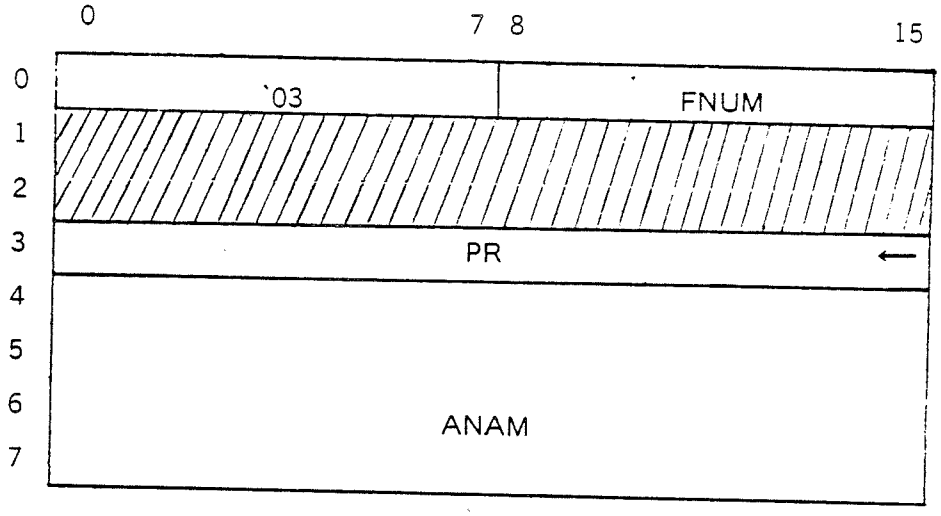
Cette primitive interdit tout accès séquentiel sur l'unité d'accès concernée.

La requête ISUP récupère la place occupée par l'index d'un article supprimé, si celui-ci se trouve à la fin du fichier. Elle désalloue également la place occupée par les trous situés à la fin du fichier c'est-à-dire ceux qui précèdent un article supprimé en fin de fichier. La place est en fait récupérée par la requête IWRITE suivante.

Nom	ISUP
But	Supprimer un article dans un fichier indéxé

Appel RA : = @ FCB ; SVC (FMS I) ; où FMSI = '3A

FCB



Désignation des paramètres

FNUM : numéro de l'unité d'accès au fichier.
ANAM : nom de l'article à supprimer. 8 caractères ASCII appartenant à l'ensemble valide, chapitre 6.3.2§(a).
PR : paramètre de retour.

Comptes rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée**
' 600A	**FAU inexistante.**
' 600E	**Article inexistant.**
' 6014	**Protection écriture.** La primitive est ineffective. Voir chapitre 6.3.3. § (d).
' 6018	**Incompatibilité primitive - fichier.**
' 601F	**Primitive en cours.**
' 6028	**Erreur de syntaxe : (@FCB, FONCT, ANAM)**
' 6029	Adresse de FCB invalide.
' 602B	Méthode d'accès non gérée.

Erreurs Graves

' 6032 ' 6034	Informations Systèmes Invalides.
' 6035	FU verrouillée par IOCS
' 4.....	Erreur hardware : ' 4000 + mot d'état PU.

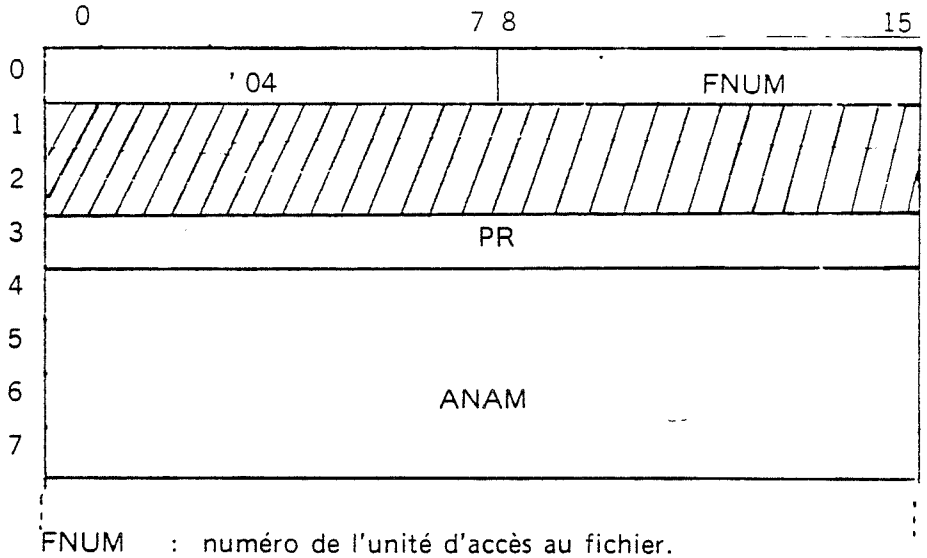
e) IRNAM : **Renommer l'article en cours.**
Cette primitive permet de changer le **nom** de l'article **en cours**, par celui fournit dans le FCB.
Cette primitive fonctionne lorsque un article est correctement sélectionné.
IREAD, IWRITE, IRWRITE sont les 3 primitives qui permettent de sélectionner un article d'un fichier Indexé. Après le contrôle de non homonymie, le nouveau nom est mis à jour dans la table d'Index.

Cette primitive interdit tout accès en séquentiel sur l'unité d'accès concernée.

Nom	IRNAM
But	Renommer l'article en cours

Appel RA := o FCB ; SVC (FMSI) ; où FMSI = 'A

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.
ANAM : nouveau nom à donner à l'article sélecté. caractères ASCII (2 caractères par mot) appartenant à l'ensemble valide, chapitre 6.3.2 § (a).
PR : paramètre de retour.

Comptes rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée.**
'A	**FAU inexistante**
'F	**Article existant :** le nouveau nom de l'article précisé dans le FCB existe déjà dans la Table d'index. La première est ineffective. Mais il n'y a plus de sélection. Voir chapitre 6.3.3 (d).
'14	**Protection écriture** : La primitive est ineffective, voir chapitre 6.3.3 (d).
'18	**Incompatibilité primitive - fichier**
'1A	**Erreur d'enchainement.** Aucun article n'était sélectionné (IREAD, IWRITE, IRWRITE). Voir chapitre 6.3.3 (d).
'1F	**Primitive en cours.**
'28	**Erreur de syntaxe :** (FCB, FONCT, ANAM)
'29	**Adresse de FCB invalide**
'2B	**Méthode d'accès non gérée.**

Erreurs Graves

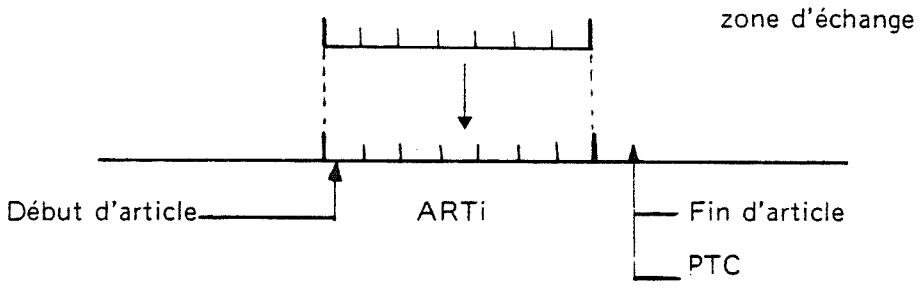
'32, '34	**Informations Systèmes Invalides.**
'35	**FU verrouillée par IOCS**
'4 . . .	**Erreur hardware :** '4000 + mot d'état PU.

f) IRWRITE : **Réécrire un article.**
Cette primitive permet de modifier le contenu d'un article désigné par son nom (ANAM). La zone d'échange contient le début de l'article.

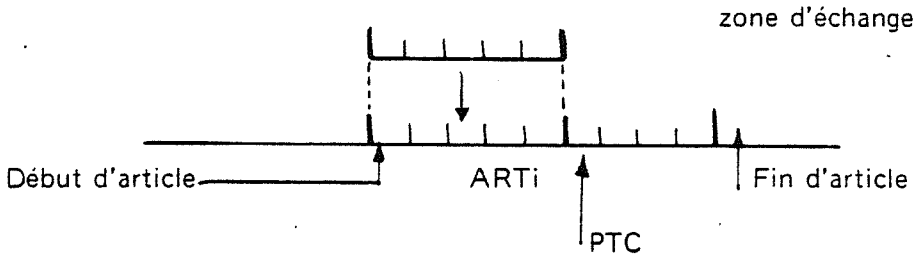
Cette primitive recherche séquentiellement le nom de l'article dans la Table d'index à fin de connaître son adresse relative dans le fichier.

La primitive IRWRITE sélectionne l'article dans les cas suivants, en vue de lui changer son nom (IRNAM), ou en vue d'un accès séquentiel selon le mode Portion d'Article Statique.

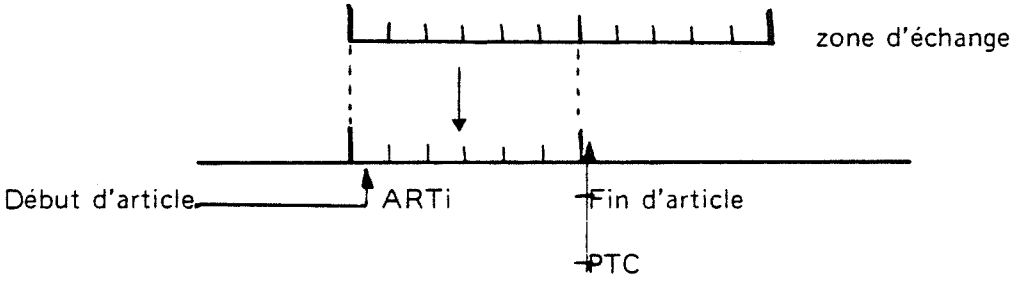
PR = 0 Primitive correctement exécutée



PR = `6003 Article du fichier plus long que la zone d'échange.



PR = `6004 Article du fichier plus court que la zone d'échange.



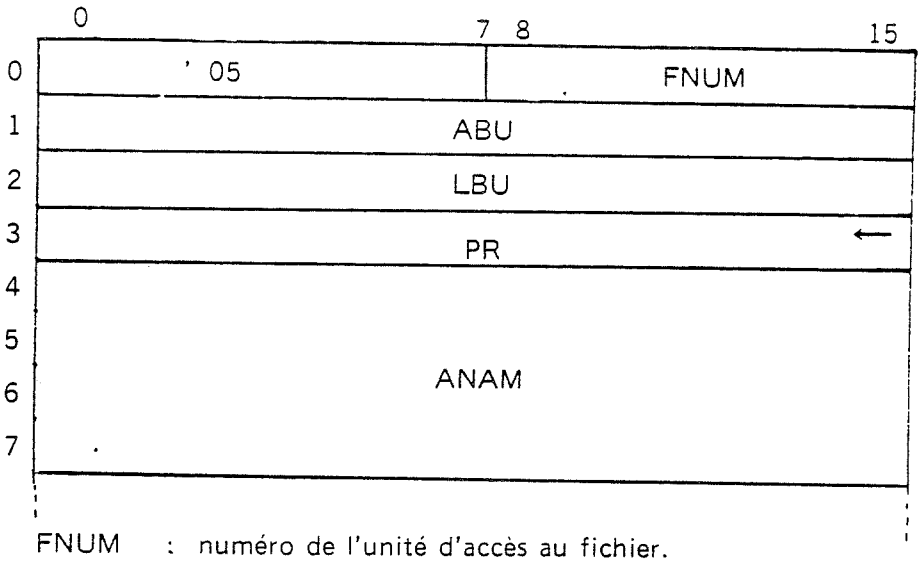
Pour les autres valeurs de PR la primitive IRWRITE invalide toute sélection et interdit tout accès séquentiel sur cette unité d'accès.

Nom IRWRITE

But Réécrire le contenu d'un article existant, dans un fichier indexé.

Appel RA := @ FCB ; SVC (FMS I) où FMSI = `3A

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.
ABU : adresse de la zone d'échange de l'utilisateur qui contient le nouveau contenu de l'article.
LBU : nombre d'octets à écrire : $0 \leqslant LBU \leqslant$ `FFFE (apairé par FMS)
ANAM : nom de l'article à réécrire. 8 caractères ASCII appartenant à l'ensemble valide, chapitre 6.3.2 § (a).
PR : paramètre de retour.

Comptes rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée.**
`6003	**Article du fichier plus long.** L'article sur disque est plus long que le buffer indiqué par la primitive. Le début de l'article est réécrit avec le buffer ; le reste de l'article sur disque est inchangé.
`6004	**Article du fichier plus court.** l'article sur disque est plus court que le buffer indiqué par la primitive. Tout l'article sur disque est réécrit avec le début du buffer.
`600A	**FAU inexistante.**
`600E	**Article inexistant dans le fichier.** La primitive est ineffective. Voir chapitre 6.3.3 § (d)
`6014	**Protection écriture.** La primitive est ineffective. Voir chapitre 6.3.3 § (d).
`6018	**Incompatibilité primitive fichier.**
`601F	**Primitive en cours.**
`6028	**Erreur de Syntaxe** (@ FCB, FONCT, ABU, LBU, ANAM)
`6029	**Adresse de FCB invalides.**
`6029	**Méthode d'accès non gérée.**

Erreurs Graves	`6032 `6034	} Informations Systèmes Invalides.
	`6035	FU verrouillée par IOCS.
	`4.....	Erreur hardware : `4000 + mot d'état PU.

g) IRTIX : **charger la table d'index en mémoire centrale.**
FMS permet d'utiliser un fichier indexé de deux façons différentes.

1) **Table d'Index sur mémoire secondaire**
Dans ce cas elle est relue, et/ou réécrite par FMS à chaque requête du niveau article à l'aide d'un buffer du système (de taille 1 secteur).
Après la création d'une Unité d'Accès (CREAT, OPEN NEW, OPEN OLD) FMS-E considère que la TIX est à exploiter à partir de la mémoire secondaire.

2) **Table d'Index en mémoire centrale**
FMS permet de rendre résidente tout ou partie de la TIX dans un buffer appartenant à l'usager. Il faut pour cela utiliser la requête IRTIX spécifiant l'adresse et la longueur du buffer qui sera exploité par FMS.

Alors, chaque primitive du niveau article sera précédée par un balayage de la table d'index en mémoire centrale, avec rechargement du buffer, que si le nom d'article recherché ne s'y trouve pas. A la libération de la ressource, c'est-à-dire à la destruction de l'unité d'accès (CLOSE), FMS si nécessaire, mettra à jour la table d'index sur disque. Il est dans ce cas, recommandé à l'utilisateur de ne pas modifier la zone où est située la table d'index.

La primitive IRTIX interdit tout accès séquentiel sur l'unité d'accès concernée.

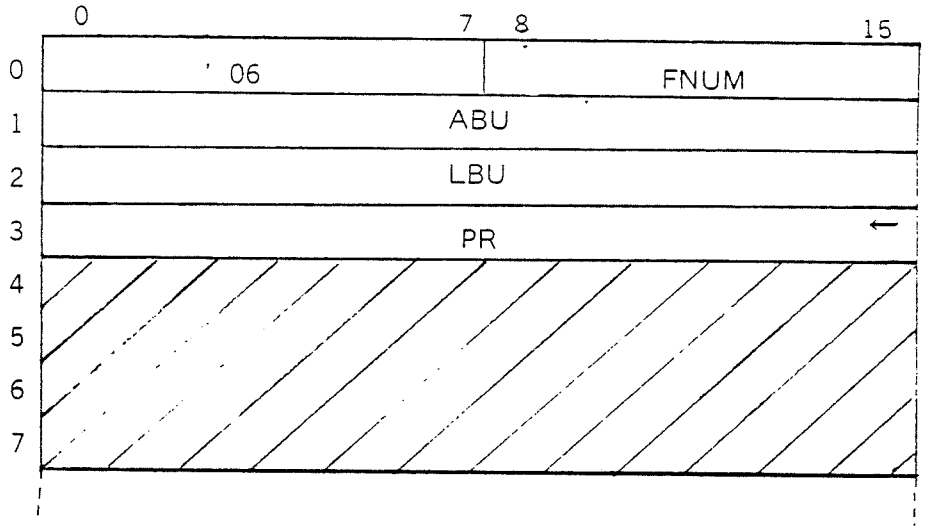
Remarque :

Pour que l'algorithme de pagination sur la table d'index fonctionne avec efficacité il est préférable de placer dans le même secteur les index des articles qui ont une forte probabilité d'être utilisés en même temps.

Nom	IRTIX
But	Charger la table d'index du fichier indexé en mémoire.

Appel : RA : = @ FCB ; SVC (FMS I) ; ou FMSI = ' 3A

FCB



description des paramètres

FNUM	:	numéro de l'unité d'accès au fichier.
ABU	:	adresse du buffer de l'utilisateur qui recevra la table d'index
LBU	:	longueur du buffer de l'utilisateur (en octets)
PR	:	paramètre de retour.

Comptes rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée.**
' 6003	**Article du fichier plus long.** La table d'index est plus longue que le buffer donné par l'utilisateur. La primitive est effective.
' 6004	**Article du fichier plus court.** La table d'index est plus courte que le buffer donné par l'utilisateur. La table d'index entière est mise dans le début du buffer. La Table d'Index sera exploitée par FMS. Le reste du buffer est inchangé.
' 600A	**FAU inexistante.**
' 6018	**Incompatibilité primitive-fichier.**
' 601A	**Erreur d'enchaînement.** Une primitive IRTIX a déjà été envoyée pour cette FAU. La primitive est ineffective. Voir chapitre 6.3.3 par. (d).
' 601F	**Primitive en cours.**
' 6028	**Erreur de syntaxe.** (@ FCB, FONCT, ABU, LBU)
' 6029	**Adresse de FCB invalide.**
' 602B	**Méthode d'accès non gérée.**

Erreurs Graves

' 6032	**Informations Systèmes Invalides**
' 6034	
' 6035	**FU verrouillée par IOCS**
' 4 . . .	**Erreur hardware :** ' 4000 + mot d'état PU

6.3.3 - Le niveau Portion d'Article.

a) **Portion d'Article Statique**

L'accès séquentiel à un article selon le mode Portion d'Article Statique (P.A.S.) est initialisé de façon identique par les requêtes :

- IREAD avec PR = [0, `6003, `6004].
- IRWRITE avec PR = [0, `6003, `6004].
- IWRITE seulement lorsqu'il y a récupération d'une place libre (PR = 0).

exemple :

1) Ouvrir le fichier de nom FIC - FM se trouvant sur la FU D 3.
   OPEN OLD (1, FIC - FM, D3 - - - - -)
   FMS ⇒ PR = 0

2) Sélectionner dans le fichier FIC - FM, D 3, l'article de nom ART 3 (longueur 25 mots) et en lire les 5 premiers mots.
   IREAD (1, ABU, LBU = 10, PR, ART3)
   FMS ⇒ PR = `6003

3) Lire l'article en séquentiel par portions de 10 mots, jusqu'à la fin de l'article.
   READ (1, ABU, LBU = 20, PR)
   FMS ⇒ PR = 20
   3ème fois FMS ⇒ PR = `6001

4) Modifier le 6è mot de l'article.
   REWIND (1, PR)
   FMS PR = 0

   SKIPF (1, CA = 0, LBU = 10, PR)
   FMS ⇒ PR = 10
   La zone d'échange contient la nouvelle valeur du 6è mot.
   WRITE (1, ABU , LBU = 2, PR)
   FMS ⇒ PR = 2

   * Autre solution pour se positionner sur le 6è mot de l'article alors que PTC désigne la fin de l'article.
     SKIPB (1, CA = 0, LBU = 40, PR)
     FMS ⇒ PR = 40

5) Fermer le fichier FIC - FM, D3.
   CLOSE (1, PR)
   FMS ⇒ PR = 0.

b) **Portion d'Article Dynamique.**

L'accès séquentiel à un article en vue de créer son contenu, selon le mode Portion d'Article dynamique (P.A.D.) est permis après la requête :

— IWRITE seulement lorsque la création de l'article se situe à la fin du fichier (Pas de récupération de place), (PR = 0).

Exemple :

Décrivant en particulier le fonctionnement du BUILDER de BOS16 lors de la création d'un fichier image mémoire.

1) Création d'un fichier temporaire Indexé.
(FNUM et FNAM, Système)
OPEN NEW (x, - - - - - - - - - -)
FMS ⇒ PR = 0

2) Création d'un article de nom BR1
IWRITE (x, ABU, LBU = 0, PR, BR1)
FMS ⇒ PR = 0

3) Création du contenu de l'article

WRITE (x, ABU , LBU = 80, PR) ↻ n — 1 fois
FMS ⇒ PR = 80

nième fois
WRITE (x, ABU , LBU 80 , PR)
FMS ⇒ PR = LBU

4) Le traitement décrit en 2) et 3), permet de créer autant d'articles que de branches et un article de nom ROOT contenant la racine et la dernière branche. Lors de chaque IWRITE la Portion d'Article Dynamique précédente est terminée et la longueur de l'article correspondant est validée ainsi que son contenu.

5) Création de l'article de nom DESC : Descripteur de l'image mémoire.
IWRITE (x, ABU , LBU = 46, PR)
FMS ⇒ PR = 0
La création de l'article ROOT selon le mode P.A.D. est terminée. La création de l'article DESC est logiquement terminée. Pour FMS elle ne le sera que par la fermeture du fichier. (Le pointeur de fin de fichier est mis à jour en mémoire centrale **seulement**, puisque le fichier est Temporaire. Autrement dit le fichier est dans un état instable.

6) Normalement l'utilisateur transforme le fichier en Permanent.
— Commande de l'utilisateur :
CATAL IM, FIC - FM, D3
— Requête correspondante envoyée par BOS16
CATAL (x, FIC - FM, D3 - - -)
FMS ⇒ PR = 0.
La mémoire secondaire contient alors une description cohérente du fichier. La portion d'article dynamique n'est cependant toujours pas terminée pour FMS.

7) Fermeture du fichier Permanent.
1ère Solution :
— Commande utilisateur : CLOSE IM
— Requête correspondante envoyée par BOS16 : CLOSE (x, PR)
FMS ⇒ PR = 0

2ème Solution :

— Commande utilisateur : EOJ
— Requête correspondante envoyée par BOS16 : EOJ (USR, PR)
FMS ⇒ PR = 0

c) **Séquentiel Pur Statique.**

L'accès séquentiel à tout le fichier considéré comme 1 seul article selon le mode Séquentiel Pur Statique (S.P.S.) est permis après l'une des 3 requêtes :

— OPEN NEW avec PR = 0
— OPEN OLD avec PR = 0
— CREAT avec PR = 0

exemple :

Lire le contenu d'un fichier Indexé en séquentiel.

1) Ouvrir le fichier. nom : FIC - FM, D3
OPEN OLD (1, FIC- FM , D3 - - - -)
FMS ⇒ PR = 0

2) Lecture séquentielle par portion d'articles de 128 mots.

READ (1, ABU ' LBU = 256, PR)
FMS ⇒ PR = 256 ↺ n — 1 fois

La n^e fois, soit L la longueur du fichier (Table d'Index plus articles)
Si L = (n - 1). 128 mots + 3 mots.

FMS ⇒ PR = 6

la n + 1^e fois.

FMS ⇒ PR = 6001

3) Fermeture du fichier.
CLOSE (1, PR)
FMS ⇒ PR = 0

d) **La Portion d'article et les comptes rendus.**

Lorsqu'une des six requêtes (du niveau article) est demandée à l'indexé, la création d'un article en séquentiel selon le mode Portion d'Article Dynamique (P.A.D.) peut être en cours. Cette requête termine alors la création de l'article. Les exceptions en fonction du compte rendu de requête sont décrites dans le tableau suivant :

- OUI : La P.A.D. en cours est terminée. Selon la requête, une nouvelle sélection d'article est réalisée.
- NON : La primitive est ineffective, la sélection de l'article en cours est inchangée. En particulier une création d'article selon le mode P.A.D. peut continuer.
- MAJ : La P.A.D. en cours est terminée, mais la sélection d'un nouvel article n'est pas réalisée. Tout accès séquentiel est interdit jusqu'à la prochaine sélection valide.
- NMAJ : La P.A.D. est anormalement terminée. Tout accès séquentiel est interdit sur l'unité d'accès concernée. Il faut terminer la création correctement (une requête du niveau article ou un CLOSE)

Le tableau suivant indique également la liste des comptes rendus de l'ensemble des requêtes de l'indexé.

Indexé Valeur de PR	Signification		P.A.D. Préc. Terminée
0	Primitive correctement exécutée		OUI
`6003	Article du fichier plus long		OUI
`6004	Article du fichier plus court		OUI
`600A	FAU inexistante		NON
`600E	Article inexistant		MAJ
`600F	Article existant		MAJ
`6014	Protection écriture		N MAJ
`6017	Fichier trop long		MAJ
`6018	Incompatibilité primitive - Fichier		NON
`601A	Erreur d'enchainement		MAJ
`601F	Primitive en cours		NON
`6021	FU saturée		MAJ
`6028	Erreur de syntaxe	FCB	N MAJ
		FONCT	NON
		ABU	N MAJ
		LBU	NON
		ANAM	NON
`6029	Adresse de FCB invalide		NON
`602B	Méthode d'accès non gérée		NON
`6032	Informations Système Invalide		N MAJ
`6034	"		"
`6035	FU verrouillée par IOCS		"
`4.....	Erreur hardware		"

7 — LE FICHIER DE TYPE DIRECT

7 — LE FICHIER DE TYPE DIRECT

7.1 - ORGANISATION LOGIQUE.

L'organisation logique d'un fichier de Type **Direct** est définie à la création du fichier, elle possède le **numéro 2.**

Organisation : **Direct.**

Un fichier Direct est constitué d'un, ou plusieurs, articles de même taille, identifiés par un numéro de 1 à N. Le nombre et la taille des articles sont définis à la création du fichier, ainsi que leurs contenants respectifs.

Règles : Direct :
— le nombre d'article (s) (NART) est un nombre entier tel que :

$$1 \leqslant \text{NART} \leqslant 65\,535$$

— La taille des articles (TART) représente un nombre entiers de mots, elle est cependant exprimée à l'aide d'un nombre pair d'octets tel que :

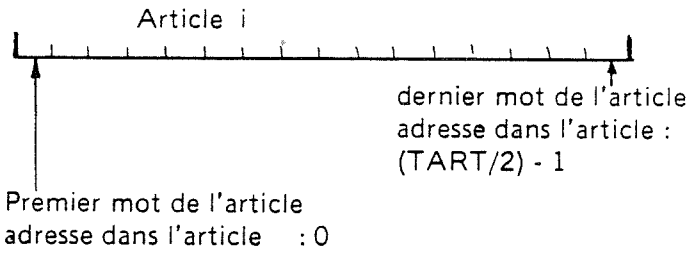
$$2 \leqslant \text{TART} \leqslant 65\,534$$

La taille d'un fichier Direct est fixe, elle est définie implicitement à sa création par (TART x NART)
L'allocation physique de **toute** la place nécessaire à un fichier Direct est réalisée automatiquement par FMS à sa création. On dit alors que l'allocation physique d'un fichier Direct est **statique.**
A fin de permettre un **accès** physique direct aux articles, un fichier Direct possède une **organisation** physique directe. Pour cela, l'utilisateur doit respecter la règle suivante.
Règle : Direct :
— Un fichier Direct doit occuper au plus 128 granules.

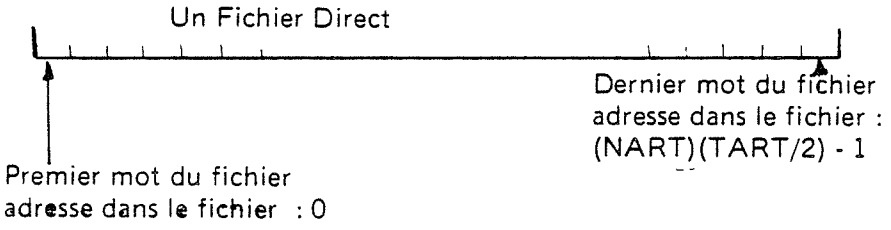
Rappel :
La taille du Granule (TG) est telle que : 3 secteurs $\leqslant$ TG $\leqslant$ la longueur maximum autorisée par IOCS pour un échange sur le périphérique concerné.
Pour plus de précision se reporter au Manuel d'Utilisation de FMS en particulier pour appliquer la règle étendue concernant la Taille des granules (TG) :

$$3 \text{ secteurs} \leqslant \text{TG} \leqslant 256 \text{ secteurs (32 K mots)}$$

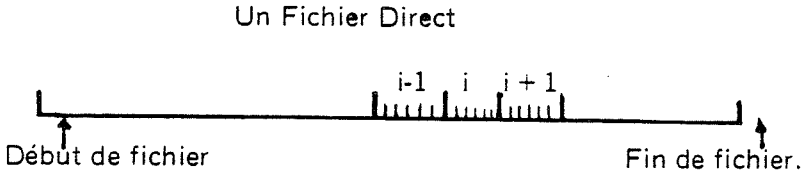
Le précédent schéma décrivant l'organisation logique d'un fichier Direct signifie que :
1) Le fichier est composé d'articles **indépendants.** C'est à dire que chaque article est un vecteur **borné** de mots.



2) Le **fichier** lui-même est un vecteur **borné** de mots.



3) Les **articles** dont les numéros sont **adjacents** sont également adjacents **physiquement** dans le fichier, aux changements de granules près.



7.2 - METHODES D'ACCES

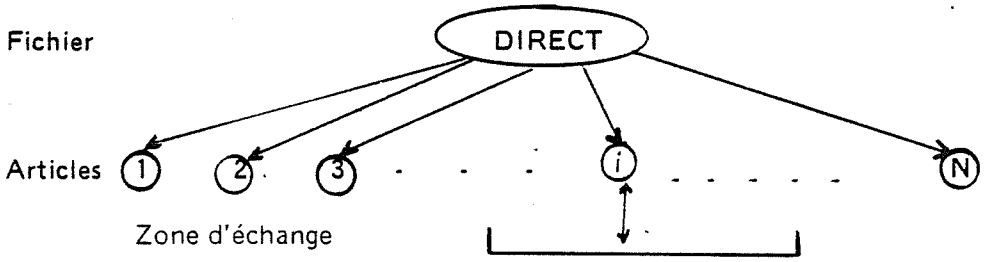
7.2.1 - Le Direct

Le Direct est une méthode d'accès capable de gérer des Unités d'Accès à des fichiers Directs (**Numéro : 2**).
Le Direct est **réentrant**.La réentrance est opérationnelle pour des **taches** différentes accédant à des fichiers Directs par des **unités d'accès** différentes.

Le Direct fournit 2 requêtes d'accès au niveau article : DREAD, DWRITE. Le Direct permet d'utiliser au niveau portion d'article les 6 requêtes de la méthode d'accès : Séquentiel. Le Direct permet pour 1 unité d'Accès à un fichier Direct, 3 modes de fonctionnement.

a) Le Direct fournit des requêtes d'accès direct à l'un quelconque des articles du fichier.

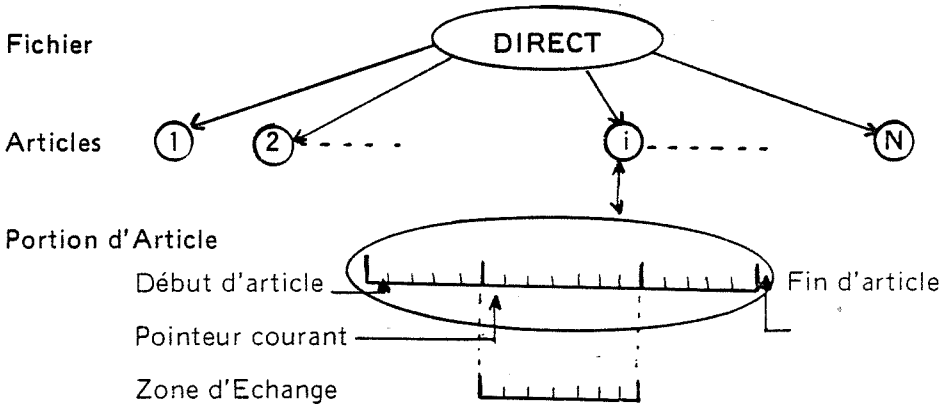
Accès direct aux articles



L'utilisateur peut lire (DREAD) ou réécrire (DWRITE) l'article i.
Le direct ne gère pas l'existence des articles, c'est à dire que dès la création du fichier, tous ses «articles» ou contenants existent et sont accessibles. Le contenu d'un fichier Direct n'est pas initialisé à sa création, il correspond à l'état antérieur de la mémoire secondaire.

b) Le Direct permet un accès séquentiel dans un article.

Accès à la Portion d'Article



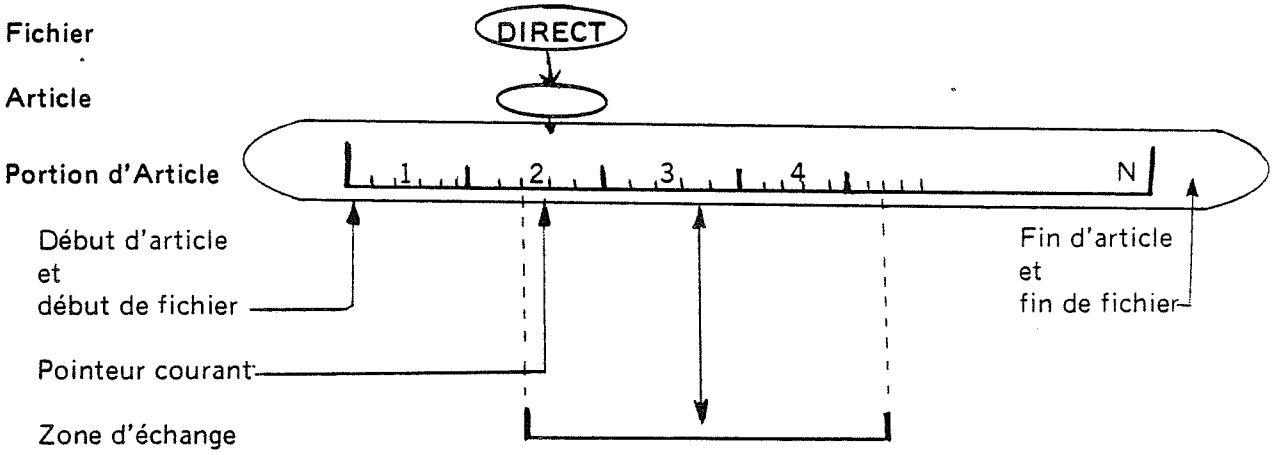
L'utilisateur
1) Sélectionne un article à l'aide d'une requête DREAD ou DWRITE
2) Accède à l'article sélectionné à l'aide de toutes les requêtes de la méthode d'accès Séquentiel.

Le Direct fournit un accès séquentiel **statique** borné à l'article correctement sélectionné. Après la sélection, le pointeur courant est positionné par rapport au début de l'article, à une distance égale à la longueur de l'échange demandé par la sélection. Si cette longueur est nulle le pointeur courant adresse le premier mot de l'article. Pour accéder à un autre article, il faut sélectionner à nouveau.

Un fichier Direct peut être alors considéré comme N «fichiers séquentiels» bornés et indépendants.

c) Le Direct permet un accès séquentiel sur tout le fichier.

Accès Séquentiel Pur Statique



Après l'une des 3 primitives de création d'une Unité d'Accès (OPEN NEW, OPEN OLD, CREAT) et **avant** toute requête du niveau article (ex : DREAD) le Direct permet un accès **séquentiel Pur Statique.** L'usager peut alors utiliser toutes les requêtes de la méthode d'accès : Séquentiel. Tout le fichier est accessible. Après la création de l'Unité d'Accès le pointeur courant adresse le premier mot du fichier. Les frontières d'articles sont alors ignorées par le Séquentiel.

Un fichier Direct peut donc être utilisé en tant que fichier Séquentiel borné.

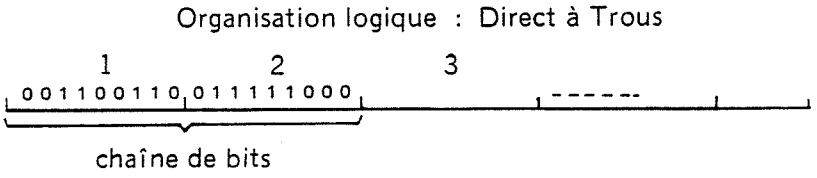
Remarque :

La requête DSEL permet également l'utilisation du fichier en séquentiel pur statique. Voir paragraphe 7.3.2 (f).

7.2.2 - Le Direct à Trous

Le Direct à Trous est une méthode d'accès capable de gérer des Unités d'Accès à des fichiers Directs (Numéro : 2). Le Direct à Trous est réentrant. La réentrance est opérationnelle pour des tâches différentes accédant à des fichiers Directs par des Unités d'Accès (Direct à Trous) différentes.
Le Direct à Trous est un **sur-ensemble** du Direct. Il possède toutes les fonctions du Direct, et possède en plus, 2 requêtes au niveau article : DCRE ,DSUP, permettant de gérer l'existence des articles et de placer «en vrac» des articles dans un fichier Direct.

a) **Organisation logique**
Le DIT utilise un complément d'organisation logique par rapport au Direct.



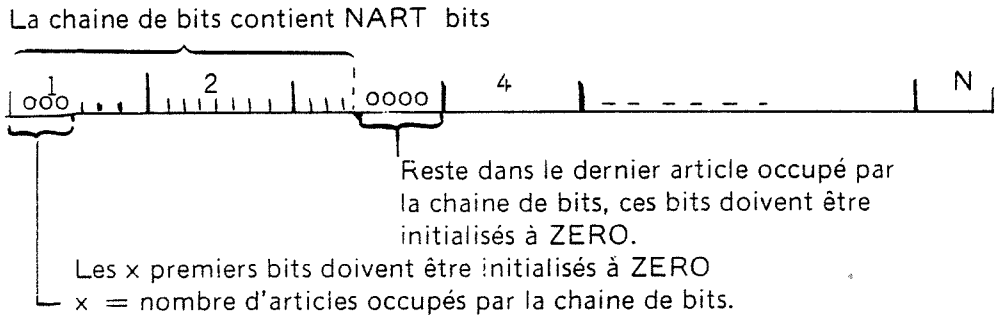
En plus de l'organisation logique directe le DIT utilise dans les premiers articles du fichier une chaine de bits permettant de gérer l'allocation et la désallocation des articles dans le fichier.

Remarques : Le terme article sera utilisé avec 2 sens différents :
— contenu, au sens créer ou supprimer un article
— contenant, au sens d'une place libre ou occupée.

De façon bijective, au bit de rang i correspond l'article de numéro i

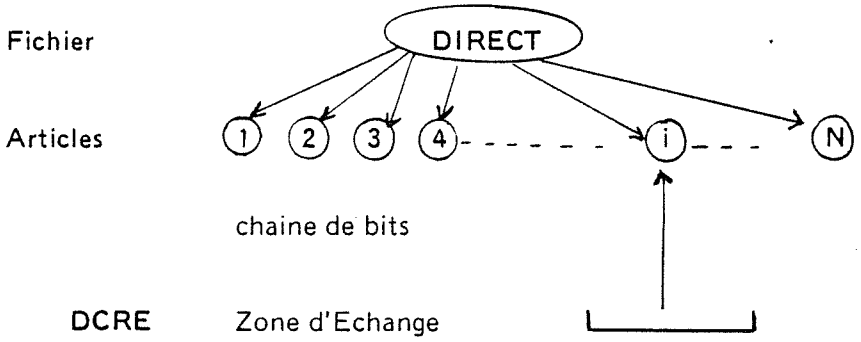
La création de la chaine de bits est à la charge de l'utilisateur, après la création du fichier (DIRECT).

Règles :
— Bit = 0 ⟺ Place occupée
— Schéma de l'état initial.



b) **Accès aux articles.**
En plus des **3 modes** de fonctionnement du Direct, le DIT fournit2 requêtes d'accès aux articles : DCRE , DSUP.

Accès aux articles.



La requête DCRE permet de créer un article, au sens allouer une place et initialiser son contenu. Le DIT rend à l'utilisateur le numéro de l'article dans lequel il a réécrit le contenu de la zone d'échange. C'est donc FMS qui définit l'identificateur logique (le numéro) d'un article.
La requête DSUP permet de supprimer l'article i, c'est à dire positionner le bit de rang i à 1.

Le DIT constitué de l'ensemble des 4 requêtes du niveau article : DREAD, DWRITE, DCRE, DSUP ne gère pas complètement l'existence des articles puisque les requêtes DREAD et DWRITE fonctionnent comme dans le Direct.
L'utilisateur doit donc protéger lui-même l'accès à la chaine de bits.

- DREAD de l'article 1 lit le premier article de la chaine de bits.
- DWRITE de l'article 1 réécrit le premier article de la chaine de bits.
- DSUP de l'article 1 libère également le premier article de la chaine de bits qui pourra être réalloué par une requête DCRE.

7.3 - FONCTIONS LOGIQUES.

7.3.1 - Le niveau Fichier.

a) **Généralités**
Les requêtes du niveau fichier dans les cas Direct et Direct à Trous sont semblables à celles des autres méthodes d'accès. Leur fonctionnement est décrit dans le chapitre 4 - L'Entité Fichier. Il n'est décrit ici que les particularités liées à la méthode d'accès Direct. Elles ne concernent que les requêtes : CREAT, OPEN NEW.
Ces 2 primitives fonctionnent de façon identique avec le Direct et le Direct à Trous.

L'Interface d'appel d'une requête du niveau fichier est en PL 1600 :

```
RA := @FCB ;
SVC   (FMS) ;        où   FMS = '38
```

b) CREAT, OPEN NEW : **Création d'un fichier Direct**
Les paramètres spécifiques de la création d'un fichier Direct sont :
— Organisation logique numéro 2
— La taille des articles : TART
— Le nombre d'articles : NART

FMS alloue à la création la place nécessaire à tout le fichier sur le support spécifié par l'utilisateur.
Un fichier direct est créé avec l'organisation physique directe. Il ne doit pas occuper plus de 128 granules, soit :
Relation à vérifier : les termes sont exprimés en mots :

$1 \leq$ Volume du fichier $=$ (Taille du secteur) x (Taille utile du granule) $\leq$ 4 177 920 mots.

Volume du fichier = (TART en octets/2) x NART

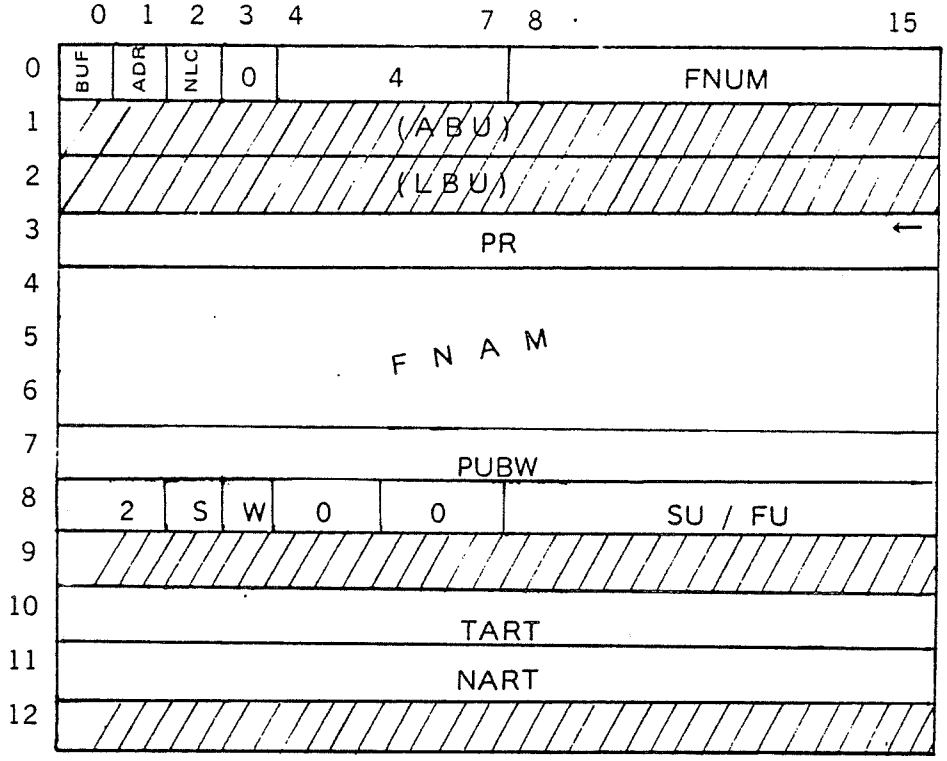
Taille du secteur = 128 mots.
Taille utile du granule = (Taille du granule définie sur le support utilisé) — (Taille du secteur).
Pour plus de précision se reporter au Manuel d'Utilisation de FMS.

La création d'un fichier Direct (CREAT, OPEN NEW) positionne le pointeur courant au début du fichier.

Nom	CREAT	Direct
But	Créer un fichier Permanent Direct et créer une Unité d'Accès à ce fichier.	

Appel RA := @ FCB ; SVC (FMS) ; où FMS = '38

FCB



Description des paramètres

TART = nombre pair d'octets tel que $2 \leqslant TART \leqslant 65\,534$
NART = nombre d'articles tel que $1 \leqslant NART \leqslant 65\,535$

Comptes rendus

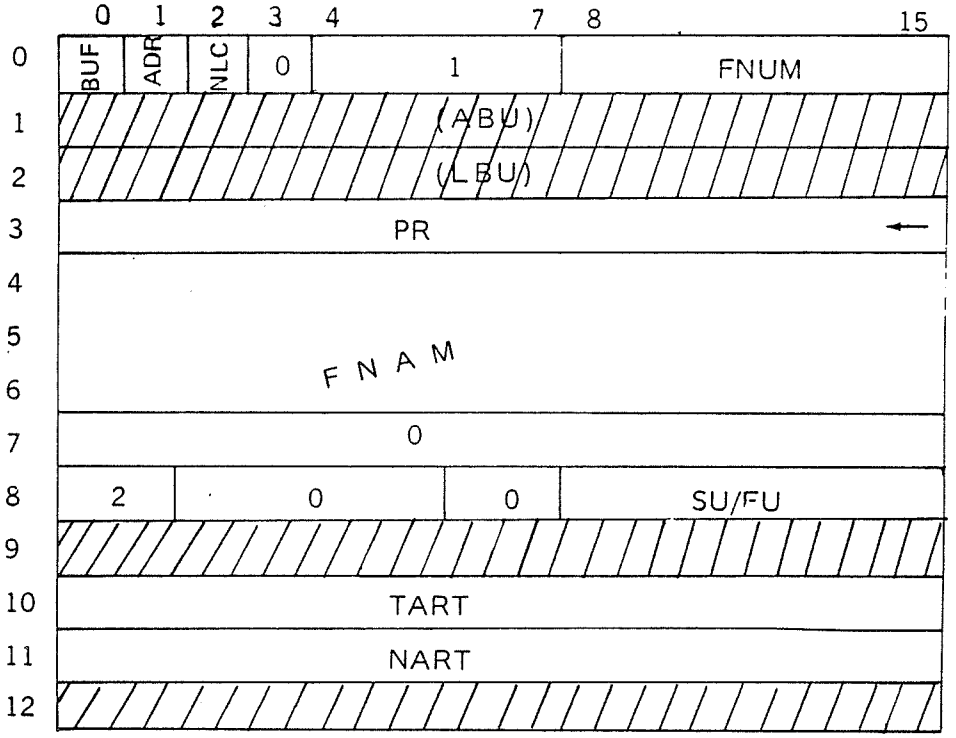
Valeur de PR	Signification
0	**Primitive exécutée correctement.**
'600B	**FAU existante.**
'600D	**Fichier existant.**
'6017	**Fichier trop long :** le fichier ne tient pas sur 128 granules, compte tenu de la taille du fichier et de la taille des granules définie sur le support désigné par SU/FU. La primitive est ineffective.
'601F	**Primitive en cours.**
'6020	**Zone de Pavés saturée.**
'6021	**FU saturée :** le nombre de granules libres sur le support désigné par SU/FU ne permet pas d'allouer la place nécessaire à tout le fichier. La primitive est ineffective.
'6022	**Table des Fichiers saturée.**
'6028	**Erreur de Syntaxe.** (@ FCB, FONCT, FNAM, PUBW, FTYP, TART, NART)
'6029	**Adresse de FCB incorrecte.**
'602A	**SU ou FU non gérée par FMS.**
'602B	**Méthode d'accès non gérée.**

Erreurs Graves	'6032 '6033 '6034	Informations Système Invalides.
	'6035	FU verrouillée par IOCS.
	'4.....	Erreur hardware : '4000 + Mot d'état PU.

Nom	OPEN NEW	Direct
But	Créer un fichier Temporaire Direct et créer une Unité d'Accès à ce fichier.	

Appel RA := @FCB ; SVC (FMS) ; où FMS = `38

FCB



Description des paramètres

TART = nombre pair d'octets tel que : 2 ⩽ TART ⩽ 65 534
NART = nombre d'articles tel que : 1 ⩽ NART ⩽ 65 535

Comptes rendus	Valeur de PR	Signification
	0	**Primitive exécutée correctement.**
	`600B	**FAU existante.**
	`600D	**Fichier existant.**
	`6017	**Fichier trop long** : Le fichier ne tient pas sur 128 granules compte tenu de la taille du fichier et de la taille des granules définie sur le support désigné par SU/FU. La primitive est ineffective.
	`601F	**Primitive en cours.**
	`6020	**Zone de Pavés saturée.**
	`6021	**FU saturée** : Le nombre de granules libres sur le support désigné par SU/FU ne permet pas d'allouer la place nécessaire à tout le fichier. La primitive est ineffective.
	`6028	**Erreur de syntaxe.** (@FCB, FONCT, FNAM, FTYP, TART, NART)
	`6029	**Adresse de FCB incorrecte.**
	`602A	**SU ou FU non gérée.**
	`602B	**Méthode d'accès non gérée par FMS.**

Erreurs Graves	`6032	Informations Système Invalides.
	`6033	
	`6034	
	`6035	FU verrouillée par IOCS.
	`4.....	Erreur hardware : `4000 – Mot d'Etat PU.

7.3.2 - Le niveau Article.

a) Généralités

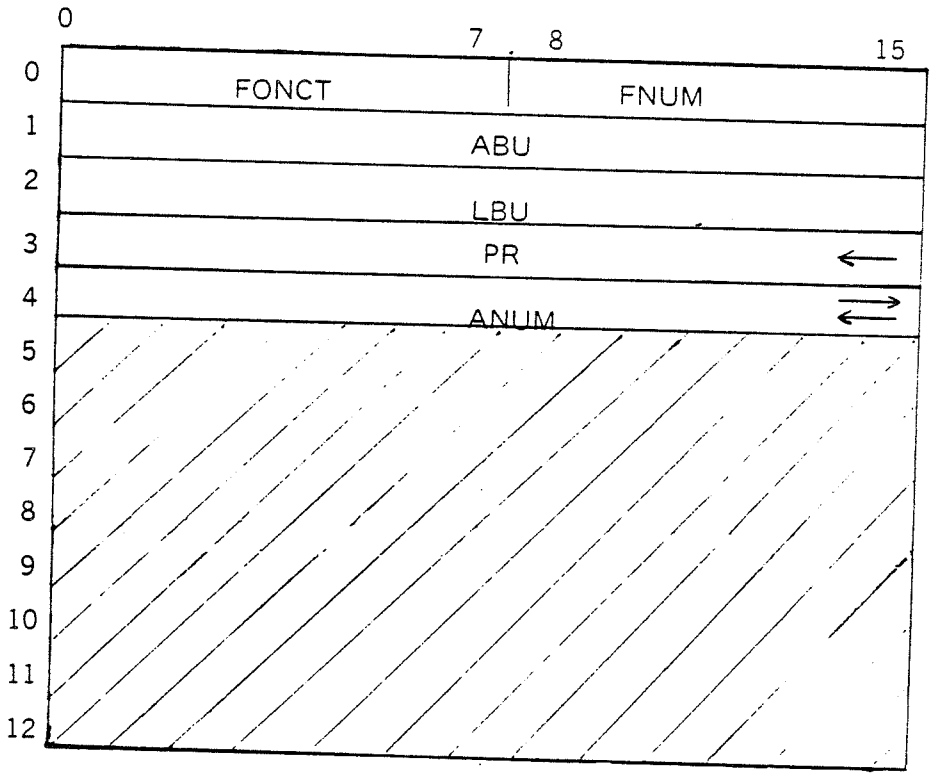
La méthode d'accès : Direct fournit les primitives : DREAD DWRITE.
La méthode d'accès : Direct à Trous fournit les primitives : DREAD, DWRITE, DCRE, DSUP.

L'interface d'appel d'une requête du niveau article pour les méthodes d'accès Direct et Direct à Trous est en PL 1600 :

RA := @ FCB ;
SVC (FMSD) ; où FMSD = '3B.

Description générale des FCB :
Direct, Direct à Trous.

FCB



Conventions :

- Les hachures signifient : zone non utilisée par FMS
- ← paramètres fournis par FMS
- ⇄ paramètre fourni par l'utilisateur et modifié par FMS

Toutes les primitives n'utilisent pas le mot 4 (ANUM) de la même façon, ceci n'est qu'un exemple de convention.

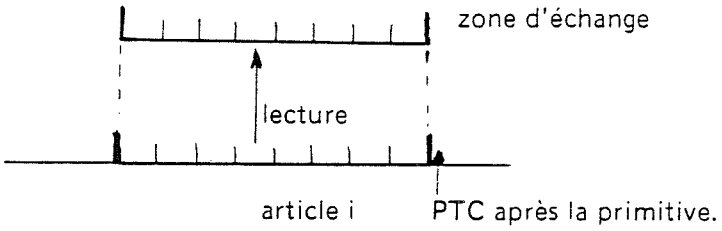
— Dans les autres cas le paramètre doit être fourni par l'utilisateur.

Toute requête du niveau article s'adresse à l'unité d'accès spécifiée par l'utilisateur (FNUM) et concerne le fichier accessible par celle-ci.
Toute requête du niveau article interdit l'accès séquentiel pur sur cette unité d'accès, et ceci jusqu'à sa destruction. Le choix d'une primitive parmi celles offertes par la méthode d'accès est fait par l'usager à l'aide de l'octet de fonction : FONCT.

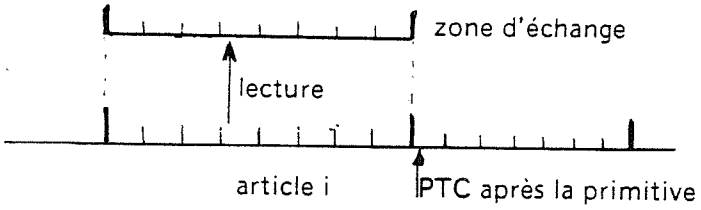
b) DREAD : **Lecture d'article.**
Cette primitive permet de lire un article désigné par son numéro (ANUM) dans la zone d'échange spécifiée par l'utilisateur. L'article est cadré à gauche dans la zone d'échange.

La primitive DREAD sélectionne l'article en vue d'un accès séquentiel à la portion d'article statique dans les cas suivants :

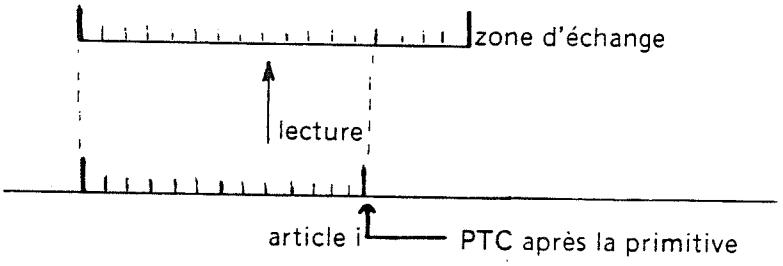
PR = 0 : Primitive exécutée correctement.



PR = `6003 : Article du fichier plus long que la zone d'échange.



PR = `6004 : Article du fichier plus court que la zone d'échange.

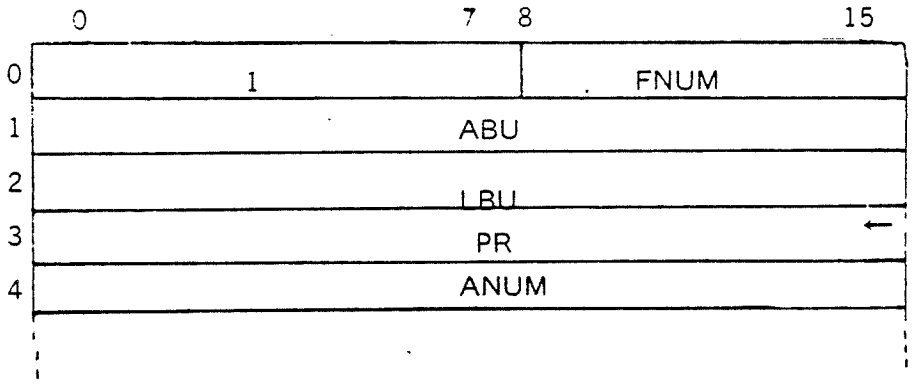


Pour les autres valeurs de PR, tout accès séquentiel est interdit sur cette Unité d'Accès (FNUM) jusqu'à la prochaine sélection valide.

Nom :	DREAD
But :	Lire un article d'un fichier Direct.

Appel RA := @FCB ; SVC (FMSD) ; où FMSD = 'B

FCB



Description des paramètres

FNUM	:	numéro de l'unité d'accès au fichier.
ABU	:	adresse de la zone d'échange utilisateur qui recevra l'article à lire.
LBU	:	taille de la zone d'échange en octets (apairée par FMS)
ANUM	:	numéro de l'article (1 ≤ ANUM ≤ NART donné à la création du fichier)
PR	:	paramètre de retour.

Comptes rendus

Valeur de PR	Signification
0	**Primitive exécutée correctement**
'6003	**Article du fichier plus long :** L'article du fichier est plus long que la zone d'échange spécifiée dans le FCB. La zone d'échange entière est remplie avec le début de l'article.
'6004	**Article du fichier plus court :** L'article du fichier est plus court que la zone d'échange spécifiée dans le FCB. Tout l'article est placé dans la zone d'échange. Il est cadré à gauche, le reste de la zone d'échange est inchangé.
'600A	**FAU inexistante.**
'600E	**Article inexistant :** le numéro de l'article spécifié n'appartient pas à l'ensemble défini à la création du fichier [1, NART]. La primitive est ineffective. Voir chapitre 7.3.3 § (c)
'6018	**Incompatibilité primitive - fichier.**
'601F	**Primitive en cours.**
'6028	**Erreur de syntaxe.** (@FCB, FONCT, ABU, LBU)
'6029	**Adresse de FCB invalide**
'602B	**Méthode d'Accès non gérée.**

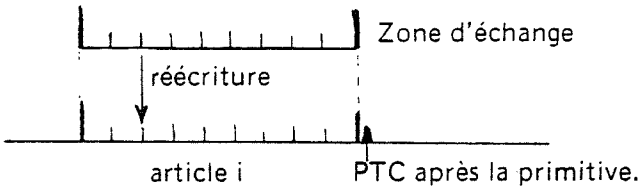
Erreurs Graves	`6032	Informations Système Invalides.
	`6034	
	`6035	FU verrouillée par IOCS.
	`4.....	Erreur hardware : `4000 – mot d'état PU.

c) DWRITE : **Ecriture d'article**

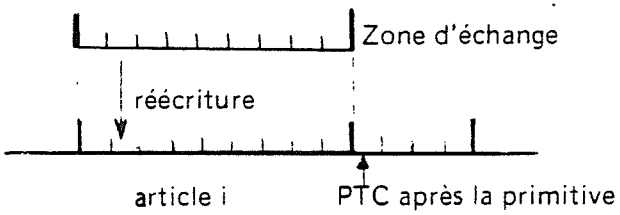
Cette primitive permet de réécrire un article, en transférant le contenu de la zone d'échange spécifiée par l'utilisateur dans l'article désigné par son numéro (ANUM). Le contenu de la zone d'échange est cadré à gauche dans l'article.

La primitive DWRITE sélectionne l'article en vue d'un accès séquentiel séquentiel, à portion d'article statique dans les cas suivants :

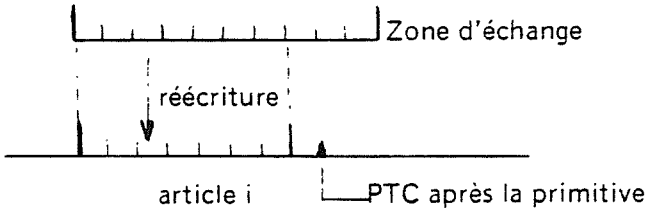
PR = 0 : Primitive exécutée correctement.



PR = `6003 : Article du fichier plus long que la zone d'échange.



PR = `6004 : Article du fichier plus court que la zone d'échange.

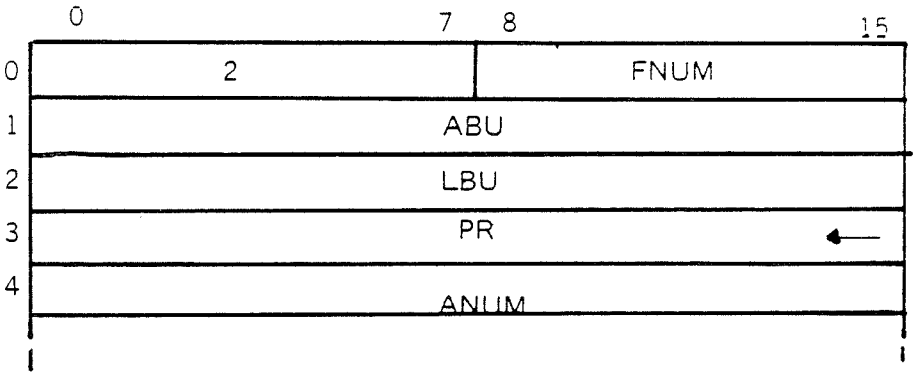


Pour les autres valeurs de PR tout accès séquentiel est interdit sur cette unité d'accès (FNUM) jusqu'à la prochaine sélection valide.

Nom	DWRITE
But	Ecrire un article d'un fichier Direct

Appel : RA : = @ FCB ; SVC (FMSD) ; ou FMSD = 3B

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.
ABU : adresse de la zone d'échange utilisateur qui contient l'article à écrire
LBU : longueur de la zone d'échange en octets (apairé par FMS)
ANUM : numéro de l'article (1 ⩽ ANUM ⩽ NART donné à la création du fichier)
PR : paramètre de retour.

Comptes rendus

Valeur de PR	Signification
0	**Primitive exécutée correctement**
6003	**Article du fichier plus long :** L'article du fichier est plus long que la zone d'échange spécifiée dans le FCB. Toute la zone d'échange est transférée dans l'article, cadrée à gauche. La fin de l'article est laissée inchangée.
6004	**Article du fichier plus court :** L'article du fichier est plus court que la zone d'échange spécifiée dans le FCB. L'article entier est réécrit avec le début de la zone d'échange.
600A	**FAU inexistante.**
600E	**Article inexistant :** Le numéro de l'article spécifié n'appartient pas à l'ensemble défini à la création du fichier [1,NART]. La primitive est ineffective. Voir chapitre 7.3.3 § (c).
6014	**Protection écriture.** Voir chapitre 7.3.3 § (e)
6018	**Incompatibilité Primitive - Fichier**
601F	**Primitive en cours.**
6028	**Erreur de syntaxe.** (@FCB, FONCT, ABU, LBU)
6029	**Adresse de FCB invalide**
602B	**Méthode d'accès non gérée**

Erreurs Graves	`6032	Informations Système Invalides.
	`6034	
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU.

d) DCRE : **Création d'un article.**

Cette primitive n'appartenant qu'au Direct à Trous, permet de créer un article contenu dans la zone d'échange spécifiée dans le FCB. Le DIT alloue une place libre pour ce nouvel article, y écrit son contenu et rend à l'utilisateur dans le FCB le numéro de l'article ainsi défini (ANUM). Le numéro de l'article est donc identique à celui de la place dans laquelle il a été alloué.

Pour allouer une place libre, le DIT utilise la chaine de bits créée par l'utilisateur dans les premiers articles du fichier. Il effectue une recherche séquentielle d'un bit à 1, en chargeant en mémoire centrale la chaine de bits séquentiellement secteur par secteur dans une zone système (ZUEP).

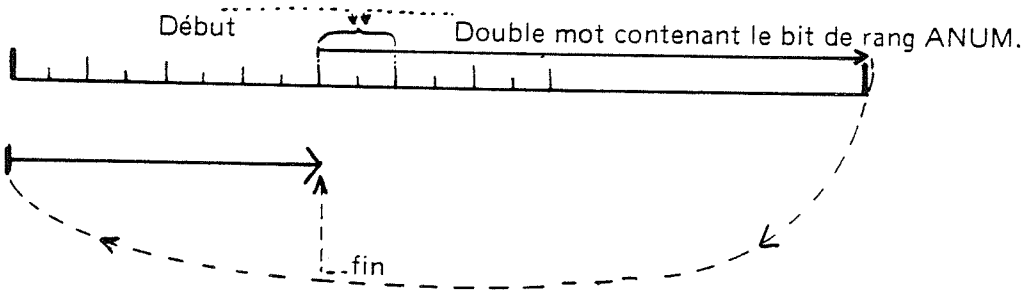
Pour accelérer l'exécution de la primitive DCRE, il est fourni à l'utilisateur la possibilité de spécifier à l'aide d'un numéro d'article (ANUM) dans le FCB, un point de départ pour la recherche séquentielle.. Pour que la recherche d'une place libre s'effectue à partir du début de la chaine de bits, l'utilisateur doit fournir en paramètre de la primitive DCRE un numéro d'article (ANUM) égal à 1.

— Une suite de primitives DCRE, avec un FCB dans lequel n'aura varié que le ANUM (modifié par FMS) revient à allouer séquentiellement les articles dans le fichier.

Remarques sur le traitement de la chaine de bits :

— **Début de Recherche** : Par souci de simplicité, la recherche d'un bit à 1 ne commence pas exactement au bit, dont le rang (ANUM) est spécifié dans le FCB. Elle débute toujours à une frontière de double-mots.

— **Cycle de recherche** :

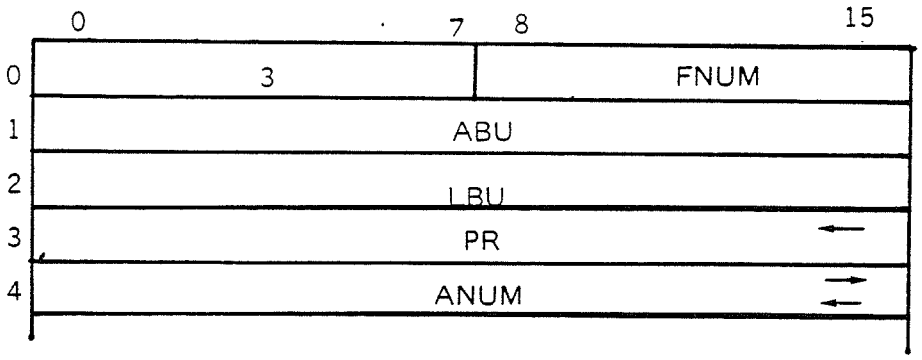


La primitive DCRE sélectionne l'article en vue d'un accès séquentiel à la portion d'article statique. Cette sélection fonctionne comme celle de la primitive DWRITE (chapitre 7.3.2.5 (c)).

Nom	DCRE
But	Créer un article et son numéro d'identification.

Appel : RA := @ FCB ; SVC (FMSD) ou FMSD = 'BB

FCB



Description des paramètres

FNUM : Numéro de l'unité d'accès au fichier
ABU : adresse de la zone d'échange qui contient l'article à créer.
LBU : longueur de la zone d'échange, en octets (apairée par FMS)
ANUM : **en entrée :** numéro spécifiant le début de la recherche séquentielle d'une place libre.
$1 \leqslant ANUM \leqslant NART \leqslant 65\,535$
en sortie : numero de la place trouvée pour créer l'article et qui devient donc le numéro de l'article.

Comptes rendus

Valeur de PR	Signification
0	**Primitive exécutée correctement.**
'6003	**Article du fichier plus long :** l'article du fichier est plus long que la zone d'échange spécifiée dans le FCB. Toute la zone d'échange est transférée dans l'article, cadrée à gauche. La fin de l'article est laissée inchangée.
'6004	**Article du fichier plus court :** l'article du fichier est plus court que la zone d'échange spécifiée dans le FCB. l'article entier est écrit avec le début de la zone d'échange.
'600A	**FAU inexistante.**
'600E	**Article inexistant :** le numéro (ANUM) spécifiant le début de la recherche séquentielle d'une place libre est invalide par rapport au paramètre NART défini à la création du fichier. La primitive est ineffective. Voir chapitre 7.3.3 § (c).
'6014	**Protection écriture.** Voir chapitre 7.3.3 § (c)
'6016	**Fichier saturé.** Il n'y a plus de place libre dans le fichier Direct à Trous. La primitive est ineffective.
'6018	**Incompatibilité primitive - fichier.**
'601F	**Primitive en cours.**
'6028	**Erreur de syntaxe.** (@FCB, FONCT, ABU, LBU)
'6029	**Adresse de FCB invalide.**
'602B	**Méthode d'accès non gérée.**

Erreurs Graves

`6032, `6034	Informations Systèmes Invalides.
`6035	FU verrouillée par IOCS.
`4 . . .	Erreur hardware : `4000 + mot d'état PU.

e) DSUP : **Suppression d'un article.**

Cette primitive n'appartenant qu'au Direct a Trous, permet de supprimer un article qui a été créé par la primitive DCRE.

L'article à supprimer est désigné par son numéro (ANUM). FMS le supprime en positionnant le bit de rang ANUM à 1.

Dans le cas ou l'article est marqué dans la chaine de bits comme un existant, FMS considere que la primitive DSUP est correctement exécutée.

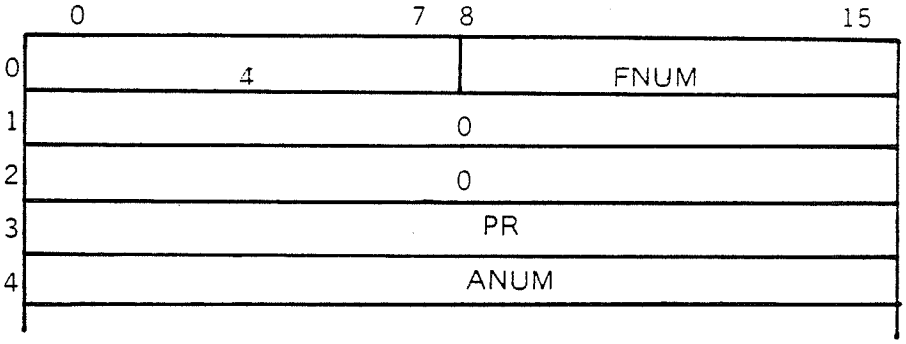
La chaine de bits n'est pas protégée par une primitive DSUP indiquant comme ANUM un article occupé par la chaine de bits.

La primitive DSUP n'effectue pas de sélection d'article, elle interdit donc tout accès sequentiel sur l'unité d'accès (FNUM) jusqu'à la prochaine sélection valide.

Nom	DSUP
But	Supprimer un article.

Appel : RA := @ FCB ; SVC (FMS D) ; où FMS D = `3B

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier.

ANUM : numéro de l'article à supprimer.
1 ⩽ ANUM ⩽ NART ⩽ 65 535

Comptes rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée.**
`600A	**FAU inexistante.**
`600E	**Article inexistant :** le numéro de l'article à supprimer est invalide. La primitive est ineffective.
`6014	**Protection écriture.**
`6018	**Incompatibilité Primitive - Fichier.**
`601F	**Primitive en cours.**
`6028	**Erreur de syntaxe :** (@FCB, FONCT)
`6029	**Adresse de FCB invalide.**
`602B	**Méthode d'accès non gérée.**

Erreurs graves

`6032	
`6034	**Informations système invalides**
`6035	**FU verrouillée par IOCS**
`4...	**Erreur hardware : `4000 mot d'état PU.**

F) DSEL : Sélection d'article

Cette requête est incluse dans le direct de FMS ainsi que dans le Direct à Trous.

Cette requête permet de sélectionner un article par son numéro ANUM, en positionnant le pointeur courant de la FAU correspondante au début de l'article. Cette requête permet ensuite d'utiliser les requêtes séquentielles pour réaliser un accès séquentiel pur statique sur tout le fichier à partir du pointeur courant. L'accès séquentiel qui en résulte n'est donc pas limité aux frontières d'articles

Nom	DSEL
But	Sélection d'article

Appel RA = @FCB ; SVC (FMSD) ; où FMSD = `3B

FCB

	0	7 8	15
0	5	FNUM	
1	0		
2	0		
3	PR		
4	ANUM		

Description des paramètres ANUM, Numéro de l'article à sélectionner. Le pointeur courant adresse le 1er mot de l'article.

Comptes rendus	Valeur de PR	Signification
	0	Primitive correctement éxécutée
	`600A	FAU inexistante
	`600E	Article inexistant : le numéro de l'article n'appartient pas à la plage définie à la création du fichier (1, NART). La requête est ineffective.
	`6018	Incompatibilité primitive fichier
	`601F	Primitive en cours
	`6028	Erreur de syntaxe (@FCB, FONCT)
	`6029	Adresse de FCB invalide
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032	Informations systèmes invalides
	`6034	
	`6035	FU Verrouillée par IOCS
	`4.....	Erreur hardware : `4000 mot d'état PU

G) Accès avec bufférisation

Les méthodes d'accès direct (DIRBUF) et direct à trous (DITFMS) de FMS permettent de bufférriser les entrées et les sorties lorsqu'un buffer a été fourni par l'une des requêtes CREAT, OPEN NEW, OPEN OLD.

Le buffer est fourni comme pour l'indexé par le booléen BUF et la définition du buffer ABU, LBU.

De plus, après les requêtes CREAT, OPEN NEW, OPEN OLD la bufférisation fonctionne à partir du début du fichier.

7.3.3 - Le niveau Portion d'Article.

a) **Portion d'Article Statique.**

L'accès séquentiel à un article selon le mode Portion d'Article Statique (P.A.S.) est initialisé de façon identique par les requêtes.

- DREAD avec PR = [0, '6003, '6004].
- DWRITE avec PR = [0, '6003, '6004].
- DCRE avec PR = [0, '6003, '6004].

Exemple : Acces au dernier mot de l'article numero 4 et modification de l'avant dernier mot de l'article 5.

1) Ouverture du fichier (Permanent)

```
OPEN OLD   (1, FIC - FM, D3 - - -      )
        FMS  ⇒  PR  = 0
```

2) Selection de l'article 4

```
DREAD    (1, ABU , LBU  = 0 , PR, 4)
        FMS  ⇒  PR  =   '6003
```

3) Positionnement sur le dernier mot de l'article en cours (4)

```
SKEOA    (1, PR)
        FMS  ⇒  PR  =   0

SKIPB (1, CA = 0, LBU = 2, PR)
        FMS  ⇒  PR  =   2
```

4) Lecture du dernier mot de l'article 4

```
READ (1, ABU, LBU = 2, PR)
        FMS  ⇒  PR  =   2
```

5) Sélection de l'article numéro 5.

```
DREAD  (1, ABU, LBU = 0, PR, 5)
        FMS  ⇒  PR  =   '6003
```

6) Positionnement sur l'avant dernier mot de l'article en cours (5).

```
SKEOA  (1, PR)
        FMS  ⇒  PR  =   0
```

```
SKIPB (1, CA = 0, LBU = 4, PR)
    FMS  ⇒  PR  =  4
```

7) Réécriture de l'avant dernier mot de l'article en cours (5). La zone d'échange contient la nouvelle valeur, le reste de l'article est inchangé.

```
WRITE (1, ABU, LBU  = 2, PR)
    FMS  ⇒  PR  =  2
```

8) Fermeture du fichier.

```
CLOSE (1, PR)
    FMS  ⇒  PR  =  0
```

b) Séquentiel pur statique.
L'accès séquentiel a tout le fichier considéré comme mono-article, selon le mode séquentiel Pur Statique est permis après l'une des 3 requêtes :

- OPEN NEW avec PR = 0
- OPEN OLD avec PR = 0
- CREAT avec PR = 0
- DSEL " " "

Exemple : Initialisation à zéro du contenu d'un fichier Direct après sa création. La structure du fichier en articles n'est pas utilisée. L'initialisation se fait à l'aide d'une zone d'échange de longueur 1 secteur mise à zéro.

1) Création du fichier de nom FIC, mot de passe public FM, sur le support D 3.

```
CREAT (1, FIC - FM, D3 - - - - )
    FMS  ⇒  PR  =  0
```

2) Mise à zéro

n - 1 fois

```
WRITE(1,ABU,LBU = 256,PR)
    FMS  ⇒ PR = 256
```

la n^{e} fois
La longueur du fichier = TART x NART.
Si TART x NART = (n - 1) 128 mots + 80 mots

```
FMS  ⇒ PR = 160
```

la n + 1^{e} fois

```
FMS  ⇒ PR = 6001
```

3) Fermeture du fichier

```
CLOSE (1, PR)
    FMS  ⇒ PR = 0
```

c) **La sélection d'article et les comptes rendus.**

La tableau suivant indique les exceptions de fonctionnement de la sélection d'article pour les requêtes des méthodes d'accès Direct et Direct à Trous.

OUI : Selon la requête la sélection d'un article est réalisée. Dans tous les cas la sélection précédente est terminée.

NON : La primitive est ineffective. La précédente sélection peut continuer.

NSEL : La précédente sélection est terminée. Aucune sélection n'est réalisée. Donc, tout accès séquentiel au fichier est interdit par l'unité d'accès concernée, jusqu'à la prochaine sélection valide.

Le tableau suivant indique également la liste des comptes rendus de l'ensemble des requêtes du Direct et du Direct à Trous.

DIR et DIT Valeur de PR	Signification	Sélection
0	Primitive correctement exécutée	OUI
`6003	Article du fichier plus long	OUI
`6004	Article du fichier plus court	OUI
`600A	FAU inexistante	NON
`600E	Article inexistant	NSEL
`6014	Protection écriture	NSEL
`6016	Fichier Saturé	NSEL
`6018	Incompatibilité primitive - Fichier	NON
`601F	Primitive en cours	NON
`6028	Erreur de syntaxe @ FCB	NSEL
	FONCT	NSEL
	ABU	NSEL
	LBU	NSEL
`6029	Adresse de FCB invalide	NON
`602B	Méthode d'accès non gérée	NON
`6032	Informations Système Invalides	NSEL
`6034	"	"
`6035	FU Verrouillée par IOCS	"
`4.....	Erreur hardware	"

7.3.4 - L'interface 2000 Mega-articles

La méthode d'accès Direct à Trous permet de gérer des fichiers Direct ou Direct à trous de :

- 64 K articles avec le Noyau standard ou le noyau rapide (KERADR).

— 2000 Méga-articles (théorique) avec le Noyau de gestion des grands disques de FMS (KERGDI).

Règles .

2000 Méga-articles

— TART est un nombre pair d'octets de [1 à 65534]
— NART est un nombre d'article de [1 à 2^{31}]
— La taille Totale du fichier est limitée :
  1°) à 2^{31} mots
  2°) à 128 fois la taille utile du granule.

L'interface 2000 Méga-articles est réalisé à l'aide de :

— 1 booleen NAR (bit 3 de FONCT) qui valide la présence des poids forts de NART ou ANUM

— Les poids forts (15 bits) sont situés dans le mot qui suit NART ou ANUM.

Remarque :

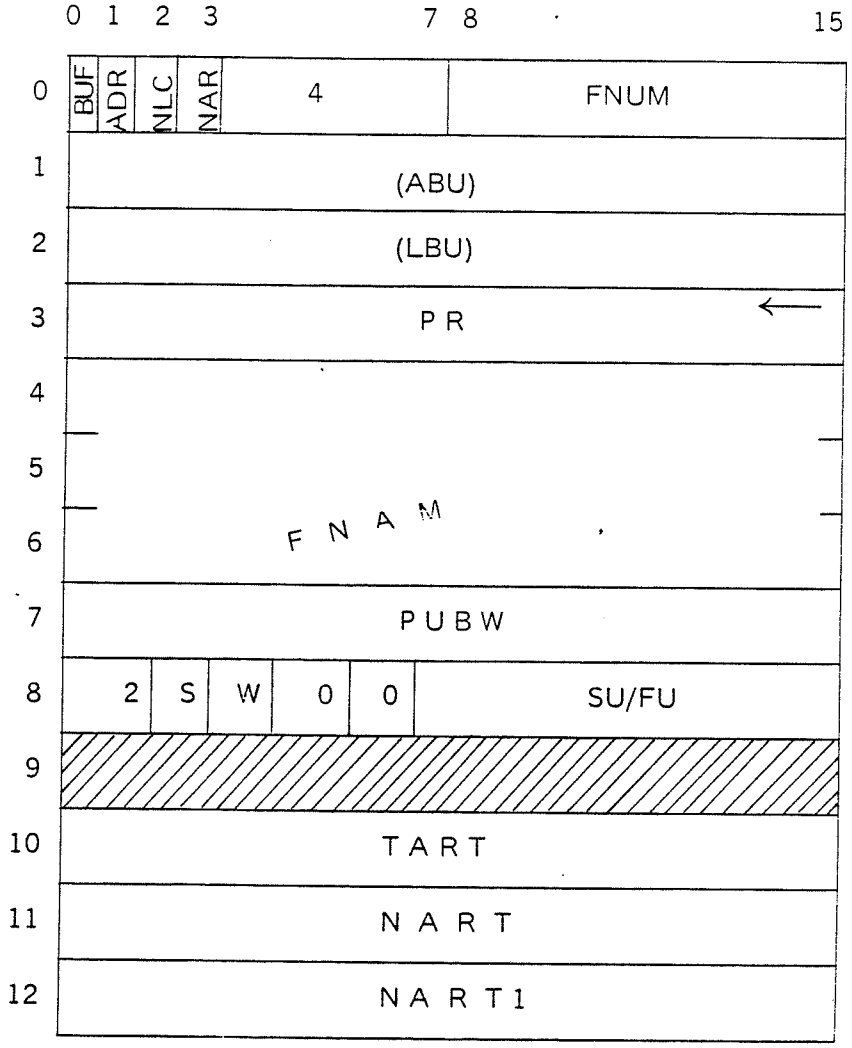
L'interface 2000 Méga-article est également géré par la méthode d'accès direct (DIRBUF) fin de pouvoir réaliser une programmation indépendante du système.
Bien entendu seule une valeur NART ou ANUM sur 1 mot est acceptée, même si ces deux modules sont associées avec le noyau de gestion des grands disques de FMS : KERGDI.

a)

Nom	CREAT
But	Créer un fichier permanent Direct et créer une Unité d'accès à ce fichier.

Appel RA : = @FCB ; SVC (FMS) ; ou FMS = ` 38

FCB



Déscription des paramètres

NAR : indique le codage de NART
NAR = 0 ⟹ NART est codé sur 1 mot
NAR= 1 ⟹ NART est codé sur 2 motsé les poids forts étant NART 1

TART : nombre pair d'octets tel que $2 \leqslant TART \leqslant 65534$
NART : nombre d'articles tel que $1 \leqslant NART < 2^{31}$

Remarque :

- Si KERGDI et — NAR = 1 traitement NART sur 2 mots
  — NAR = 0 traitement NART sur 1 mot
- Si KERADR et — NAR = 1 traitement NART sur 1 mot
  Si NART 1 = 0 sinon FICTLO
  — NAR = 0 traitement NART sur 1 mot

Comptes rendus :

Valeur de PR	Signification
0	primitive exécutée correctement
'600B	FAU existante
'600D	Fichier existant
'6017	Fichier trop long : le fichier ne tient pas sur 128 granules, compte tenu de la taille du fichier et de la taille des granules définie sur ' support désigné par SU/FU. NART1 non nul avec KER ADR. TART x NART ⩾ 2^{31}. La primitive est ineffective.
'601F	Primitive en cours
'6020	Zone de pavés saturée
'6021	Fu saturée : le nombre de granules libres sur le support désigné par SU/FU ne permet pas d'allouer la place nécessaire à tout le fichier. La primitive est innefective.
'6022	Table des fichiers saturée
'6028	Erreur de syntaxe (@FCB, FONCT, FNAM, PUBW, FTYP, TART, N
'6029	Adresse de FCB incorrecte
'602A	SU ou FU non gérée par FMS : Le noyau configuré dans FMS ne doit pas gérer des grands disques. La primitive est ineffective.
'602B	Méthode d'accès non gérée

Erreurs graves

'6032, '6033, '6034	Informations système Invalides
'6035	FU vérrouillée par IOCS
'4- - -	Erreur hardware : '4000 + mot d'état PU

b)

Nom	OPEN NEW
But	Créer un fichier temporaire direct et créer une unité d'accès à ce fichier

Appel RA := @FCB ; SVC (FMS) OU FMS = `38

FCB

	0 1 2 3 4	7 8 . 15
0	BUF ADR NLC NAR	1 FNUM
1	(ABU)	
2	(LBU)	
3	PR ←	
4		
5	FNAM	
6		
7	0	
8	2 0 0 SU/FU	
9		
10	TART	
11	NART	
12	NART1	

Description des paramètres Idem CREAT

Comptes rendus :

Valeur de PR	Signification
0	Primitive exécutée correctement
`600B	FAU existante

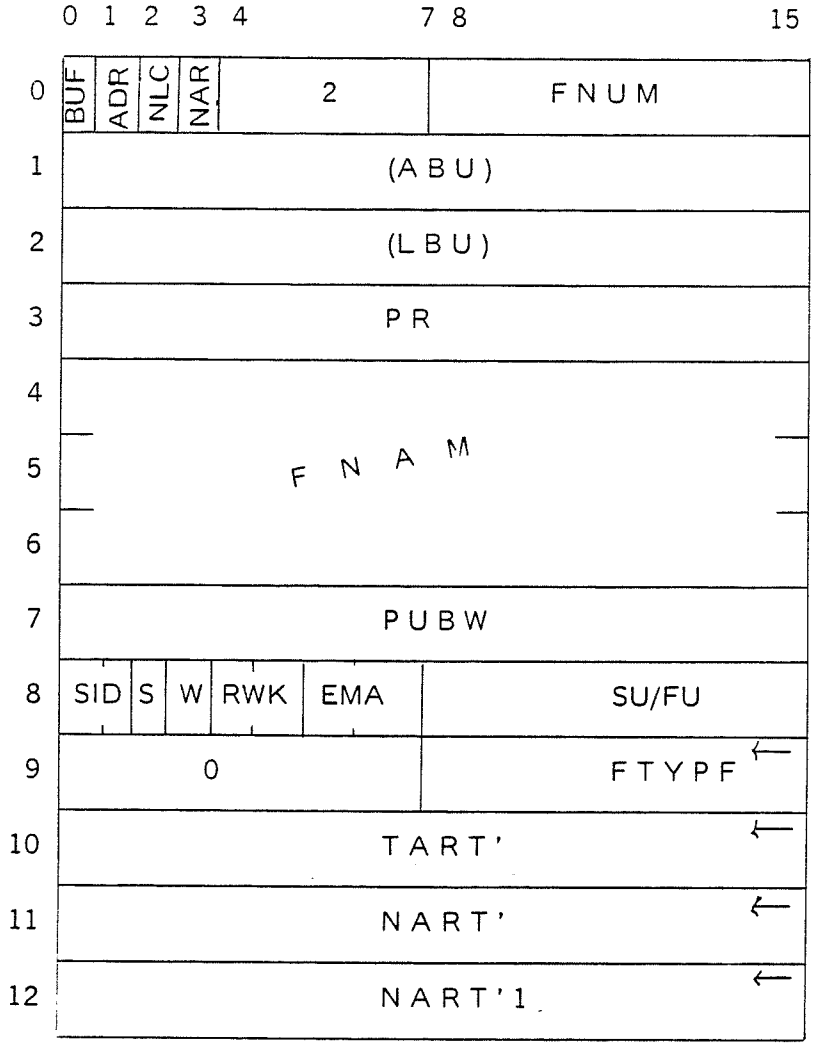
	`600D	**Fichier existant**
	`6017	**Fichier trop long** : le fichier ne tient pas sur 128 granules compte tenu de la taille du fichier et de la taille des granules définie sur le support désigné par SU/FU. NART 1 non nul avec KERADR ou TART x NART $\geqslant 2^{31}$ mots. La primitive est ineffective.
	`601F	**Primitive en cours**
	`6020	**Zone de pavés saturés**
	`6021	**FU saturée** : le nombre de granules libres sur le support désigné par SU/FU ne permet pas d'allouer la place nécessaire à tout le fichier. La primitive est ineffective.
	`6028	**Erreur de syntaxe** (@ FCB ,FONCT FNAM, FTYP, TART, NART)
	`6029	**Adresse de FCB incorrecte**
	`602A	**SU ou FU non gérée par FMS** : Le noyau configuré dans FMS ne sait pas gérer des grands disques. La primitive est ineffective.
	`602B	**Méthode d'accès non gérée par FMS**
Erreurs graves	`6032 `6033 `6034	**Informations Système Invalides**
	`6035	**FU verrouillée par IOCS**
	`4 ---	**Erreur hardware : `4000** – Mot d'état PU

c)

Nom	OPEN OLD
But	Ouvrir un fichier permanent et créer une unité d'accès à ce fichier

Appel : RA : = @ FCB ; SVC (FMS) ; ou FMS = '38

FCB



Ouverture d'un permanent en mode séquentiel (OPEN F)

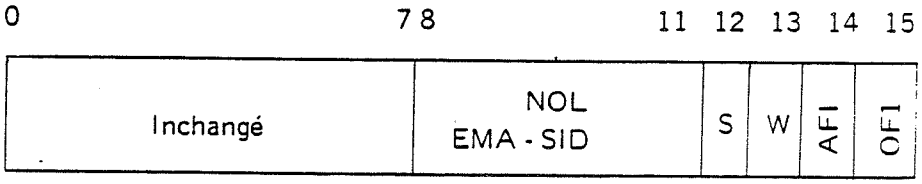
L'utilisateur charge le FCB comme pour OPEN OLD avec cependant :

NAR = 0 (ineffectif en entrée)
BUF = 0 si pas de buffer
EMASID = 'F

FMS rend à l'utilisateur :

1) NAR = 0 si FMS rend NART' sur 1 mot
= 1 si FMS rend NART' sur 2 mots
(NART' 1 = Poids Forts)

2) FTYPF :



3) TART' : TART' est exprimé en mot

4) NART' } : Le codage de NART est fonction de NAR
NART' 1 }

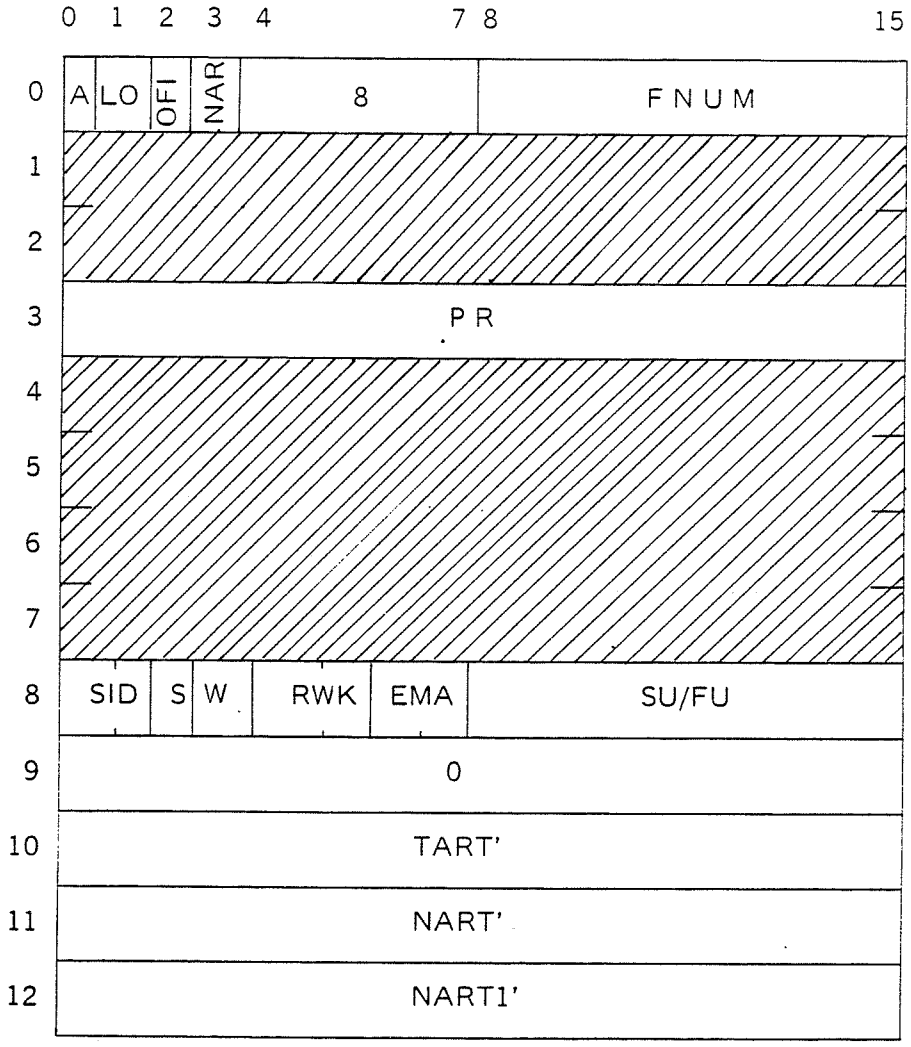
Comptes rendus	Valeur de PR	Signification
	idem	OPEN OLD (chapitre 4.2.2) sauf en plus :
	'602A	**SU ou FU non gérée par FMS :** Le noyau configuré dans FMS ne sait pas gérer des grands disques. La primitive est ineffective.

d)

Nom	ALTER
But	Modifier les attribus d'un fichier et W, RWK d'une FAU

Appel RA : = @ FCB ; SVC (FMS) ; ou FMS = `38

FCB



Description des paramètres :

A
= 0 Modification de S, W, RWK : ALTER SIMPLE
= 1 Modification totale (NOL, TART', NART', NART 1', OFI, S, W, RWK) ALTER TOTAL

LO = 1, modifications uniquement en mémoire centrale : ALTER LOGIQUE
OFI numéro d'organisation physique
OFI = 0 organisation séquentielle OFI sera remis à zéro
OFI = 1 organisation directe. Le traitement effectué sera équivalent à celui de FAST. Création d'une TLG et OFI : = 1

NAR indique le codage de NART
NAR = 0 NART codé sur 1 mot
NAR = 1 NART codé sur 2 mots (NART 1 = Poids Forts)

Remarque :

Si KER GDI et NAR = 1 traitement effectif avec NART sur 2 mots
NAR = 0 traitement effectif avec NART sur 1 mot

Si KER ADR et NAR = 1 traitement effectif avec NART sur 1 mot si NART1' = 0, sinon FICTI ○

NAR = 0 traitement effectif avec NART sur 1 mot

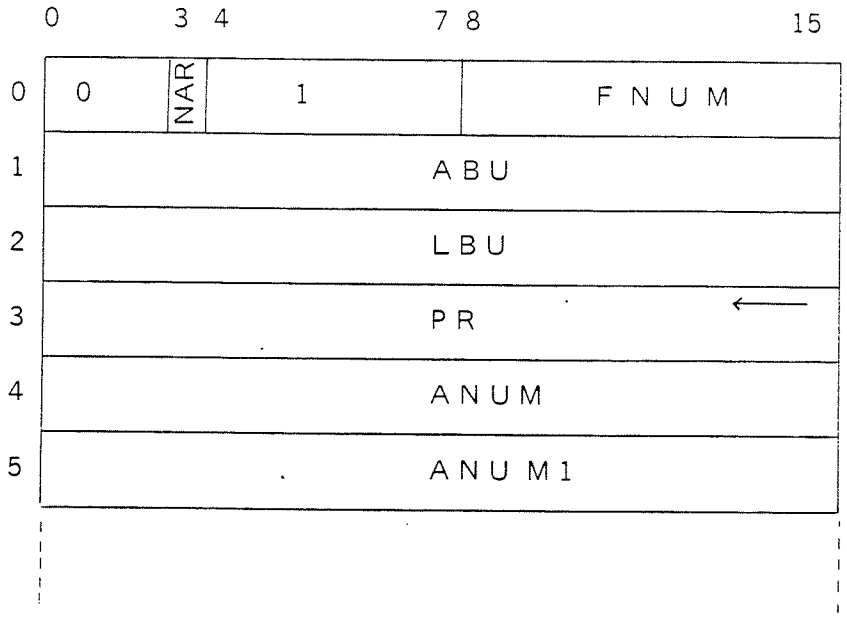
Compte rendu	Valeur de PR	Signification
	0	**Primitive correctement exécutée**
	'600A	**FAU inéxistante**
	'6018	**Incompatibilité primitive fichier** : le fichier n'est pas permanent la requête est ineffective
	'6014	**Protection écriture.** La requête est ineffective. Alter Total (A = 1) : la FAU n'est pas en écriture, ou le fichier est ouvert avec RWK = 0, ou les nouvelles RWK (du FCB) ne sont pas nulles.
	'6017	**Fichier trop long** : incompatibilité codage de NART et noyau
	'601E	**Fichier occupé** : La requête est ineffective. Alter Total (A = 1) le fichier doit être en mono-accès.
	'601F	**Primitive en cours**
	'6028	**Erreur de syntaxe** (@FCB, FONCT, EMA-SID, incompatibilité codage NART' et noyau)
	'6029	**Adresse de FCB invalide**
	'602A	**SU ou FU non gérée par FMS**
Erreurs graves		
	'6034	**Informations système invalides**
	'6035	**FU verrouillée par IOCS**
	'4 - - -	**Erreur hardware : '4000** + mot d'état PU

e)

Nom	DREAD
But	Lire un article d'un fichier Direct

Appel RA : = @FCB SVC (FMSD) ou FMSD = '3B

FCB



Description des paramètres

NAR : indique le codage de ANUM
NAR = 0 ⟹ ANUM est codé sur 1 mot
NAR = 1 ⟹ ANUM est codé sur 2 mots, ANUM 1 = P. Fort.

FNUM : numéro de l'unité d'accès au fichier

ABU : adresse de la zone d'échange utilisateur qui recevra l'article à lire

LBU : Taille de la zone d'échange en octets (apairée par FMS)

ANUM / ANUM 1 : numéro de l'article (1 ⩽ ANUM ⩽ NART donné à la création du fichier
ANUM 1 ⩽ NART1

PR : paramètre de retour

Remarque :

- Si KERGDI et NAR = 1 alors le traitement est effectif si ANUM, A... ⩽ NART, NART 1, sinon INART
  NAR = 0 alors traitement effectif si ANUM ⩽ NART, NART 1, sinon INART

- Si KERADR et NAR = 1 alors le traitement est effectif si ANUM 1 = 0, sinon INART
  et si ANUM ⩽ NART, sinon INART
  NAR = 0 alors traitement effectif si ANUM ⩽ NART, sinon INART

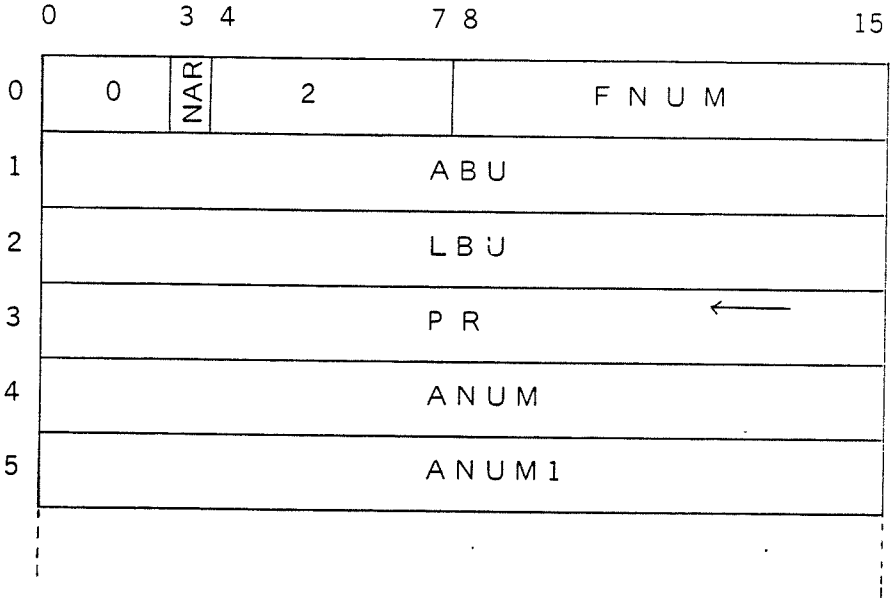
Compte Rendus	Valeur de PR	Signification
	0	**primitive exécutée correctement**
	'6003	**Article du fichier plus long.** L'article du fichier est plus long que la zone d'échange spécifiée dans le FCB. La zone d'échange entière est remplie avec le début de l'article.
	'6004	**Article du fichier plus court.** L'article du fichier est plus court que la zone d'échange spécifiée dans le FCB. Tout l'article est placé dans la zone d'échange. Il est cadré à gauche, le reste de la zone d'échange est inchangé.
	'600A	**FAU inéxistante**
	'600E	**Article inéxistant :** Le numéro de l'article spécifié n'appartient pas à l'ensemble défini à la création du fichier [1, NART (NART1)]. La primitive est ineffective.
	'6018	**Incompatibilité primitive - fichier.**
	'601F	**Primitive en cours**
	'6028	**Erreur de syntaxe** (@FCB, FONCT, ABU, LBU)
	'6029	**Adresse de FCB invalide**
	'602B	**Méthode d'accès non gérée**
Erreurs graves		
	'6032	**Informations système invalides**
	'6034	
	'6035	**FU verrouillée par FMS**
	'4- - -	**Erreur hardware : '4000** + mot d'état PU

f)

Nom	DWRITE
But	Ecrire un article d'un fichier direct

Appel RA := @ FCB SVC (FMS) ou FMSD = '3B

FCB



Description des paramètres

NAR : indique le codage de ANUM
NAR = 0 ⟹ codage de ANUM sur 1 mot
NAR = 1 ⟹ codage de ANUM sur 2 mots (ANUM1 = P. Forts)

FNUM : numéro de l'unité d'accès au fichier

ABU : adresse de la zone d'échange utilisateur qui contient l'article à écrire

LBU : Longueur de la zone d'échange en octets (apairé par FMS)

ANUM, ANUM1 : numéro de l'article (1 ⩽ ANUM ⩽ NART)
ANUM1 NART1
donné à la création du fichier)

PR : paramètre de retour

Remarque .

en fonction du Noyau : idem DREAD

Si NAR = 0 on met ANUM1 à 0
Traitement effectif si ANUM, ANUM1 < NART, NART1, sinon INART

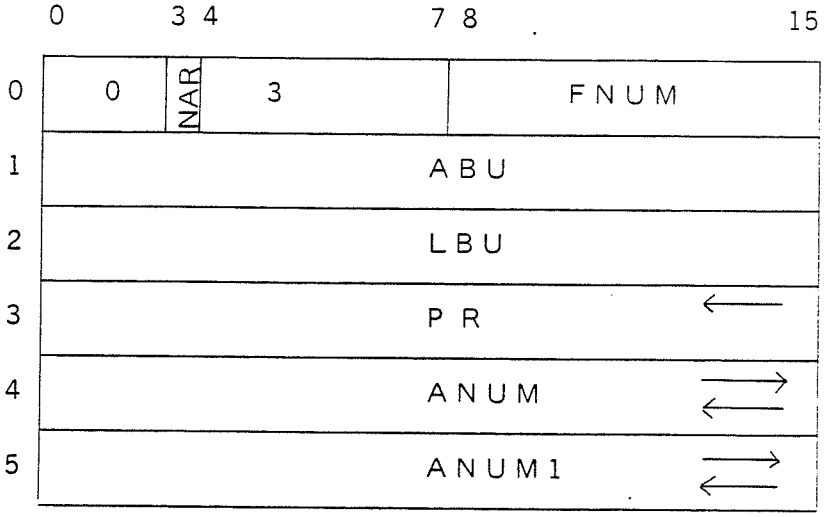
Comptes Rendus	Valeur de PR	Signification
	0	**Primitive exécutée correctement**
	`6003	**Article du fichier plus long :** l'article du fichier est plus long que la zone d'échange spécifiée dans le FCB. Toute la zone d'échange est transférée dans l'article, cadrée à gauche. La fin de l'article est laissée inchangée.
	`6004	**Article du fichier plus court :** L'article du fichier est plus court que la zone d'échange spécifiée dans le FCB. L'article entier est réécrit avec le début de la zone d'échange.
	`600A	**FAU inéxistante**
	`600E	**Article inexistante :** Le numéro de l'article spécifié n'appartient pas à l'ensemble défini à la création du fichier [1, NART (NART1) La primitive est ineffective.
	`6014	**Protection écriture**
	`6018	**Incompatibilité primitive - fichier**
	`601F	**Primitive en cours**
	`6028	**Erreur de syntaxe** (@ FCB, FONCT, ABU, LBU, incompatibilité codage ANUM et noyau)
	`6029	**Adresse FCB invalide**
	`602B	**Méthode d'accès non gérée**
Erreurs graves		
	`6032 `6034	**Informations système invalides**
	`6035	**FU verrouillée par IOCS**
	`4 - - -	**Erreur hardware : `4000** + mot d'état PU

g)

Nom	DCRE
But	Créer un article et son numéro d'identification

Appel RA : = @ FCB SVC (FMSD) ou FMSD = '3B

FCB



Description des paramètres

NAR : codage de ANUM
NAR = 0 ⟹ codage de ANUM sur 1 mot
NAR = 1 ⟹ codage de ANUM sur 2 mots (ANUM 1 = Poids Forts)

FNUM : Numéro de l'unité d'accès au fichier

ABU : Adresse de la zone d'échange qui contient l'article à créer

LBU : Longueur de la zone d'échange en octets (apairé par FMS)

ANUM, ANUM1 :
en entrée : numéro spécifiant le début de la recherche séquentielle d'une place libre.
$1 \leqslant ANUM, ANUM\ 1 \leqslant NART, NART\ 1$
en sortie : numéro de la place trouvée, pour créer l'article et qui devient donc le numéro de l'article

Remarque :

en fonction du noyau : idem DREAD

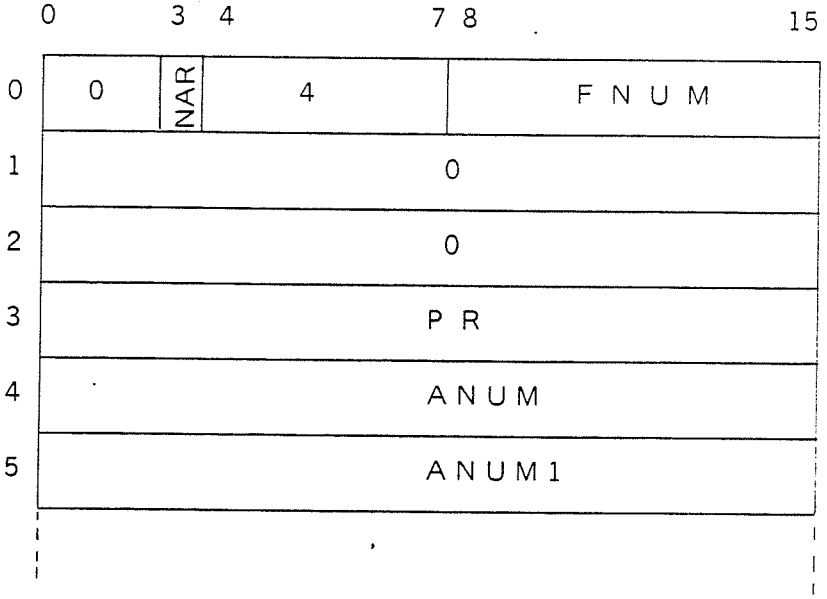
Comptes Rendus	Valeur de PR	Signification
	0	**Primitive correctement exécutée**
	`6003	**Article du fichier plus long :** l'article du fichier est plus long que la zone d'échange spécifiée dans le FCB. Toute la zone d'échange est transférée dans l'article, cadré à gauche. La fin de l'article est laissée inchangée.
	`6004	**Article du fichier plus court :** l'article du fichier est plus court que la zone d'échange spécifiée dans le FCB. L'article entier est écrit avec le début de la zone d'échange.
	`600A	**FAU inéxistante**
	`600E	**Article inéxistante :** le numéro spécifiant le début de la recherche séquentielle d'une place libre est invalide par rapport au paramètre (NART, NART1) défini à la création du fichier. La primitive est ineffective.
	`6014	**Protection écriture**
	`6016	**Fichier saturé :** il n'y a plus de place libre dans le fichier Direct à Trous. La primitive est ineffective.
	`6017	**Fichier trop long :** incompatibilité codage NART et noyau
	`6018	**Incompatibilité primitive - fichier**
	`601F	**Primitive en cours**
	`6028	**Erreur de syntaxe** (@ FCB, FONCT, ABU, LBU)
	`6029	**Adresse de FCB invalide**
	`602B	**Méthode d'accès non gérée**
Erreurs graves :		
	`6032 `6034	**Informations système invalides**
	`6035	**FU verrouillée par IOCS**
	`4 - - -	**Erreur hardware : `4000** + mot d'état PU

h)

Nom	**DSUP**
But	Supprimer un article

Appel RA : =@FCB SVC (FMSD) ou FMSD= '3B

FCB



Description des paramètres

NAR : codage de ANUM
NAR = 0 ⟹ ANUM codé sur 1 mot
NAR = 1 ⟹ ANUM codé sur 2 mots (ANUM 1 = Poids Forts)

FNUM numéro de l'unité d'accès au fichier

ANUM, ANUM1 : numéro de l'article à supprimer
1 ⩽ ANUM, ANUM1 ⩽ NART, NART1

Remarque :

en fonction du noyau : idem DREAD

Comptes Rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée**
'600A	**FAU inexistante**
'600E	**Article inéxistante :** le numéro de l'article à supprimer est invalide La primitive est ineffective
'6014	**Protection écriture**
'6017	**Fichier trop long :** incompatibilité codage NART et noyau
'6018	**Incompatibilité primitive - fichier**

`601F	**Primitive en cours**
`6028	**Erreur de syntaxe** (@FCB, FONCT, incompatibilité codage de ANUM et noyau)
`6029	**Adresse de FCB invalide**
`602B	**Méthode d'accès non gérée**

Erreurs graves

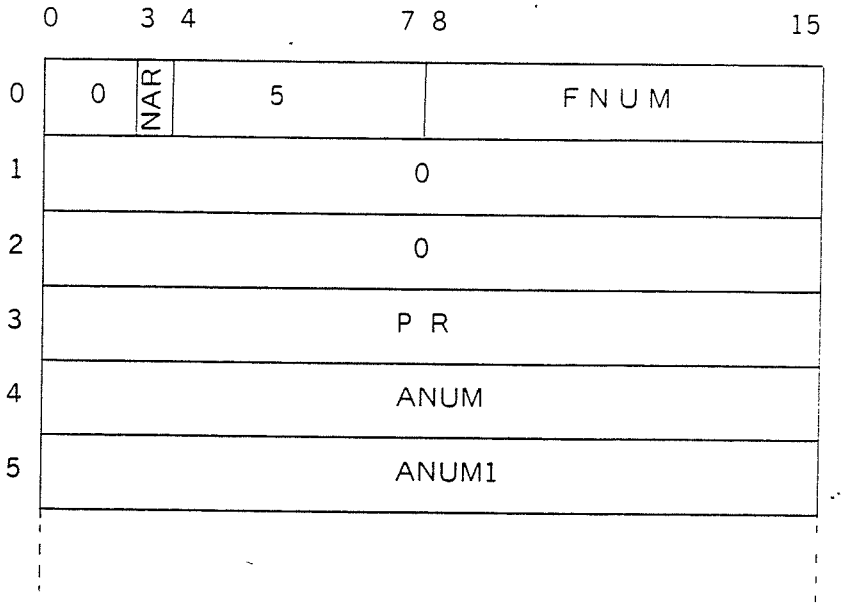
`6032 `6034	**Informations système invalide**
`6035	**FU verrouillée par IOCS**
`4 - - -	**Erreur hardware : `4000** + mot d'état PU

i)

Nom	DSEL
But	Sélection d'article : 1er mot

Appel RA : = @ FCB SVC (FMSD) ou FMSD = '3B

FCB



Description des paramètres

NAR : codage de ANUM
NAR = 0 ⟹ ANUM est codé sur 1 mot
NAR = 1 ⟹ ANUM codé sur 2 mots (ANUM 1 = Poids Forts)

FNUM : numéro de l'unité d'accès au fichier

ANUM, ANUM1 : numéro de l'article à sélectionner, le pointeur courant adresse le 1er mot de l'article

Remarque :

en fonction du noyau : idem DREAD

Comptes Rendus

Valeur de PR	Signification
0	**Primitive correctement exécutée**
'600A	**FAU inexistante**
'600E	**Article inexistant:** Le numéro de l'article n'appartient pas à la plage définie à la création du fichier (1, NART (NART1)). La requête est ineffective.

`6017	**Fichier trop long** : incompatibilité codage NART et noyau
`6018	**Incompatibilité primitive fichier**
`601F	**Primitive en cours**
`6028	**Erreur de syntace** (@ FCB, FONCT)
`6029	**Adresse de FCB invalide**
`602B	**Méthode d'accès non gérée**

Erreurs graves

`6032	**Informations système invalides**
`6034	
`6035	**FU verrouillée par IOCS**
`4 - - -	**Erreur hardware : `4000** + mot d'état PU

8 — LE FICHIER DE TYPE SEQUENTIEL INDEXE

8 — LE FICHIER DE TYPE SEQUENTIEL INDEXE

8.1 - ORGANISATION

8.1.1 - Organisation logique

L'organisation logique d'un fichier de Type **Séquentiel Indexé** est définie à la création du fichier. elle posséde le **numéro 3.**

(a) Description générale

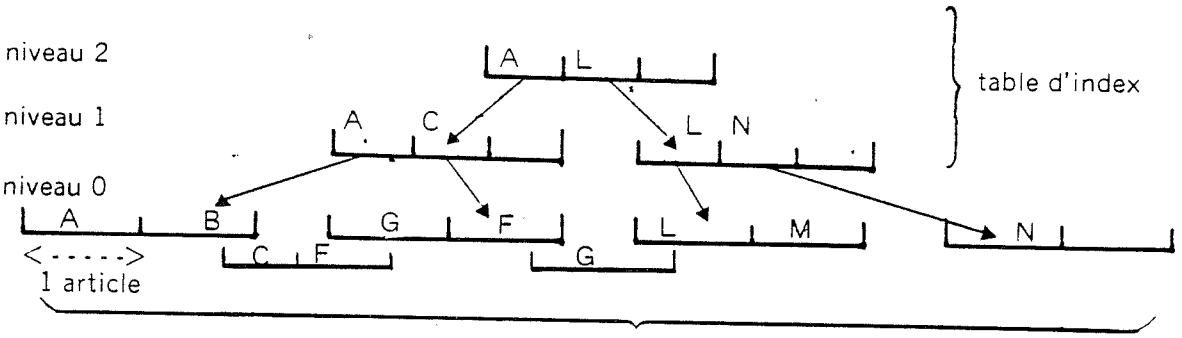
— Les articles

Un fichier séquentiel indexé peut être constitué d'un grand nombre d'articles (de l'ordre de 100 000) identifiés par un nom (ou clé d'accès).
Les noms des articles peuvent être quelconques : lettres, chiffres, caractères spéciaux. Pour FMS, ce ne sont que des quantités binaires sans signe.
Leur longueur est fixée par l'utilisateur à la création du fichier. Elle n'est pas limitée mais doit être constante pour un même fichier.
De même l'article peut-être de taille quelconque (la limite théorique est de 2 K mots) mais constante pour un même fichier.
Quant au nombre d'articles, il n'est pas borné par FMS. La seule contrainte est la taille du support un fichier ne pouvant occuper plus d'une FU (donc au maximum 2,5 millions de mots).

— Structure multi-niveaux

L'organisation séquentielle indexée est dite "multi-niveaux". En effet, l'article n'est accessible qu'après avoir parcouru un certain nombre de niveaux de la table d'index.
La table d'index est organisée selon une structure arborescente basée sur le classement des articles.
La localisation de l'article s'affine au fur et à mesure que l'on descend de niveau.

Schéma d'un fichier séquentiel indexé



Le nombre de niveaux est géré dynamiquement par FMS selon le principe suivant : un poste de table d'index a une taille fixe et le sommet de l'arborescence est constitué d'un seul poste. Quand il est plein il faut donc créer un niveau de plus.
le nombre de niveaux est compris entre 2 et 5.

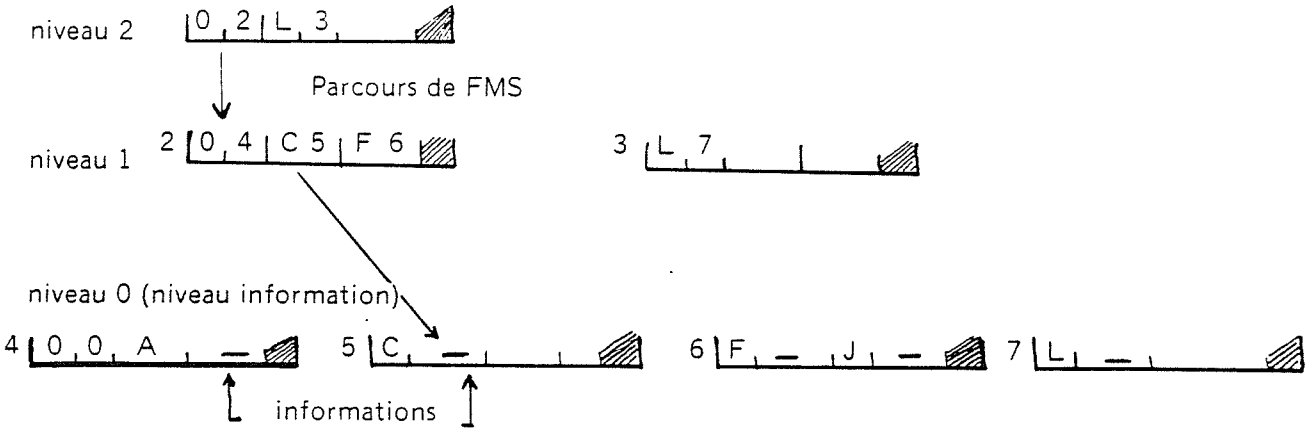
— Organisation dynamique

La place du fichier sur le support physique est réservée dès sa création.
Cependant, à l'intérieur de cet espace, FMS gère dynamiquement l'allocation de la place utilisée.
L'espace réservé est découpé en n **postes** de même taille.
Le poste est l'unité d'allocation gérée par la méthode d'accès.

A la création du fichier, trois postes sont systématiquement alloués : un poste système, un poste table d'index et un poste information.
Lors de l'addition d'un article, FMS lit la table d'index de plus haut niveau, et recherche dans cette table le plus grand nom inférieur ou égal au nom de l'article. FMS lit alors le poste dont le numéro est associé à ce nom. Le même processus se répète jusqu'à atteindre le poste information.
FMS cherche alors à insérer le nouvel article de façon à préserver l'ordre croissant des noms.
Si ce poste est saturé, FMS alloue un nouveau poste qu'il insère devant le poste saturé, et répartit les éléments à égalité dans les deux postes.
L'allocation de ce nouveau poste provoque alors l'insertion d'un nouvel élément dans la table d'index. Cela peut, à nouveau, en cas de saturation, provoquer l'allocation d'un nouveau poste, et même éventuellement la création d'un niveau supplémentaire.

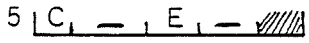
Par ailleurs le déplacement des articles dans les postes peut provoquer la mise à jour de la table d'index. En effet, une table associe à un poste le nom du premier article de ce poste. Si celui-ci change, il faut donc mettre à jour la table.
Une telle addition a donc réorganisé le fichier. C'est grâce à ce dynamisme que le fichier ne se détériore jamais et que le temps d'accès à un article est constant quelle que soit sa place.

Exemple : soit un fichier de ce type dans lequel on veut ajouter l'article de nom E

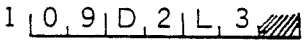


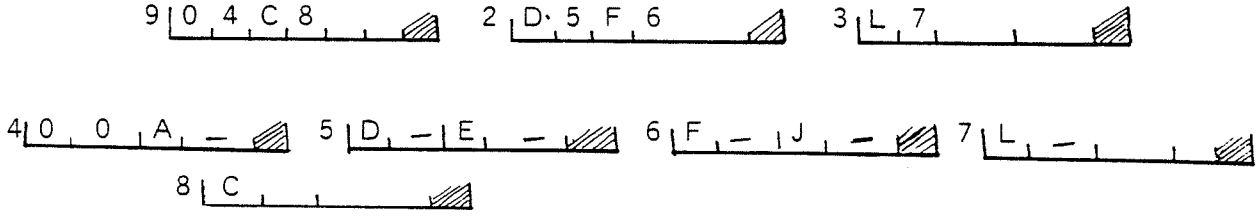
Après addition :

Seul le poste n° 5 est modifié et devient



Si maintenant, on ajoute un article de nom D, le fichier devient, après un parcours identique :





Cette addition a bouleversé le fichier. Elle est donc assez longue à exécuter.
C'est pour favoriser les additions non "réorganisantes" que le système essaye de répartir au mieux les articles dans les postes.
C'est pour la même raison qu'il est conseillé de configurer le fichier de manière à ce qu'un poste puisse contenir un nombre suffisamment grand d'articles : par exemple une dizaine, six au moins.

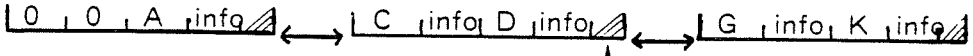
Minimum : 2 articles par poste et 3 index par poste.

Ce n'est pas l'emplacement physique des postes qui assure le tri des articles. En réalité les postes d'un même niveau sont chainés entre eux. Cela facilite les insertions et permet le parcours séquentiel au niveau information.

Cependant le niveau information peut être structuré de 2 façons différentes : la structure **"table directe"** et la structure **"table inverse"**. Le choix entre ces structures est fait par l'utilisateur selon la quantité d'articles de même nom qu'il pense stocker.

(b) Les tables directes

La structure de table directe est celle représentée à l'exemple d'addition précedent. Elle se caractérise par un niveau information de ce type (niveau 0).



chaine amont-aval info système

Les postes de ce niveau constituent une seule chaine continue.
Un même poste peut contenir des articles de noms différents et seul le nom du 1er article est cité au niveau supérieur.
La structure table directe étant choisie, l'utilisateur précise à la création si le fichier peut ou non contenir des homonymes.
Dans une table directe sans homonyme le système refuse d'ajouter un article si son nom existe déjà dans le fichier. Il n'y a donc jamais d'ambiguité à la lecture.
Dans une table directe avec homonymes FMS autorise toute addition, quel que soit le nom de l'article. Cependant FMS précise, en compte-rendu d'une lecture ou d'une addition, s'il existe plusieurs articles du nom cité. L'ordre de rangement des homonymes est l'ordre chronologique. Lors d'une lecture directe, FMS rend le dernier article créé parmi les homonymes. L'utilisateur peut obtenir les autres homonymes à l'aide de lectures séquentielles.

(c) Les tables inverses

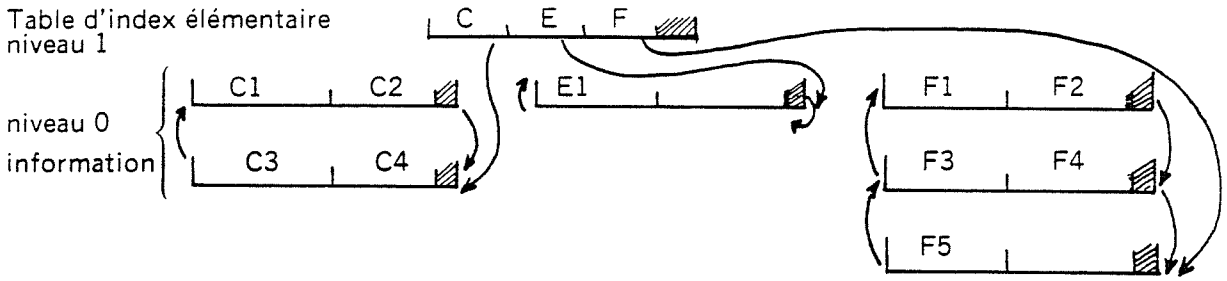
— Définitions

On parle de table inverse lorsque, le fichier comportant beaucoup d'articles de même nom, il est préférable de ne pas répéter ce nom pour chaque article.
L'utilisateur choisit donc une structure de table inverse lorsqu'il sait qu'à une valeur de clé donnée correspond non pas un article, mais une liste d'articles (appelée liste inverse) : autrement dit une clé d'accès n'est pas discriminante d'un article.
Exemple : Supposons un fichier contenant le nom et l'âge de tout le personnel. Si l'utilisateur a besoin de connaitre l'âge d'une personne il choisit la clé d'accès : nom et créé une table directe avec homonymes car il peut exister plusieurs personnes de même nom mais c'est assez rare. Par contre, s'il a besoin de répertorier toutes les personnes ayant par exemple 30 ans, il choisit la clé : âge, et crée une table inverse car il s'attend systématiquement à plusieurs articles répondant à la valeur de clé : 30 ans.

— **Structure**

Dans une table inverse, la table d'index a la même structure que dans une table directe. Cependant les noms de tous les articles figurent au dernier niveau de table d'index et ne sont pas répétés au niveau information.
Le niveau information est constitué d'un ensemble de chaines indépendantes rassemblant tous les postes contenant des articles de même clé : **les listes inverses.**
A partir d'une table d'index on accède au dernier poste (dernier selon la chronologie de création) d'une liste inverse. Les autres postes sont ensuites accessibles à l'aide d'un parcours séquentiel.

Schéma (partiel) d'une table inverse



C, E, F sont les noms (ou clés) des articles
C_i est la partie information du ième article de nom C.

— **Avantages** des tables inverses

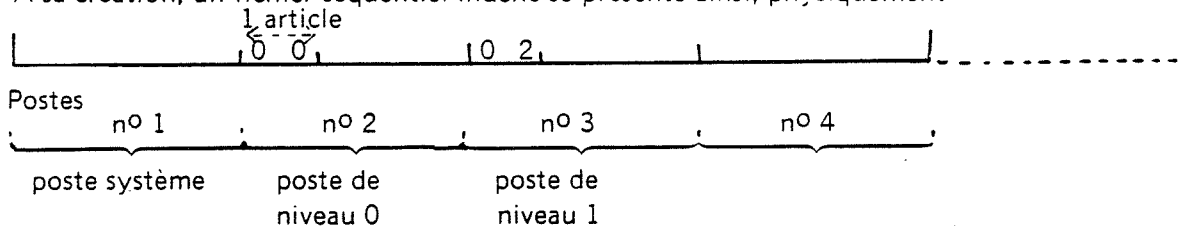
Quand elles sont utilisées pour des configurations de données adaptées, les tables inverses présentent un certain nombre d'avantages.

— gain de place sur disque, puisque le nom n'est pas répété dans chaque article. Le gain n'est cependant appréciable que si les noms sont volumineux et que le nombre moyen d'homonymes est suffisant pour que les parties information remplissent un nombre entier de postes.
— rapidité d'accès liée au gain de place. Ne pas répéter les noms peut diminuer le nombre de niveaux de fichiers.
— facilité d'accès aux homonymes. Les listes inverses individualisant les articles de même clé, l'accès séquentiel permet automatiquement d'obtenir la liste de tous les homonymes. Par contre le parcours séquentiel de plusieurs listes contigües est plus long que dans une table directe.
— solution au problème des articles de longueur variable.

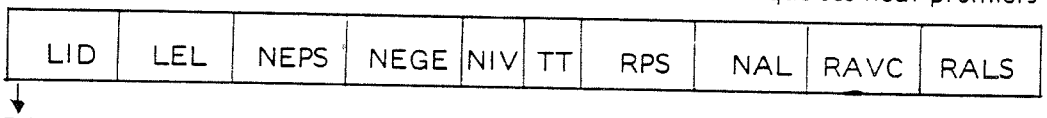
Dans le cas où l'utilisateur a des données de longueur très variable accessibles par nom, il peut créer une table inverse en définissant comme taille d'article la taille minimale de ses données. Toute donnée dépassant cette taille, sera découpée en articles de taille fixe introduits consécutivement. Dans ce cas, une liste inverse deviendrait une donnée unique, accessible par morceaux.

8.1.2 - Structure physique

A sa création, un fichier séquentiel indexé se présente ainsi, physiquement



Au fur et à mesure que le fichier évolue l'ordre physique des postes alloués devient alléatoire.
Seul le poste système reste en tête du fichier. Ce dernier n'a que ses neuf premiers mots significatifs.



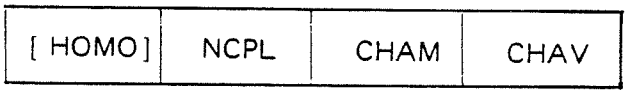
Début
de secteur

LID	=	longueur de la clé (en mots)
LEL	=	taille de l'article (en mots)
NEPS	=	nombre maximum d'éléments par poste supérieur
NEGE	=	nombre maximum d'articles par poste information
NIV	=	nº du plus haut niveau (octet gauche)
TT	=	type de table (cf CREAT) (octet droit)
RPS	=	nº du poste de plus haut niveau
NAL	=	nombre de postes libres
RAVC	=	nº du 1er poste vide chainé
RALS		nº du 1er poste libre en séquence (à la fin du fichier)

N.B. les deux dernières informations s'expliquent par le mode de gestion des postes libres. Il y a deux types de postes libres :
— ceux de la fin du fichier qui n'ont sans doute jamais été alloués. Il suffit de connaitre le nº du 1er poste de cette zone c'est RALS.
— les postes désalloués au coeur du fichier qui sont chainés entre eux et dont RAVC indique la tête de chaine
Lors d'une allocation, FMS choisit en priorité un poste vide chainé afin de condenser au maximum le fichier. Cela permet de corriger éventuellement un dimensionnement trop imprécis à la création, et ceci grâce à FUP, pendant la vie du fichier.

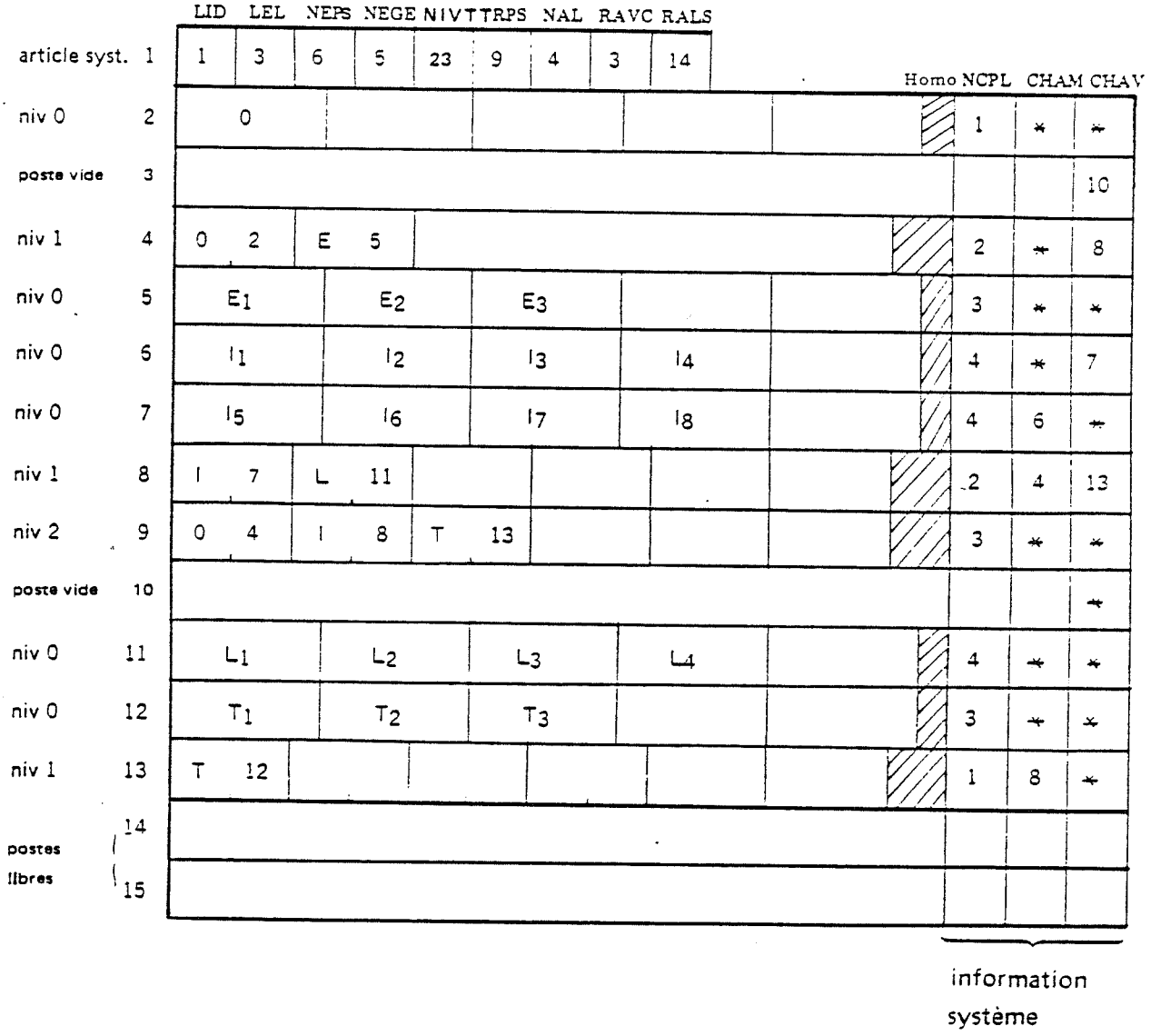
A la fin de chaque poste, le système utilise trois mots (quatre pour les postes information des tables directes avec homonymes).

Ce sont :



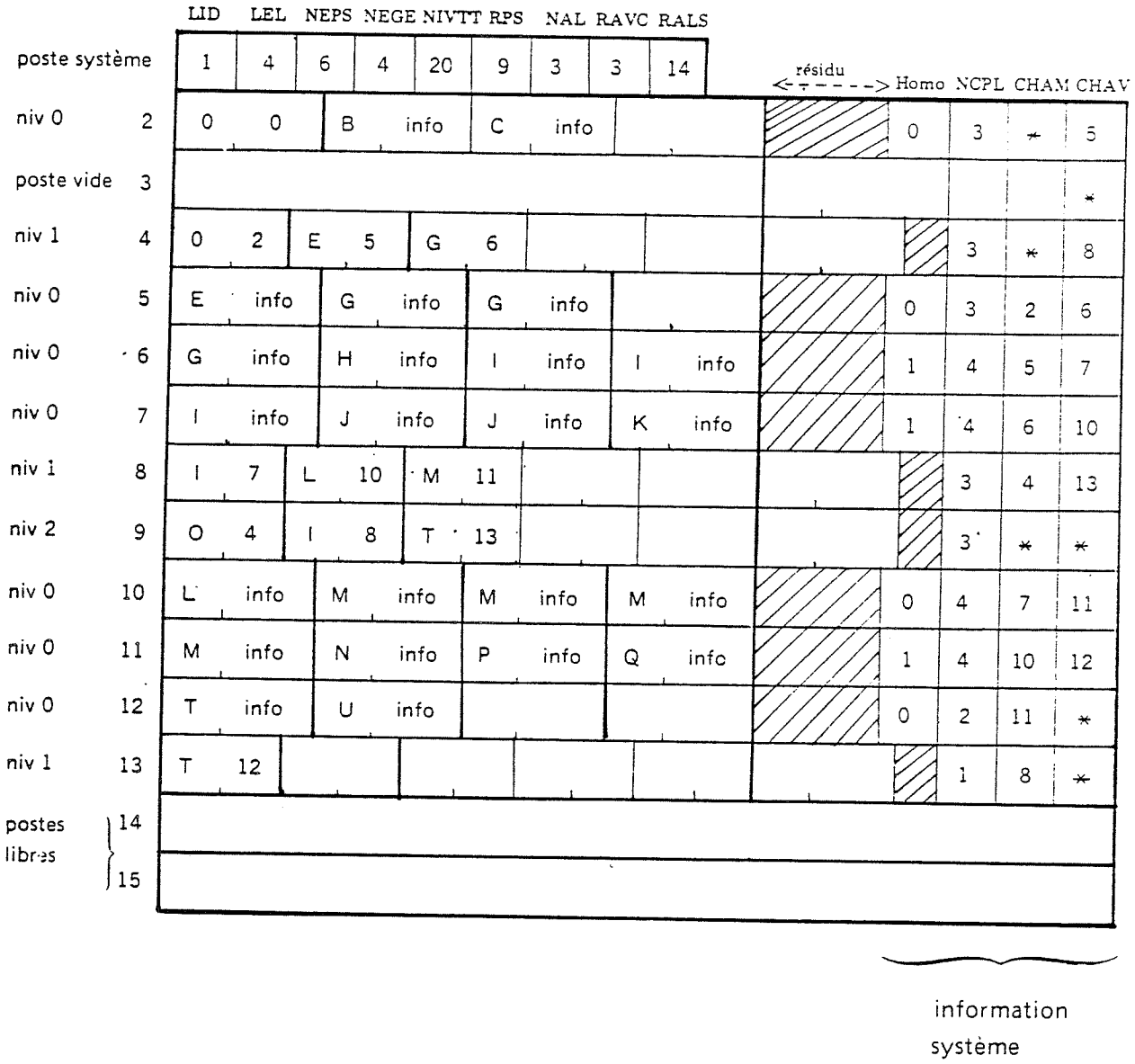
HOMO	indicateur d'homonymie (inexistant si TT = 01)	= 0/1 (1 si homonyme)
NCPL	nombre d'articles existants dans le poste	
CHAM	nº du poste précédent	
CHAV	nº du poste suivant	

Exemple de table inverse à 3 niveaux



N.B. : E_i signifie information relative à l'article de nom E
* signifie "fin de chaine"

Exemple de table à 3 niveaux avec homonymes



Pour retracer l'arborescence logique du fichier à partir d'un tel schéma, il faut partir de RPS dans le poste système pour avoir la tête de l'arbre, puis lire le contenu de ce poste supérieur pour obtenir les postes du niveau strictement inférieur et ainsi de suite. . . .

L'exemple précédent se traduit par :

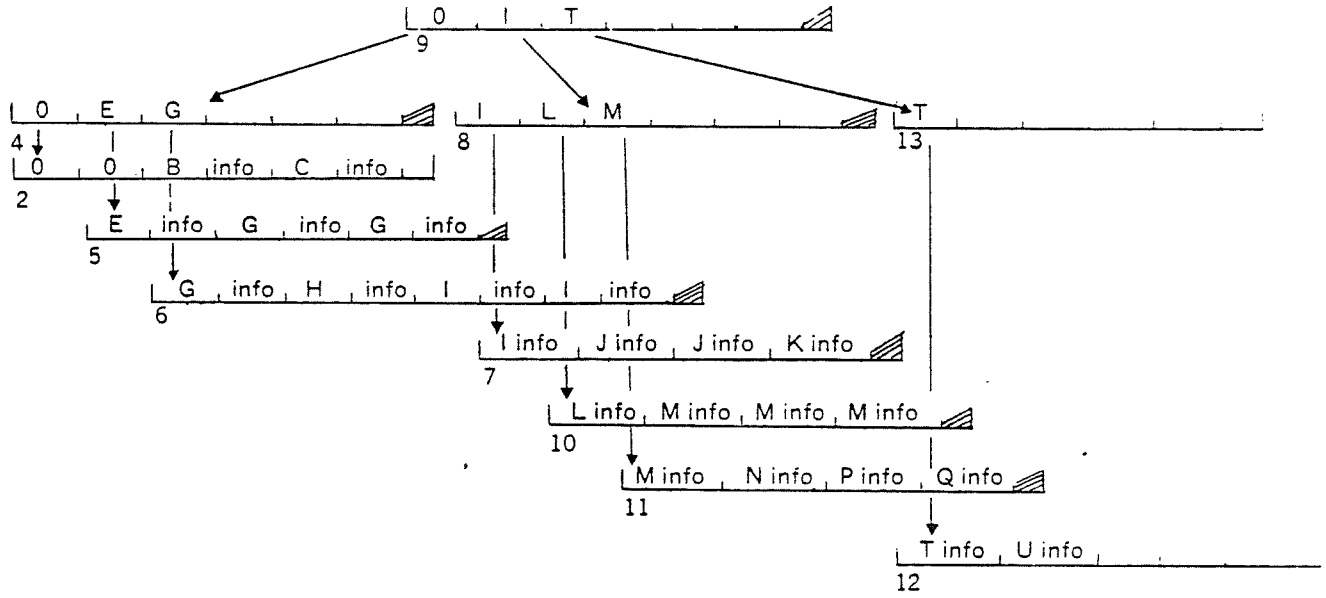
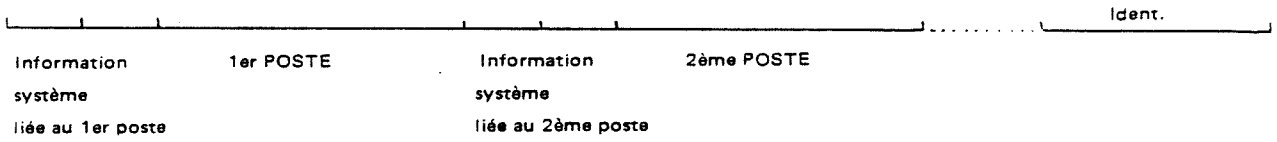


Schéma du buffer de travail



Informations liées à un poste

	1er Mot	2ème Mot
	Numéro poste	DFPM Numéro NIV:

DFPM : 4 premiers bit de l'octet gauche du 2ème mot.
- D : 0/1 1 si 1er poste du buffer
- F : 0/1 1 si dernier poste du buffer
- P : 0/1 1 si poste protégé
- M : 0/1 1 si poste modifié (à réecrire)

8.2 - SA METHODE D'ACCES

La méthode d'accès Séquentiel Indexé est celle qui gère l'organisation des fichiers séquentiels indexés (numéro 3).
Elle est entièrement réentrante : si une tâche est en train d'exécuter une requête séquentiel indexé une autre tâche plus prioritaire faisant elle aussi appel à la méthode d'accès pourra exécuter sa requête.

8.2.1 - Types d'accès fournis

Ce sont les mêmes que la structure du fichier soit du type table directe ou table inverse

ⓐ Accès direct par nom

Il existe deux requêtes d'accès direct par nom à l'article
Ce sont :
SIREAD : lecture d'un article de nom donné
SIADD : insertion d'un nouvel article à une place dans le fichier fonction de son nom.

Ces deux requêtes accèdent aux articles après avoir parcouru les différents niveaux de tables.

ⓑ Accès séquentiel logique

Il existe une requête permettant d'accéder à un article d'après sa position relativement à l'article précédemment traité.
SIRIS (± n) : lecture du nième article précédant (—) ou suivant (+) l'article courant.

Grâce à cette primitive l'utilisateur peut parcourir tout le niveau information de son fichier séquentiellement. Etant donné que les articles sont rangés selon l'ordre croissant binaire de leurs clés, le parcours séquentiel permet d'obtenir une liste ordonnée des articles.

ⓒ Accès à l'article courant

Deux primitives utilisent le positionnement dans le fichier effectué par la primitive précédente et traitent l'article pointé. Ce sont :
SIWRIT : réécriture de l'article courant
SISUP : suppression de l'article courant.

8.2.2 - Type d'accès autorisé

La méthode d'accès autorise l'accès séquentiel pur à tout le fichier. Ceci ne peut se faire qu'à l'aide de la méthode d'accès séquentiel après l'une des trois primitives de création d'une mise d'accès (CREAT, OPEN NEW, OPEN OLD) et avant toute requête du niveau article du Séquentiel indexé. Tout le fichier est alors accessible (y compris les tables d'index). Après CREAT ou OPEN NEW le pointeur courant adresse le poste de niveau 0 initialisé avec l'article nul.
Après OPEN OLD, le pointeur courant adresse le début physique du fichier, c'est-à-dire le début du poste système. Ce type d'accès est principalement utilisé pour les archivages de fichiers.
Par contre le Séquentiel en Portion d'Article est interdit par les requêtes de la méthodes d'accès Séquentiel Indexé.

8.2.3 · Gestion des mémoires tampons

La méthode d'accès nécessite deux types de buffers pour son fonctionnement :
- un buffer de travail, lié à une unité d'accès au fichier, et réservé au système pour ses échanges avec le disque
- une zone d'échange, liée à une primitive du niveau article et utilisée pour les transferts d'information entre l'utilisateur et FMS.

a) **Le buffer de travail** (voir Page 8-9)

Ce buffer est donné par l'utilisateur à la création d'une unité d'accès au fichier : CREAT, OPEN NEW, OPEN OLD.
Il devient la propriété de FMS pendant toute la durée de vie de l'unité d'accès, c'est-à-dire jusqu'au CLOSE.
Pendant toute cette période, l'utilisateur ne doit donc pas le modifier.
FMS l'utilise pour ses échanges avec le disque. Sachant que l'unité d'échange est le poste, ce buffer doit au moins pouvoir contenir un poste.
Cependant, les temps de réponse s'améliorent lorsqu'il est de taille suffisante pour contenir simultanément plusieurs postes.
En effet le mécanisme de gestion du buffer est basé sur un principe analogue à celui d'un algorithme de pagination.
Le fichier étant découpé en un ensemble de postes (que l'on peut assimiler à des pages) il existe un algorithme chargé d'alimenter le buffer avec les postes référencés.
Le système attache à chaque poste une priorité associée au niveau de ce poste. Lorsqu'un poste demandé n'est pas nécessaire l'algorithme remplace le poste de plus bas niveau à condition qu'il n'ait pas été déclaré comme "protégé".
Ce choix du poste à remplacer satisfait deux types de besoin :
- conserver en mémoire les postes de plus haut niveau puisque ce sont les plus souvent réutilisés.
- pouvoir travailler sur deux postes adjacents de même niveau lors d'une réorganisation du fichier (cas de protection).

Les performances maximales sont donc obtenues avec un buffer pouvant contenir autant de postes qu'il existe de niveaux plus un.
Continuant l'analogie avec un système de pagination, un poste n'est réécrit sur disque qu'au moment où il est remplacé en mémoire : c'est ce que FMS appelle **écriture retardée.** Celle-ci est demandée par l'utilisateur lors des primitives d'écriture. Si pour des raisons de sécurité, l'utilisateur préfère que toute modification d'un poste soit immédiatement répercutée sur le disque, il a la possibilité de demander **l'écriture immédiate.**

b) **La zone d'échange**

A chaque primitive du niveau article, l'utilisateur fournit une zone mémoire qui lui permet d'échanger avec le système le contenu d'un article.
Cette zone doit pouvoir contenir au moins une clé. Si sa taille est différente de celle de l'article, le système le signale en tant qu'avertissement, mais réalise l'échange.

8.3 - FONCTIONS LOGIQUES

8.3.1 - Le niveau fichier

a) **Généralités**

Les primitives du niveau fichier dans le cas séquentiel indexé sont semblables à celles de toute autre méthode d'accès.
Il n'est décrit ici que les particularités, liées à la méthode d'accès, des primitives créant une unité d'accès (CREAT, OPEN NEW, OPEN OLD).
Le fonctionnement général de ces primitives ainsi que la description de toutes les autres primitives (CLOSE, DELET , CATAL, RENUM, ALTER, RENAM) se trouvent dans le chapitre 4 L'ENTITE FICHIER.

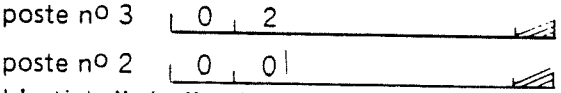
L'interface d'appel de toute primitive du niveau fichier est en PL1600 :
RA :=@FCB ;
SVC (FMS) ; où FMS = '38

b) **Création d'un fichier : CREAT, OPEN NEW**

A la création de son fichier, l'utilisateur fournit à FMS :
- la longueur de l'article
- la longueur du nom de l'article
- la longueur du poste
- le nombre maximal de postes dans le fichier.

FMS-E réserve alors sur disque la place nécessaire pour le nombre maximal de postes.
A l'intérieur de cet espace, FMS commence à allouer trois postes.
- le premier est un poste système mémorisant les caractéristiques du fichier
- le deuxième est un poste information initialisé avec un article entièrement nul
- le troisième est une table d'index dont le premier élément associe le nom "zéro" au numéro du poste information contenant l'article nul.

Après création, le fichier contient donc :



L'article "zéro" créé par le système sera toujours le plus petit (donc le premier) article du fichier. Sa suppression est interdite à l'utilisateur.
En fin de chaque poste, FMS se réserve une zone d'informations système (zone hachurée).

Nom	CREAT	Séquentiel Indexé
But	Créer un fichier permanent et initialiser une unité d'accès à ce fichier.	

Appel : RA : = @FCB ; SVC (FMS) ; ou FMS = '38

FCB

	0	1	2	3	4 ... 7	8 ... 15
0	1	ALDR	NLC	0	4	FNUM
1	ABU					
2	LBU					
3	PR ←					
4	FNAM 1					
5	FNAM 2					
6	FNAM 3					
7	PUBW					
8	FTYP					SU/FU
9	LART					
10	LP					
11	NP					
12	LCLE					

Description des paramètres

FONCT : '84 : code de fonction de CREAT
Si l'utilisateur souhaite des options (accès direct rapide, entrées-sorties paramètrées) il le code dans FONCT en positionnant les bits spécifiques

FNUM : nº d'utilisation du fichier

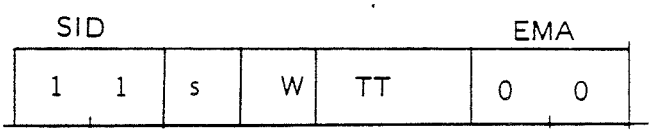
ABU : adresse du buffer de travail

LBU : longueur du buffer de travail
LBU doit être un nombre pair d'octets et vérifier :
LBU ⩾ (longueur du poste + 4)+ longueur de la clé.
Pour que le buffer de travail puisse contenir i postes simultanément
LBU doit vérifier :
LBU = i (longueur du poste + 4) + longueur de la clé

FNAM1
FNAM2 } nom du fichier (en ASCII)
FNAM3

PUBW : nom du catalogue (en ASCII)

FTYP : type du fichier décomposé en



S = 1 ⇔ fichier simultané
W = 1 ⇔ écriture autorisée
TT = 00 ⇔ table directe avec homonymes
01 ⇔ table directe sans homonyme
11 ⇔ table inverse .

SU/FU désignation du support (par SU ou FU)

LART longueur de l'article (nombre pair d'octets). L'article est l'élément générique du niveau information. Il comprend une partie clé et une partie information dans les tables directes et uniquement la partie information dans les tables inverses.

Règle : Configurez au minimum, 2 articles par poste et 3 index par poste.

LP longueur du poste (k × 256 octets). Le poste doit obligatoirement représenter un nombre entier de secteurs disque tel que LP ⩽ 12 K octets

NP nombre maximum de postes
Il doit vérifier 3 ⩽ NP ⩽ 65 535

LCLE longueur de la clé d'accès (nombre pair d'octets). Dans les tables directes, le système vérifique que 0 ⩽ LCLE ⩽ LART

Comptes Rendus

Valeur de PR	Signification CREAT
0	**Primitive déroulée correctement**
'600B	**FAU existante**
'600D	**Fichier existant**
'6017	**Fichier trop long :** le fichier ne tient pas sur 128 granules
'601F	**Primitive en cours**
'6020	**Zone de pavés saturée**
'6021	**FU saturée.** Il n'y a pas suffisamment de place sur la FU support pour tout le fichier
'6022	**Table des fichiers saturée**
'6028	**Erreur de syntaxe :** (FCB, FONCT, ABU, LBU, FNAM, PUBW, FTYP, LART, LP, NP, LCLE).
'6029	**Adresse du FCB invalide**
'602A	**SU ou FU non gérée par FMS ou IOCS**
'602B	**Méthode d'accès non gérée**

Erreurs graves	· 6032	Informations système invalides
	· 6033	"
	· 6034	"
	· 6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU

Nom	OPEN NEW	Séquentiel Indexé
But	Créer un fichier temporaire et initialiser une unité d'accès à ce fichier	

Appel : RA : =aFCB ; SVC (FMS) ; ou FMS = `38

FCB

	0	1	2	3	4	7	8	15
0	1	ADR	NLC	0	1		FNUM	
1	ABU							
2	LBU							
3	PR ←							
4	FNAM 1							
5	FNAM 2							
6	FNAM 3							
7	PUBW = 0							
8	FTYP						SU/FU	
9	LART							
10	LP							
11	NP							
12	LCLE							

Description des paramètres

Les paramètres utilisés pour un OPEN NEW sont identiques à ceux d'un CREAT (voir pages précédentes) s'adressant à la même méthode d'accès, aux exceptions près suivantes :

FONCT : `81 : code fonction d'un OPEN NEW Séquentiel Indexé. Si l'utilisateur désire les options de performances, il le code à l'aide des bits correspondants de FONCT.

PUBW = 0

FTYP type du fichier

SID				EMA	
1	1	0	0	T T	0 0

TT = 00 ⇔ table directe avec homonymes
= 01 ⇔ table directe sans homonyme
= 11 ⇔ table inverse

Comptes Rendus	Valeur du PR	Signification	OPEN NEW Séquentiel Indexé
	0	La primitive s'est déroulée correctement	
	`600B	FAU existante	
	`600D	Fichier existant	
	`6017	Fichier trop long : le fichier ne tient pas sur 128 granules	
	`601F	Primitive en cours	
	`6020	Zone de pavés saturée	
	`6021	FU saturée : Il n'y a pas suffisamment de place sur la FU support pour tout le fichier	
	`6028	Erreur de syntaxe : (@FCB, FONCT, ABU, LBU, FNAM, FTYP, LART, LP, NP, LCLE)	
	`6029	Adresse du FCB invalide	
	`602A	SU ou FU non gérée par FMS ou IOCS	
	`602B	Méthode d'accès non gérée	
Erreurs graves	`6032	Informations système invalides	
	`6033	"	
	`6034	"	
	`6035	FU verrouillée par IOCS	
	`4 . . .	Erreur hardware : `4000 + mot d'état PU	

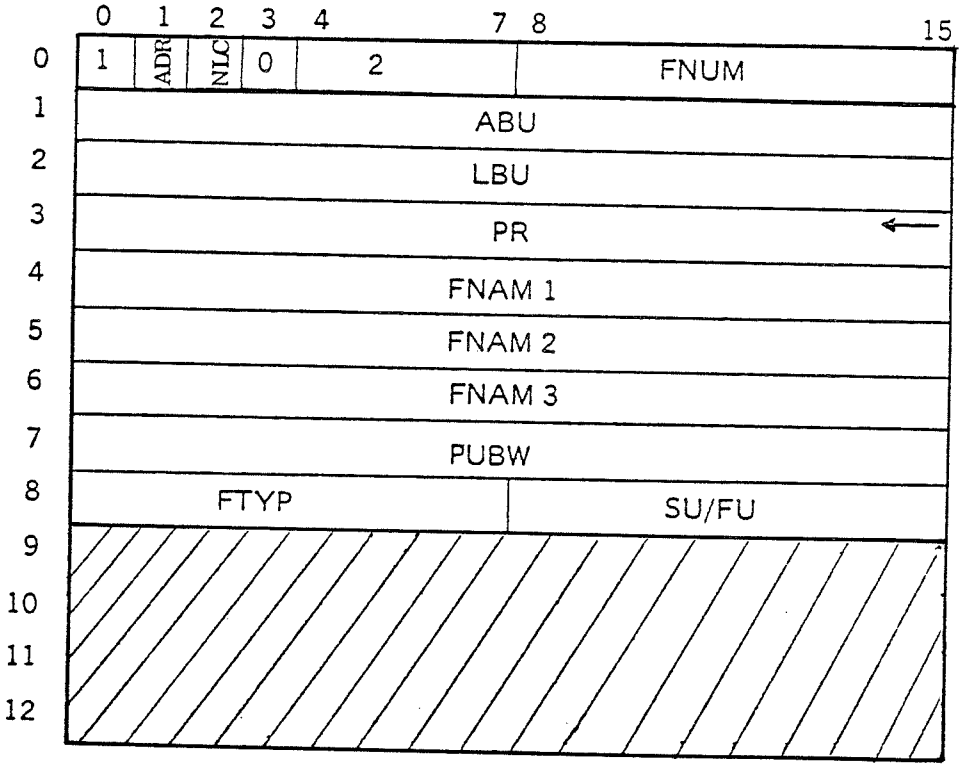
c) Ouverture d'un fichier OPEN OLD

Le seul traitement propre à la méthode d'accès lors de l'ouverture d'un fichier séquentiel indexé est de charger le poste système en mémoire et d'initialiser le buffer de travail.

Nom	OPEN OLD Séquentiel Indéxé
But	Créer une unité d'accès à un fichier permanent

Appel : RA : =@FCB ; SVC (FMS) ; où FMS = `38

FCB



Description des paramètres

FONCT : `82 sans tenir compte d'éventuels bits spécifiant des options choisies par l'utilisateur
FNUM : n° d'utilisation du fichier
ABU : adresse du buffer de travail
LBU : longueur du buffer de travail (≥ longueur du poste + 4 + longueur de la clé)
FNAM1, FNAM2, FNAM3 : nom du fichier en ASCII

FTYP : type du fichier

0	0	S	W	RWK	0	0

S = 1 ⇔ fichier simultané
W = 1 ⇔ demande d'accès en écriture
RWK n'a de signification que si S = 0.
Les seules valeurs autorisées sont :
RWK = 00 ⇔ l'unité d'accès doit être seule sur ce fichier
RWK = 01 ⇔ multi-accès en lecture
si S = 1 alors RWK = 00

SU/FU désignation du support
→PR paramètre de retour

Comptes Rendus	Valeur du PR	Signification	OPEN OLD Séquentiel indexé
	0	La primitive s'est déroulée correctement	
	`600B	FAU existante	
	`600C	Fichier inexistant	
	`6014	Protection écriture : Soit le fichier est protégé en écriture et l'utilisateur demande l'accès en écriture. Soit le fichier est non simultané, et l'utilisateur précise une RWK = 3 ou une RWK = 1 avec demande d'écriture	
	`6015	Permanent de nature différente	
	`601E	Fichier occupé	
	`601F	Primitive en cours	
	`6020	Zone de pavés saturée	
	`6028	Erreur de syntaxe : (↺ FCB, ABU, LBU, FNAM, PUBW, FTYP)	
	`6029	Adresse du FCB invalide	
	`602A	SU ou FU non gérée par FMS ou IOCS	
	`602B	Méthode d'accès non gérée	
Erreurs Graves	`6032	Informations Système Invalides	
	`6034	"	
	`6035	FU verrouillée par IOCS	
	`4...	Erreur hardware : `4000 + mot d'état PU.	

8.3.2 - Le niveau article

a) Notion de sélection

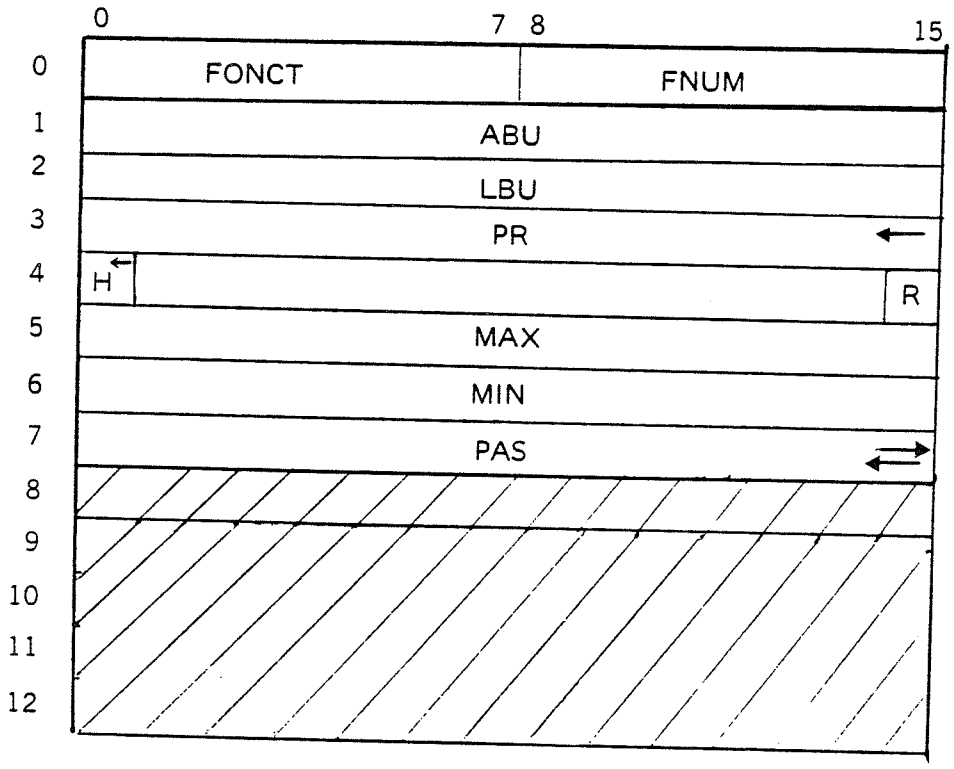
On dit qu'il y a sélection d'article lorsque, à la fin d'une primitive, le pointeur courant géré par FMS est positionné sur l'article cité par la primitive.
La sélection se fait à l'aide des primitives de lecture (SIREAD, SIRIS). Lorsqu'elles sont utilisées dans ce but, il suffit que l'utilisateur fournisse une zone d'échange, de la taille du nom de l'article.
La réécriture ne modifie pas la sélection antérieure.
Par contre, après une addition ou une suppression, aucun article n'est sélectionné.
Certaines primitives (SIRIS, SIWRIT, SISUP) nécessitent une sélection préalable. Elles ne peuvent donc s'enchainer qu'après des lectures ou des réécritures (SIREAD, SIRIS, SIWRIT).

b) **Généralités**

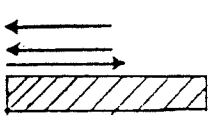
La séquence d'appel à FMS est, en PL1600.

```
RA : =@FCB ;
SVC (FMS X) ; <<FMS X = '28
```

Description générale des FCB Séquentiel Indexé



Chacun de ces paramètres sera explicite pour chaque primitive.
Conventions de notation :

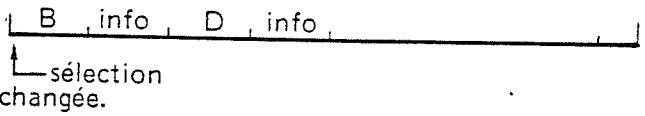


signifie paramètre de retour
signifie paramètre d'entrée mis à jour par FMS
signifie zone inutilisée par FMS

c) Lecture directe : SIREAD

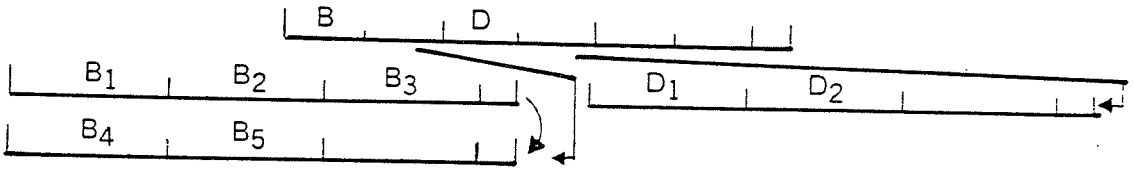
Lors d'une lecture d'un article à partir de son nom, FMS parcourt les niveaux successifs jusqu'à trouver le poste information susceptible de contenir l'article, c'est-à-dire celui caractérisé par le plus grand nom inférieur ou égal à celui de l'article (un poste information est caractérisé par le nom de son premier ou plus petit article).
L'article trouvé est alors chargé dans la zone d'échange, éventuellement partiellement si celle-ci est trop petite pour le contenir entièrement.
Dans le cas où il existe plusieurs articles de même nom, FMS rend le dernier créé, (chronologiquement) en signalant qu'il en existe d'autres par un compte rendu d'homonyme.
Lorsque l'article n'existe pas, FMS le signale, et sélectionne l'article de nom immédiatement inférieur
Le traitement de la zone d'échange est différent selon le type du fichier.
Supposons qu'un utilisateur demande SIREAD (C)

Table directe
Soit un poste de ce type :

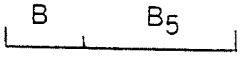


La zone d'échange reste inchangée.
L'utilisateur pourra lire B ou D selon ses besoins par des lectures séquentielles
SIRIS (0) fournit : B, info.
SIRIS (1) fournit : D, info.

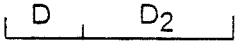
Table inverse
Soit un fichier de ce type



Si ce n'est le compte-rendu "article inexistant" SIREAD (C) est équivalent à SIREAD (B).
La zone d'échange est chargée :



Si l'utilisateur a besoin de D il utilisera la lecture séquentielle :
SIRIS (+ 1) provoquera "fin de chaine" et chargera D dans la zone d'échange
SIREAD (D) chargera la zone d'échange avec

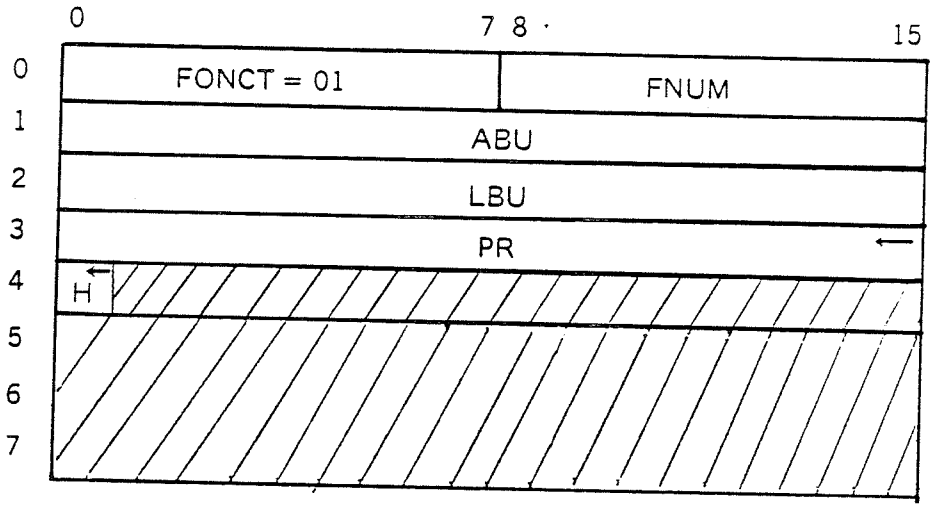


Ceci permet la recherche d'un article dont on ne connait pas parfaitement le nom (faute d'orthographe, ou nom incomplet).

Nom :	**SIREAD**
But :	Lecture directe d'un article

Appel RA : =@FCB ; SVC (FMSX) ; << FMSX = `28

FCB

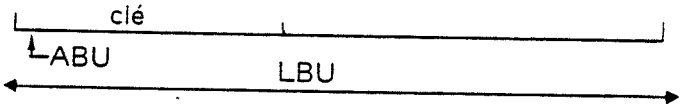


Description des paramètres

FNUM : nº d'utilisation du fichier
ABU : adresse de la zone d'échange
LBU : longueur de la zone d'échange (octets). Elle doit être supérieure ou égale à la longueur de la clé
PR : paramètre de retour
H : compte-rendu d'homonymie (Tables Directes avec Homonymes)
H = 1 ⇔ il existe des homonymes

La zone d'échange

— En entrée



— En sortie

Comptes Rendus	Valeur du PR	Signification	SIREAD
	0	Primitive exécutée normalement	
	`6003	Article du fichier plus long : celle-ci est remplie avec le début de l'article.	
	`6004	Article du fichier plus court : tout l'article est lu dans cette zone le reste de la zone est inchangé	
	`600A	FAU inexistante	
	`600E	Article inexistant : aucun article de ce nom dans le fichier	
	`6018	Incompatibilité primitive fichier	
	`601F	Primitive en cours	
	`6028	Erreur de syntaxe (@FCB, FONCT, ABU, LBU).	
	`6029	Adresse du FCB incorrecte	
	`602B	Méthode d'accès non gérée	
Erreurs graves	`6032	Informations système invalides	
	`6034	"	
	`6035	FU verrouillée par IOCS	
	`4 . . .	Erreur hardware : `4000 + mot d'état PU	

d) Lecture séquentielle : SIRIS (± n)

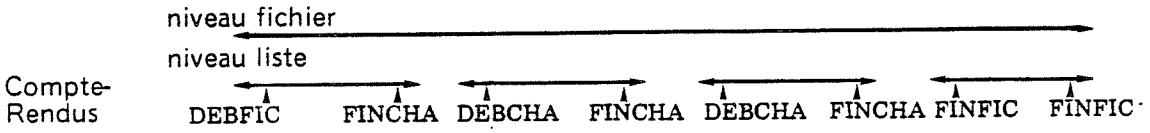
SIRIS (± n) provoque la lecture du nième article suivant (+ n) ou précédant (— n) l'article courant. Après avoir vérifié qu'il y avait eu sélection d'article, FMS parcoure séquentiellement le niveau information (en avant ou en arrière selon le signe du déplacement) jusqu'à atteindre le nième demandé. Celui-ci est alors chargé dans la zone d'échange.
Si n = 0, FMS charge l'article courant.
Si FMS arrive en fin de chaine avant d'avoir atteint le $n^{ième}$, il charge le dernier article trouvé et rend le déplacement effectivement réalisé.
Les compte-rendus du type "fin de chaine" sont donnés lorsque, avant la primitive, l'article courant était déjà le dernier de la chaine et que l'utilisateur envoie alors SIRIS dans le même sens de parcours.
Le parcours séquentiel est différent selon le type du fichier :

Table directe

Le fichier peut se parcourir dans les deux sens sans discontinuité. Pour lister un fichier selon l'ordre croissant des clés, l'utilisateur envoie :
SIREAD (0)
SIRIS (+ 1) ↺ jusqu'à obtenir le compte-rendu "fin de fichier".

Table inverse

Par contre dans une **table inverse**, seules les listes inverses se parcourent dans les deux sens. Cependant le système offre la possibilité d'un parcours **en avant** sur tout le fichier.
Schéma du parcours et des compte-rendus.

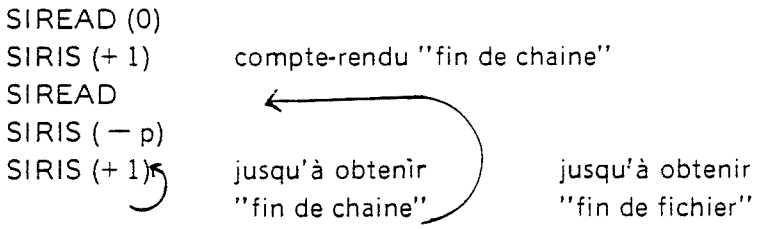


Pour parcourir séquentiellement toute une table inverse l'utilisateur écrit une séquence du type :

SIREAD (0)
SIRIS (— 1) ⇒ compte-rendu "début de fichier"
SIREAD
SIRIS (— 1) ↺ jusqu'à obtenir "début de chaine" alors jusqu'à "fin de fichier"

En effet zéro est le plus petit nom du fichier. Si l'utilisateur n'a pas créé d'autre article zéro, SIRIS (— 1) provoquera le compte-rendu "début de fichier".
En même temps que tout compte-rendu du type fin de chaine, FMS charge dans la zone d'échange de l'utilisateur le nom de la liste inverse suivante. Celui-ci peut donc se positionner à l'aide d'un SIREAD sur le dernier article de la liste suivante et la parcourir alors par SIRIS (— 1). Le compte-rendu "début de chaine" lui signalera la fin de la liste et il pourra passer à la liste suivante par le même processus. La dernière liste du fichier lui sera signalée par "fin de fichier". Le fichier est considéré comme circulaire et FMS charge alors le nom zéro dans la zone d'échange. Cela peut permettre de revenir au début du fichier.

La séquence proposée parcourt les listes inverses dans le sens opposé à la chronologie de leur création. L'utilisateur peut les parcourir dans le sens avant.
Il fait alors :



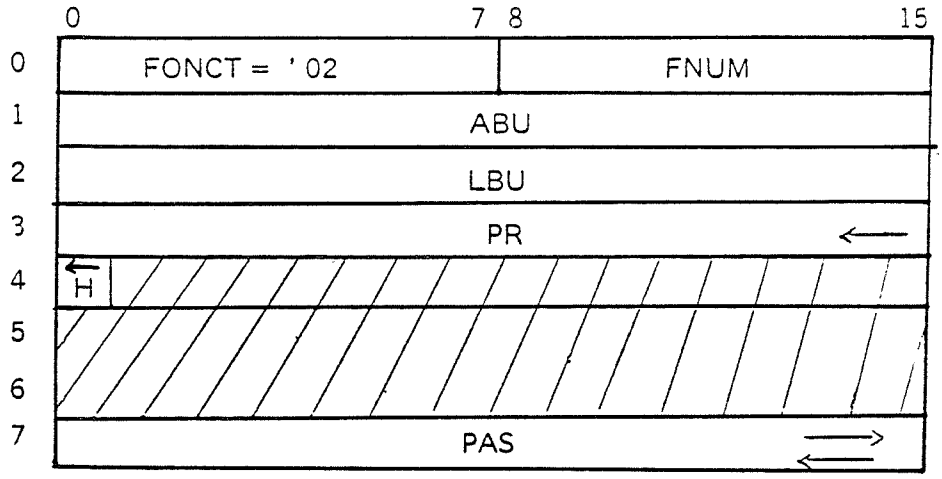
Il suffit de choisir p suffisamment grand (au moins égal à la plus grande longueur de liste) pour que SIRIS (— p) positionne sur le 1er article d'une liste quelle que soit cette liste.

Remarque : Dans une table directe avec homonymes, la primitive SIRIS fournit un compte-rendu d'homonymie qui, s'il est à 1 doit être interprété ainsi : l'article qui vient d'être lu a le même nom que l'article précédemment sélectionné.

Nom	**SIRIS**
But	Lecture du nième article précédent ou suivant

Appel RA : =ωFCB ; SVC (FMSX) ; où FMSX = `28

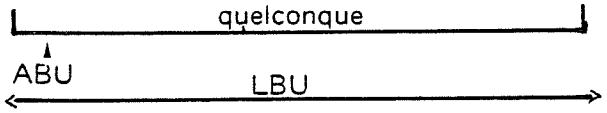
FCB



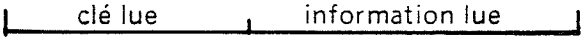
Description des paramètres

FNUM : nº d'utilisation du fichier
ABU : adresse de la zone d'échange
LBU : longueur de la zone d'échange (nombre pair d'octets ⩾ longueur d'une clé)
PAS : nombre algébrique précisant le déplacement par rapport à l'article courant
PR : paramètre de retour
H : compte-rendu d'homonymie (Tables Directes avec Homonymes)
H = 1 ⇔ l'article lu a le même nom que l'article précédemment sélectionné

— En entrée :



— En sortie normale



Sur compte-rendu "fin de chaine" ou "début de chaine" ou "fin de fichier" ou "début de fichier" d'une table inverse :

Comptes Rendus	Valeur du PR	Signification	SIRIS
	0	**Primitive exécutée correctement**	
	`6001	**Fin de fichier :** avant l'exécution de cette primitive le pointeur logique était déjà à la fin du fichier.	
	`6002	**Début de fichier :** avant l'exécution de la primitive le pointeur logique était déjà au début du fichier	
	`6003	**Article du fichier plus long :** Seul le début de l'article est chargé	
	`6004	**Article du fichier plus court :** l'article est chargé et le reste de la zone est inchangé	
	`6006	**Fin de chaine :** (table inverse) avant l'exécution de la primitive, le pointeur logique était déjà à la fin d'une liste inverse	
	`6007	**Début de chaine :** (table inverse) avant l'exécution de cette primitive le pointeur courant du fichier était déjà en début d'une liste inverse	
	`600A	**FAU inexistante**	
	`6018	**Incompatible primitive fichier**	
	`601A	**Erreur d'enchainement :** La dernière primitive effectuée n'a pas fait de sélection d'article.	
	`601F	**Primitive en cours**	
	`6028	**Erreur de syntaxe** (@FCB, FONCT, ABU, LBU, PAS).	
	`6029	**Adresse du FCB incorrecte**	
	`602B	**Méthode d'accès non gérée**	
Erreurs graves	`6032	**Informations système invalides**	
	`6034	"	
	`6035	**FU verrouillée par IOCS**	
	`4...	**Erreur hardware :** `4000 + mot d'état PU	

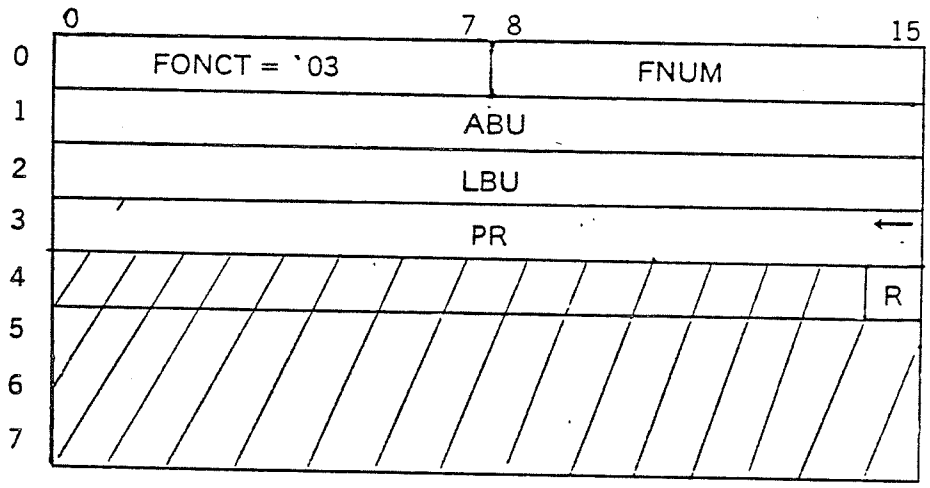
e) Réécriture de l'article courant : SIWRIT

FMS vérifie qu'il y a eu sélection d'un article et, dans le cas des tables directes, que le nom de l'article cité correspond bien à celui de l'article sélectionné.
FMS réécrit l'information dans le buffer de travail, puis sur disque.
L'utilisateur a la possibilité de demander une écriture retardée. Dans ce cas il n'y a pas écriture sur disque tant que le poste touché peut rester dans le buffer de travail. Cela permet le cumul de plusieurs modifications dans un même poste, sans pénalisation par le temps d'écriture sur disque.

Nom	SIWRIT
But	Réécriture de l'article courant

Appel RA : =@FCB ; SVC (FMSX) ; où FMSX = `28

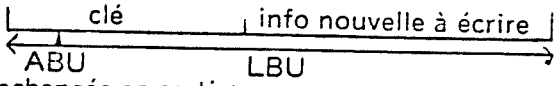
FCB



Description des paramètres

FNUM : n° d'utilisation du fichier
ABU : adresse de la zone d'échange
LBU : longueur de la zone d'échange (nombre pair d'octets ⩾ longueur de la clé)
R : indicateur d'écriture retardée
R = 1 ⇔ écriture retardée
PR : paramètre de retour

La zone d'échange
— En entrée



— Inchangée en sortie

Comptes Rendus	Valeur du PR	Signification	SIWRIT
	0	Primitive exécutée normalement	
	`6003	Article du fichier plus long : seul le début de l'article est réécrit	
	`6004	Article du fichier plus court : l'article est réécrit. La fin de la zone d'échange est ignorée	
	`6005	Enregistrement incorrect : la clé donnée ne correspond pas à celle de l'article courant	
	`600A	FAU inexistant	
	`6014	Protection écriture	
	`6018	Incompatibilité primitive fichier	
	`601A	Erreur d'enchainement : la primitive précédente n'a pas sélectionné d'article	
	`601F	Primitive en cours	
	`6028	Erreur de syntaxe : (@FCB, FONCT, ABU, LBU).	
	`6029	Adresse du FCB invalide	
	`602B	Méthode d'accès non gérée	
Erreurs graves	`6032	Informations système invalides	
	`6034	"	
	`6035	FU verrouillée par IOCS	
	`4...	Erreur hardware : `4000 + mot d'état PU	

f) Addition d'un nouvel élément : SIADD

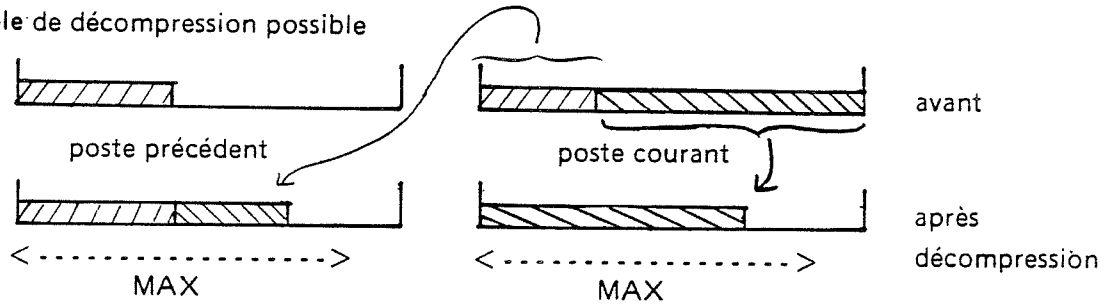
Après parcours à travers les niveaux de tables d'index, le traitement est différent selon le type de table

Cas des tables directes

FMS insère l'article au niveau information de manière à respecter le tri selon les noms. En cas d'homonymie, FMS insère le nouvel article derrière ses homonymes et donne un compte-rendu d'homonymie

Si le poste où doit se placer l'article est plein, FMS cherche à le décharger partiellement dans le poste précédent c'est l'algorithme de **"décompression"**. Pour qu'il y ait décompression il faut qu'après avoir réparti également les articles dans les deux postes, le remplissage de chacun de ces postes ne dépasse pas une borne fixée par l'utilisateur : MAX article.

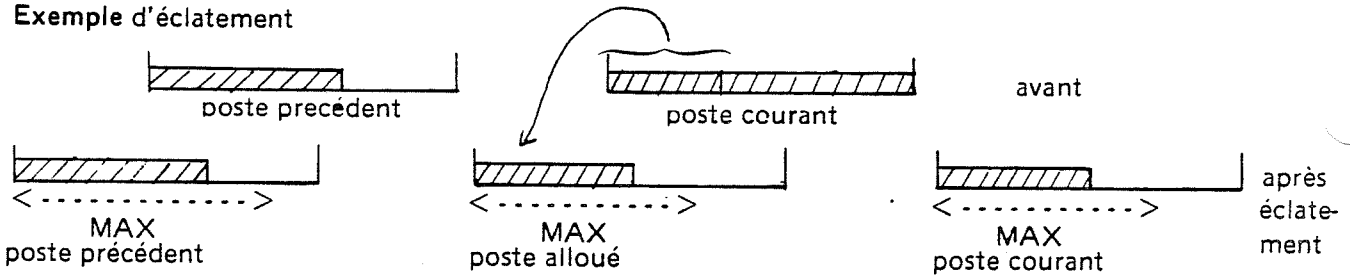
Exemple de décompression possible



Si la décompression n'est pas possible (il n'y a pas de poste précédent ou celui-ci est trop plein), FMS-E provoque un **"éclatement"**.

L'éclatement consiste à allouer un nouveau poste, à le placer devant le poste courant et à répartir également les articles dans le poste alloué et le poste courant.

Exemple d'éclatement



Remarque :

Les deux schémas précédents n'individualisent pas les articles. Seuls sont représentés les remplissages des postes. Cependant le transfert d'un poste dans un autre respecte les frontières d'article.

La borne MAX permet de paramètrer le taux moyen de remplissage du fichier.

L'utilisateur détermine le taux de remplissage suivant la volatilité de son fichier.

En effet il est souhaitable de pouvoir laisser une marge de place libre dans chaque poste afin de permettre un certain nombre d'additions sans saturer à nouveau le poste. Plus la volatilité est grande plus il est conseillé de laisser une marge relativement importante. Le domaine de valeurs conseillées pour ce taux est l'intervalle (75%, 95%). Cependant étant donné le dynamisme du fichier les valeurs données par l'utilisateur ne correspondent pas à tout instant avec le taux de remplissage réel du fichier. L'utilisateur devra prévoir, lors du dimensionnement de son fichier un taux de remplissage moyen de 75%.

Exemple moyen : MAX ≡ 80% soit pour un poste pouvant contenir 20 articles MAX = 16

Remarque :

Dans les postes de niveau supérieur (tables d'index) la borne MAX est systématiquement choisie par le système à sa plus grande valeur.

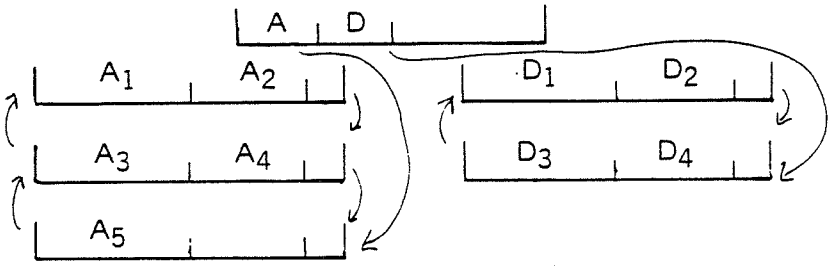
Cas des tables inverses

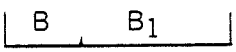
Lors de l'addition d'un nouvel article deux situations peuvent se présenter :

- c'est le premier article ayant un tel nom, FMS insère alors le nouveau nom dans le niveau 1 et alloue un nouveau poste au niveau 0 où il stocke l'information de l'article. C'est la création d'une nouvelle liste inverse.
- il existe des articles ayant même nom, FMS ajoute alors l'information du nouvel article à la suite de celles existant déjà dans la liste inverse. Si le dernier poste de cette liste était saturé, FMS alloue un nouveau poste. Il le chaine à la fin de la liste, et y stocke la nouvelle information. Aucun taux de remplissage n'est alors pris en compte. Les postes du niveau information sont remplis jusqu'à saturation puisqu'il n'y a pas d'insertion au milieu d'une liste inverse.

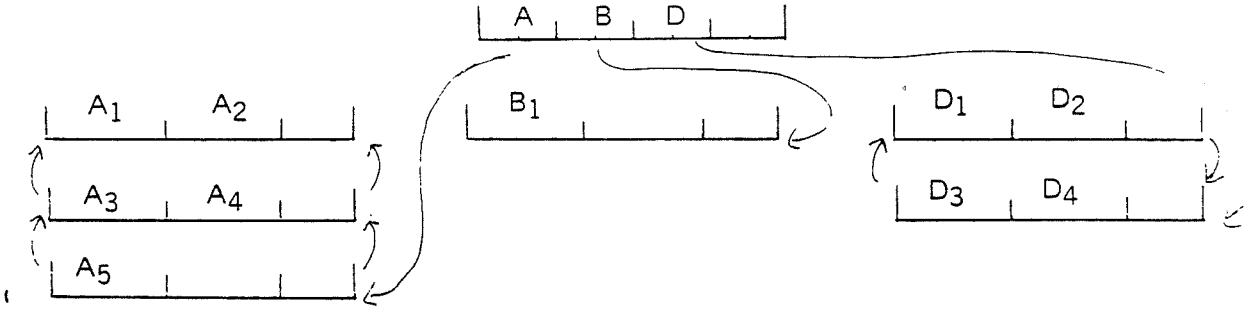
Exemple d'additions

Soit un fichier table inverse de ce type

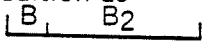


Supposons une addition avec dans la zone d'échange : 

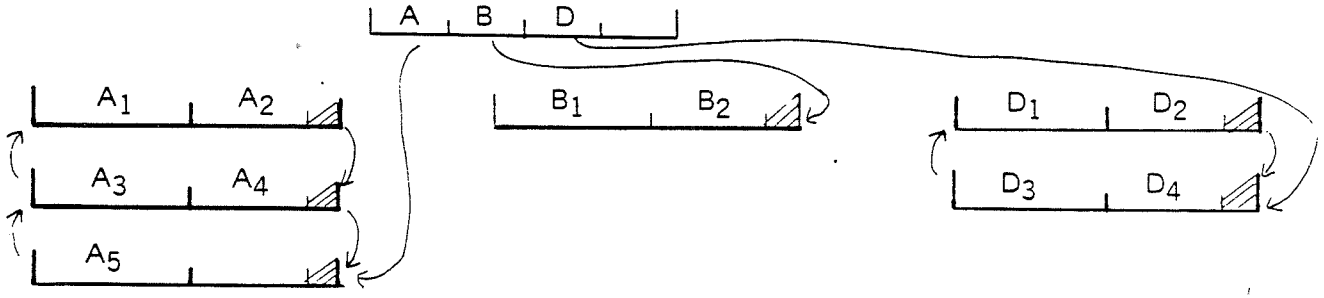
Il y a création d'une liste inverse :



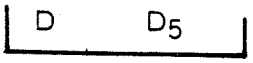
S'il arrive alors une addition de



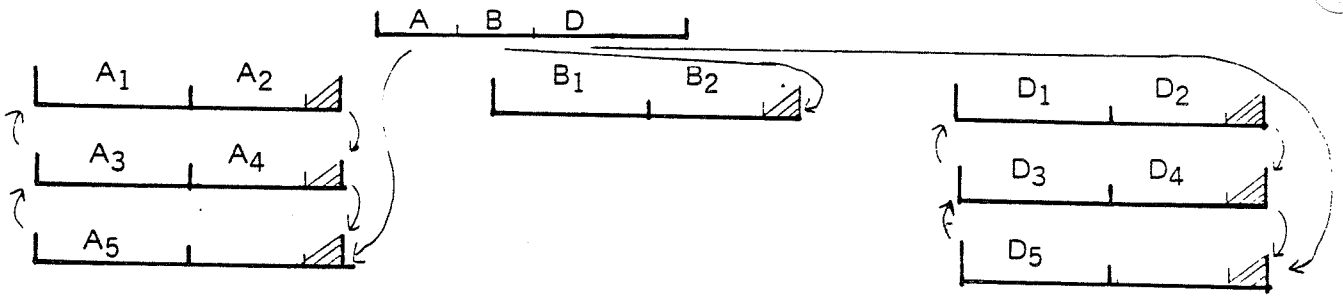
le fichier devient



Enfin avec l'addition de



on obtient



Remarque :

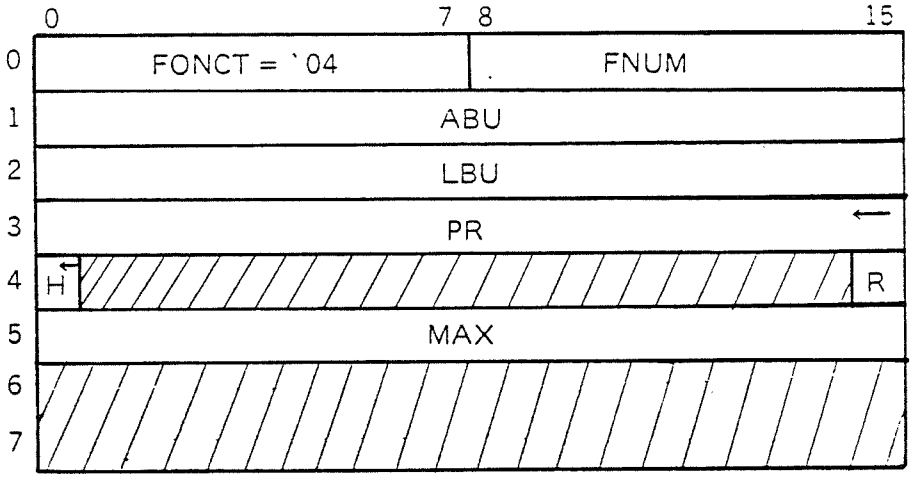
D_i représente l'information du iième article ajouté de nom D. Cela ne préjuge en rien du contenu de cette information.

Les petites zones hachurées représentent les informations système d'un poste.

Nom	**SIADD**
But	Ajout d'un nouvel article

Appel RA : =@FCB ; SVC (FMSX) ; où FMSX = `28

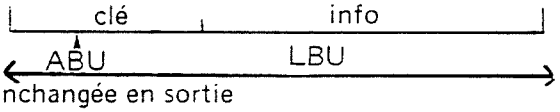
FCB



Description des paramètres

FNUM : nº d'utilisation du fichier
ABU : adresse de la zone d'échange
LBU : longueur de la zone d'échange (nombre pair d'octets ⩾ longueur de la clé)
R : indicateur d'écriture retardée
R = 1 ⇔ écriture retardée
MAX : nombre maximum d'éléments dans un poste après réorganisation
MAX < nombre d'article par poste.
PR : paramètre de retour
H : compte-rendu d'homonymie
H = 1 ⇔ il existe des articles de même nom

La zone d'échange
— En entrée



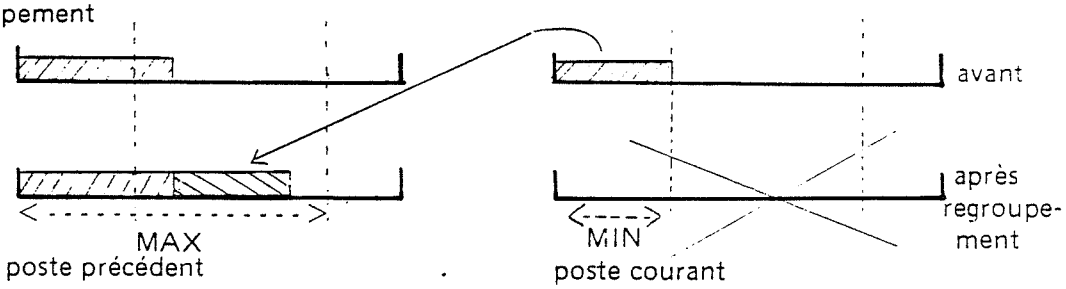
— Inchangée en sortie

Comptes Rendus	Valeurs du PR	Signification	SIADD
	0	Primitive exécutée normalement	
	`6003	**Article du fichier plus long :** seul le début de l'article est initialisé avec le contenu de cette zone	
	`6004	**Article du fichier plus court :** seul le début de la zone déchangée est écrit dans l'article. Le reste est ignoré	
	`600A	**FAU inexistante**	
	`600F	**Article existant :** (dans une table sans homonymes) un article de ce nom existe déjà dans le fichier	
	`6014	**Protection écriture**	
	`6016	**Fichier saturé :** le nombre maximum de postes demandé à la créatic est atteint ou le fichier dépasse 5 niveaux	
	`6018	**Incompatibilité primitive fichier**	
	`601F	**Primitive en cours**	
	`6028	**Erreur de syntaxe :** Code fonction erroné, ou zone d'échange hors limite (calculateur ou SLE) ou MAX non compris entre la moitié et le total — 1 du nombre d'articles par poste, ou zone d'échange plus petite qu'un nom d'article.	
	`6029	**Adresse du FCB incorrecte**	
	`602B	**Méthode d'accès non gérée**	
Erreurs graves	`6032	**Informations système invalides**	
	`6034	"	
	`6035	**FU verrouillée par IOCS**	
	`4 . . .	**Erreur hardware :** `4000 + mot d'état PU	

g) **Suppression de l'article courant : SISUP**

Cette primitive n'est acceptée qu'après une primitive de sélection SIREAD, SIRIS, SIWRIT.
FMS vérifie l'identité du nom de l'article sélectionné et du nom fourni
Il supprime l'article et tasse le poste. Si celui-ci est vide, il est désalloué.
Cependant la désallocation peut être provoquée plus tôt à l'aide du paramètre MIN donné par l'utilisateur, qui représente le plus petit nombre d'articles que l'utilisateur accepte dans un poste.
Lorsque, après suppression il reste moins de MIN articles dans un poste. FMS lit le poste précédent et tente un **"regroupement"**.
Le regroupement consiste à reporter les articles du poste courant dans le poste précédent. Il se fait si, comme dans l'addition, le remplissage du poste précédent ne dépasse alors pas MAX articles. Le poste courant est désalloué

Schéma d'un regroupement



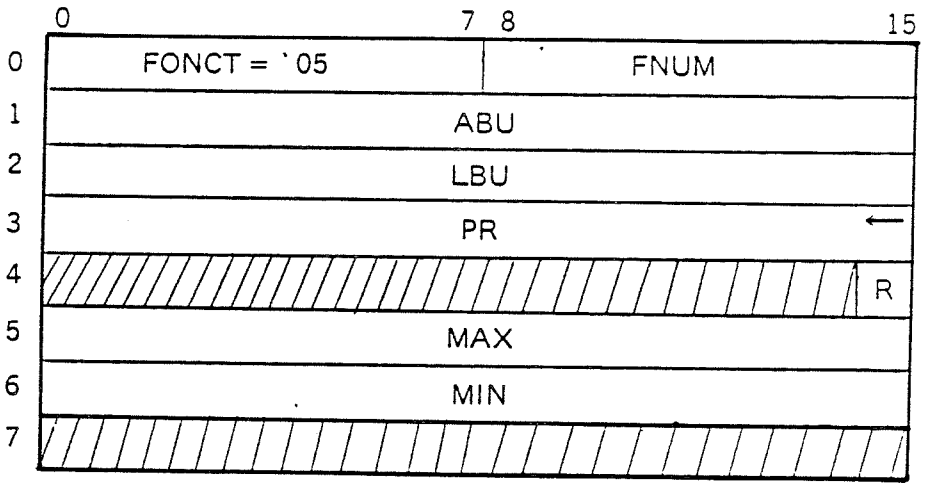
Si le regroupement n'a pu avoir lieu, le poste courant, bien que faiblement rempli, subsiste :
FMS évite la libération d'un poste si elle provoque la saturation d'un autre poste.
L'utilisateur peut donner MIN = 0 s'il préfère minimiser les temps d'accès quitte à augmenter la taille de son fichier.
S'il veut avoir un meilleur taux de remplissage il augmente la valeur de MIN. Cette valeur ne doit cependant pas dépasser $\frac{MAX}{2}$ sous peine de provoquer de nombreuses tentatives de regroupement ineffectives.
Exemple moyen MAX ≡ 80% MIN ≡ 20%
Soit pour un poste pouvant contenir 20 articles
MAX = 16 MIN = 4

Aux niveaux supérieurs il n'y a pas de regroupements.
Dans une table inverse FMS ne vérifie pas le nom de l'article et les regroupements ne se font qu'à l'intérieur d'une même liste inverse.
Comme pour toute primitive d'écriture, l'utilisateur peut demander une écriture retardée qui minimise les écritures sur disque surtout lorsque le buffer de travail peut contenir plusieurs postes.

Nom	**SISUP**
But	Suppression de l'article courant

Appel RA : =@FCB; SVC (FMSX) : où FMSX = `28

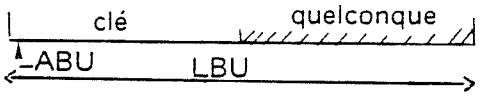
FCB



Description des paramètres

FNUM : n° d'utilisation du fichier
ABU : adresse de la zone d'échange
LBU : longueur de la zone d'échange (nombre pair d'octets ⩾ longueur de la clé)
R : indicateur d'écriture retardée
R = 1 écriture retardée
R = 0 écriture immédiate sur disque
MAX : nombre maximum d'articles dans un poste après réorganisation
MAX < nombre d'article par poste
MIN : nombre minimum d'articles dans un poste en respectant la règle $0 \leqslant MIN \leqslant \frac{MAX}{2}$
PR : paramètre de retour

La zone d'échange :
— En entrée :



— En sortie : inchangée

Comptes Rendus	Valeur du PR	Signification	SISUP
	0	Primitive exécutée normalement	
	`6005	Enregistrement incorrect : la clé donnée n'est celle de l'article courant	
	`600A	FAU inexistante	
	`6014	Protection écriture	
	`6018	Incompatibilité primitive fichier	
	`601A	Enchainement incorrect : la primitive précédente n'a pas sélectionné d'article (ex : SIADD, SISUP)	
	`601F	Primitive en cours	
	`6028	Erreur de syntaxe : ex : code fonction erroné, zone d'échange en dehors des limites du calculateur ou de SLE en mode esclave, MAX non compris entre la moitié et le total — 1 du nombre d'articles par poste, MIN plus grand que la moitié du nombre d'articles par poste ou que MAX	
	`6029	Adresse du FCB incorrecte	
	`602B	Méthode d'accès non gérée	
Erreurs graves	`6032	Information système invalides	
	`6034	"	
	`6035	FU verrouillée dans IOCS	
	`4 . . .	Erreur hardware : `4000 + mot d'état PU	

PR	Signification	OPEN NEW	OPEN OLD	CREAT	SIREAD	SIRIS	SIWRIT	SIADD	SISUP	
`6001	Fin de Fichier					X				avertissements (il y a sélection)
`6002	Début de Fichier					X				
`6003	Article plus long que la zone d'échange				X	X	X	X		
`6004	Article plus court que la zone d'échange				X	X	X	X		
`6005	Enregistrement incorrect						X		X	
`6006	Fin de Chaine					X				
`6007	Début de Chaine					X				
`600A	FAU inexistante				X	X	X	X	X	INART provoque la sélection les autres erreurs laissent inchangée la sélection antérieure
`600B	FAU existante	X	X	X						
`600C	Fichier inexistant		X							
`600D	Fichier existant	X		X						
`600E	Article inexistant				X					
`600F	Article existant							X		
`6014	Protection Ecriture		X				X	X	X	
`6015	Fichier permanent de nature différente		X							
`6016	Fichier saturé							X		
`6017	Fichier trop long	X		X						
`6018	Primitive incompatible avec le type de fichier				X	X	X	X	X	
`601A	Enchainement de primitives incorrect					X	X		X	
`601E	Fichier occupé		X							
`601F	Primitive en cours	X	X	X	X	X	X	X	X	limite des erreurs transmises
`6020	Zone de pavés système saturée	X	X	X						à l'utilisateur sous BOS16, RTES16
`6021	FU saturée	X		X						
`6022	Zone de descripteurs de fichiers saturée			X						
`6028	Erreur de syntaxe	X	X	X	X	X	X	X	X	
`6029	Adresse du FCB incorrecte	X	X	X	X	X	X	X	X	
`602A	SU ou FU inconnue de FMS-E	X	X	X						
`602B	Méthode d'accès non gérée	X	X	X	X	X	X	X	X	
`6032	Information système invalide sur disque	X	X	X	X	X	X	X	X	erreurs graves annulent la sélection
`6033	// // // //	X		X						
`6034	Information système invalide en mémoire	X	X	X	X	X	X	X	X	
`6035	FU verrouillée dans IOCS	X	X	X	X	X	X	X	X	
`4...	Erreur hardware `4000 + mot d'état PU	X	X	X	X	X	X	X	X	

SYNOPTIQUE D'ERREURS

9 — LE FICHIER DE TYPE SEQUENTIEL CHAINE

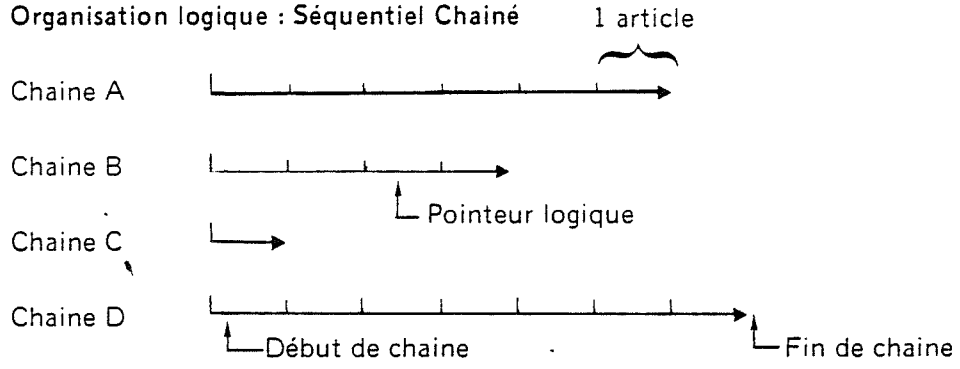
9 — LE FICHIER DE TYPE SÉQUENTIEL CHAINÉ

9.1 - ORGANISATION LOGIQUE

9.1.1 - Description générale

L'organisation logique d'un fichier de type séquentiel chainé est définie à la création du fichier, elle posséde le numéro 4.

Organisation logique : Séquentiel Chainé



On peut distinguer deux niveaux logiques dans un fichier séquentiel chainé :
— les chaines : un fichier est un ensemble de chaines
— les articles : une chaine est une suite séquentielle d'articles.

Unechaine est identifiée par deux types d'informations :
— ID1, ID2 : un identificateur binaire sur 2 mots, défini par l'utilisateur à la création de la chaine.
— AR1, AR2 : un couple d'adresses défini par FMS à chaque addition d'un article dans la chaine.

Remarques importantes :
— la valeur de l'identificateur utilisateur ID1, ID2 est constante pour une chaine donnée. C'est pour l'utilisateur le **nom** de la chaine. Plusieurs chaines peuvent avoir le **même** nom.
— la valeur de l'identificateur système AR1, AR2 est variable pendant la vie d'une chaine. C'est pour FMS-E l'information qui permet l'**accès** direct à la chaine et que l'utilisateur doit fournir dans toutes les primitives du niveau article. ID1, ID2 étant seulement un argument de contrôle.

L'utilisateur doit donc mémoriser, par exemple dans un autre fichier, pour chaque chaine la valeur du couple AR1, AR2 et le mettre à jour à chaque addition d'article dans la chaine.

A sa création le fichier ne contient aucune chaine. L'utilisateur crée une chaine lorsqu'il crée le premier article de la chaine.
Une chaine est constituée d'un nombre variable d'articles de taille fixe pour un fichier donné.

L'organisation logique d'une chaine est séquentielle c'est-à-dire que les articles se trouvent rangés dans l'ordre chronologique de création dans la chaine. A ce titre une chaine peut donc être assimilée à un "fichier séquentiel".
L'addition d'un article dans une chaine signifie que l'on ajoute un article à la fin de la chaine.

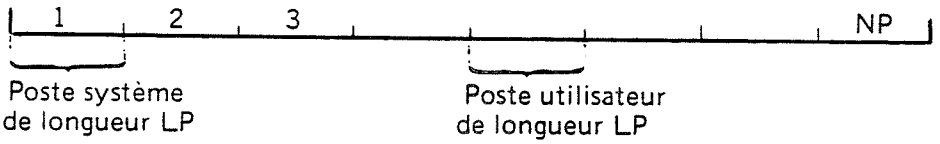
La taille d'un fichier séquentiel chainé est fixe et définie à sa création, le nombre et la taille des chaines sont limités par la taille du fichier.

9.1.2 - Structure physique

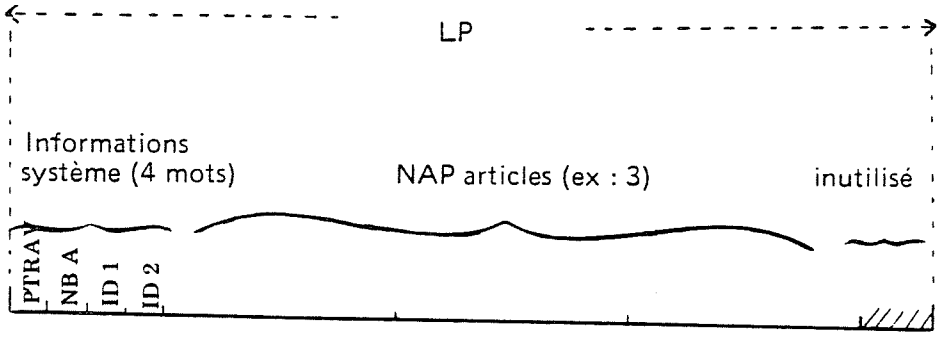
Un fichier Séquentiel Chainé est physiquement structuré en postes de taille fixe.
Le poste constitue l'unité d'allocation de place pour les chaines d'un fichier.
— les postes d'une même chaine sont chainés entre eux.
— les postes libres sont également chainés entre eux.

Lorsque l'utilisateur crée un fichier Séquentiel Chainé il définit la longueur du poste (LP), le nombre de postes (NP), et le nombre d'articles par poste (NAP).

Séquentiel Chainé : structure physique



Séquentiel Chainé : structure d'un poste utilisateur



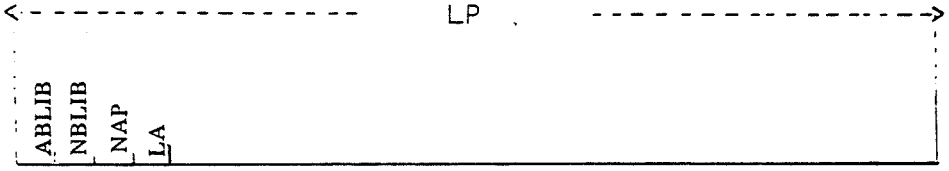
PTRAV : pointeur aval : adresse du poste suivant dans la chaine (adresse en 1/2 secteur). Le pointeur aval du dernier poste de la chaine adresse le poste lui-même.

NBA : Dans le dernier poste d'une chaine ;
NBA = nombre d'articles de la chaine.
Dans le premier poste d'une chaine multi-poste : NBA = 0

ID1, ID2 : le nom de la chaine à laquelle appartient le poste.

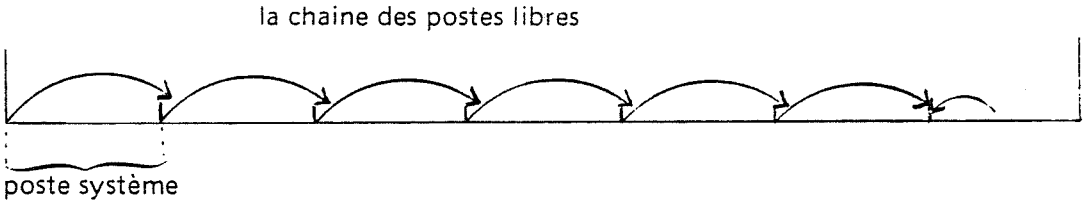
NAP articles (ici 3) :
Taille des articles en mots = quotient : (LP/2 — 4) / NAP
le reste de la division indique le nombre de mots inutilisés dans le poste.

Séquentiel Chainé : structure du poste système



ABLIB = adresse du premier poste de la chaine des postes libres (adresse en 1/2 secteur).
NBLIB = nombre de postes de la chaine des postes libres.
NAP = nombre d'article par poste.
LA = longueur des articles, en mots

Séquentiel Chainé : structure à la création.



Un fichier Séquentiel Chainé est créé avec une organisation physique directe. Toute la place disque nécessaire au fichier est allouée dès sa création.

Contrôle des paramètres de création

— le fichier doit occuper au maximum 128 granules.

La taille maximum d'un fichier Séquentiel Chainé est environ de 2Mmots ($\simeq$ 2 090 000).

Lorsque la taille du granule est supérieure à 16 K mots, la FU-support permet de stocker le plus grand des fichiers Séquentiel Chainé.

LP : **la longueur du poste (en octets)**, est un nombre entier de demi-secteur.

— LP = K x 128 octets

— $128 \leqslant LP \leqslant 95 \times 128 < 12$ K mots

NP : le nombre de poste

— soit K la longueur du poste en 1/2 secteur.
K = LP/128

— alors : K x NP < 32 K

Exemple :

— Si LP = 1 demi-secteur
2 ⩽ NP ⩽ 32 767

— Si LP = 95 demi-secteur
2 ⩽ NP ⩽ 344

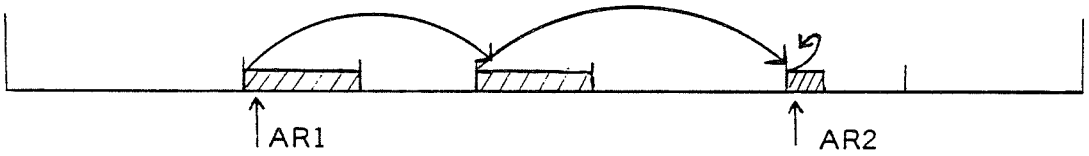
NAP : le nombre d'articles par poste

— NAP < LP/2 — 4

Structure d'une chaine

Une chaine est physiquement constituée de 1 à NP — 1 postes.
L'unité d'incrementation physique de la chaine est le poste.
Lorsque l'utilisateur crée unarticle dans une chaine (AR1, AR2, ID1, ID2) et que le dernier poste de la chaine est saturé (il contient NAP articles), FMS alloue un nouveau poste à la fin de la chaine et y place l'article à créer. FMS rend à l'utilisateur la nouvelle valeur du couple AR1, AR2 (AR2 a été modifié).

Exemple après allocation d'un poste pour créer un article.



AR1 et AR2 sont des adresses en 1/2 secteur relatives au début du fichier.

Les 2 premiers postes de la chaine sont pleins. Le 3ème poste contient un seul article celui qui vient d'être créé.

L'allocation d'un poste à une chaine se fait au fur et à mesure des besoins des différentes chaines. Les postes d'une même chaine ne sont généralement pas adjacents, sauf si les chaines sont créées en une seule fois (n primitives d'addition d'article SCADD) et l'une après l'autre.

Remarque :

Une chaine peut contenir de 1 à 32 767 articles.

9.2 - METHODE D'ACCES (SCH)

Le Séquentiel Chainé est une méthode d'accès capable de gérer des Unités d'Accès à des fichiers Séquentiel Chainé (Organisation logique n° 4).

Le séquentiel Chainé fournit 4 requêtes d'accès au niveau article. Il permet d'utiliser les 6 requêtes de la méthode d'accès Séquentiel selon le mode séquentiel pur statique.

9.2.1 - Accès aux articles

(a) Création d'une chaîne et du premier article

La primitive SCADD permet de créer une chaîne et le premier article de la chaîne.
L'utilisateur fournit l'article dans une zone d'échange.
Pour indiquer qu'il veut créer une nouvelle chaîne de nom ID1, ID2, il fournit à FMS des paramètres AR1 =AR2 = 0.
FMS alloue un poste pour cette nouvelle chaîne et y place l'article de l'utilisateur.
FMS rend dans le FCB la valeur du couple AR1, AR2 qui permettra par la suite l'accès direct à la chaîne (AR1 = AR2 = l'adresse relative du poste en 1/2 secteur).

(b) Addition d'un article dans une chaîne existante

La primitive SCADD permet d'ajouter un article à la fin d'une chaîne existante:
L'utilisateur fournit l'article dans une zone d'échange.
La chaîne est identifiée par : AR1, AR2, ID1, ID2. FMS rend à l'utilisateur dans le FCB :

- le nombre d'articles de la chaîne:
- la valeur du couple AR1, AR2 pour cette chaîne.

Attention :

la valeur de AR2 est modifiée lorsqu'il y a eu allocation d'un nouveau poste à la fin de la chaîne.

(c) Lecture séquentielle d'un article

La primitive SCREAD permet de lire en séquentiel les articles d'une chaîne identifiée par AR1, AR2, ID1, ID2.

L'utilisateur fournit un paramètre. ORDR qui indique le pas de la lecture séquentielle. Ainsi d'une façon générale SCREAD permet de lire le $n^{ième}$ **article suivant** dans une chaîne.

- lorsque ORDR = 1 et pour la même chaîne, la requête effectue la lecture de l'article suivant.
- lorsque ORDR = 2 et pour la même chaîne, la requête effectue la lecture du 2e article suivant (lecture séquentielle de 2 en 2).

d Réécriture de l'article en cours

La primitive SCWRIT permet de réécrire l'article en cours, c'est-à-dire l'article qui vient d'être créé, lu ou réécrit. L'utilisateur fournit dans la zone d'échange le nouveau contenu de l'article.
FMS contrôle à l'aide de AR1, AR2 que la réécriture est bien demandée sur la chaîne en cours.

e Lecture directe d'un article

La primitive SCDREAD permet de lire en accès direct (1 accès disque avec l'option ADR, 2 si non), le $n^{ième}$ article d'une chaîne. La chaîne est identifiée par AR1, AR2, ID1, ID2. Le $n^{ième}$ article est identifié par ARP, ORDR

- ARP est l'adresse relative du poste contenant l'article à lire.
- ORDR est le rang de l'article dans la chaîne ou dans le poste.

Les valeurs ARP et ORDR sont fournies à l'utilisateur lors de la création de l'article par la primitive

SCADD :

- ARP = AR2 fourni par SCADD
- ORDR = ORDR fourni par SCADD

L'utilisateur doit donc sauvegarder ces valeurs lors de la création de l'article pour pouvoir y accéder directement par la suite.

La primitive SCDREAD sélectionne la chaîne et l'article, pour que l'utilisateur puisse le réécrire (SCWRIT) ou lire la fin de la chaîne séquentiellement (SCREAD).

(f) Remarques

Le Séquentiel Chainé ne permet pas de supprimer un article ou une chaine. L'utilisateur devra donc procéder comme avec les fichiers séquentiels classiques :

- marquage de suppression (SIWRIT)
- recopie d'une chaine dans une autre chaine ou d'un fichier dans un autre fichier par les requêtes du Séquentiel Chainé.

Pour fonctionner la méthode d'accès Séquentiel Chainé demande à l'utilisateur de lui fournir un buffer de travail de la longueur du poste.
Ce buffer est utilisé, pour créer le contenu initial du fichier, CREAT, OPEN NEW au niveau fichier, et pour exécuter les requêtes du niveau article du Séquentiel Chainé.
Le buffer de travail est fourni lors des requêtes de création d'une Unité d'Accès au fichier : CREAT, OPEN NEW, OPEN OLD.

9.2.2 - Accès séquentiel pur statique

La méthode d'accès permet l'accès séquentiel pur à tout le fichier.
Ceci ne peut se faire qu'à l'aide de la méthode d'accès séquentiel, après l'une des trois primitives de création d'une Unité d'Accès (CREAT, OPEN NEW, OPEN OLD) et avant toute requête du niveau article du Séquentiel Chainé.
Tout le fichier est accessible (lecture, réécriture), séquentiellement selon sa **structure physique.** Après CREAT ou OPEN NEW le pointeur courant adresse la fin physique du fichier.
Après OPEN OLD le pointeur courant adresse le début physique du fichier c'est-à-dire le début du poste système.
Ce type d'accès est principalement utilisé pour des archivages de fichiers.

9.3 - FONCTIONS LOGIQUES

9.3.1 - Le niveau fichier

(a) Généralités

Les requêtes du niveau Fichier pour un Sequentiel Chainé sont semblables à celles des autres méthodes d'accès. Leur fonctionnement est décrit dans le chapitre 4 L'ENTITE FICHIER.
Il n'est décrit ici que les particularités liées à la méthode d'accès Séquentiel Chainé.
Elles ne concernent que les requêtes :
CREAT, OPEN NEW, OPEN OLD

L'interface d'appel d'une requête du niveau fichier est en PL1600 :

```
RA : =@FCB ;
SVC (FMS) ;          << FMS = '38
```

(b) CREAT, OPEN NEW : création

Les paramètres spécifiques de la création d'un fichier séquentiel chainé sont :
- Organisation logique nº 4 EMA/SID
- La longueur du poste LP
- Le nombre de postes NP
- Le nombre d'articles par poste NAP
- Un buffer de travail : ABU, LBU, BUF.

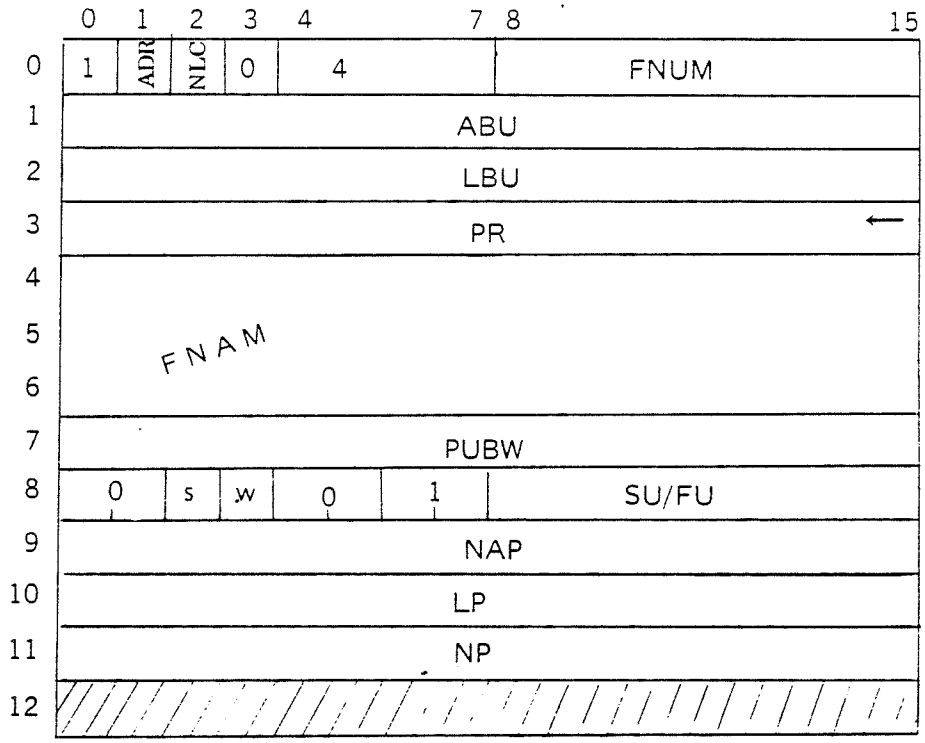
FMS alloue à la création la place nécessaire à tout le fichier sur le support spécifié par l'utilisateur. Un fichier Séquentiel Chainé est créé avec une organisation physique directe. Il ne doit donc pas occuper plus de 128 granules (voir chapitre 8.1.2 : Structure Physique).

La création d'un fichier Séquentiel Chainé positionne le pointeur courant à la fin physique du fichier.

Nom	**CREAT**	**Séquentiel Chainé**
But	Créer un fichier permanent Séquentiel Chainé et créer une Unité d'Accès à ce fichier.	

Appel RA : =⌒FCB ; SVC (FMS) ; << FMS = '38

FCB



Description des paramètres

ABU : est l'adresse (16 bits) du buffer de travail
LBU : est la longueur en octets du buffer de travail : LBU = LP
NAP : nombre d'articles par poste
LP : longueur du poste en octets
NP : nombre de postes

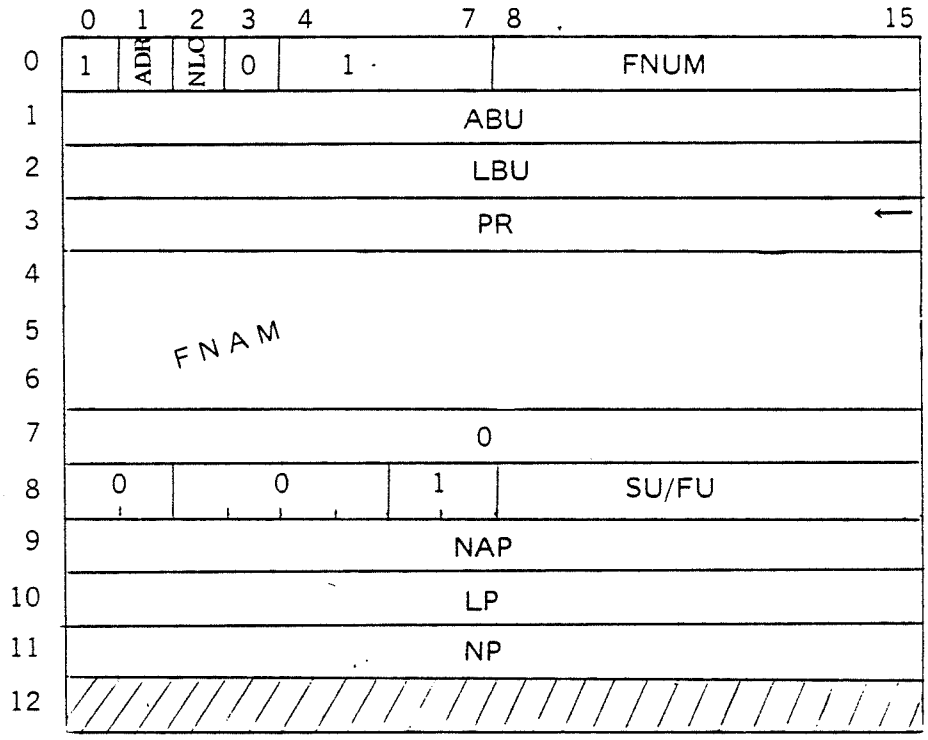
Voir dans le chapitre 9.1.2 : Structure Physique, le domaine de validité des 3 paramètres : NAP, LP, NP.

Comptes Rendus	Valeur du PR	Signification	CREAT Séquentiel Chainé
	0	Primitive exécutée correctement	
	`600B	FAU existante	
	`600D	Fichier existant	
	`6017	Fichier trop long : le fichier ne tient pas sur 128 granules compte tenu de la taille du fichier (LP x NP) et de la taille des granules définie sur le support désigné par SU/FU. La primitive est ineffective.	
	`601F	Primitive en cours	
	`6020	Zone de pavés saturée	
	`6021	FU saturée :le nombre de granules libres sur le support désigné par SU/FU ne permet pas d'allouer la place nécessaire à tout le fichier. La primitive est ineffective.	
	`6022	Table des fichiers saturée	
	`6028	Erreur de syntaxe : (ρ FCB, FONCT, ABU, LBU, FNAM, PUBW, FTYP NAP, LP, NP).	
	`6029	Adresse de FCB invalide	
	`602A	SU ou FU non gérée par FMS ou IOCS	
	`602B	Méthode d'accès non gérée	
Erreurs graves	`6032	Informations système invalides.	
	`6033	"	
	`6034	"	
	`6035	FU verrouillée par IOCS	
	`4 . . .	Erreur hardware : `4000 + mot d'état PU	

Nom	OPEN NEW	Séquentiel chainé
But	Créer un fichier temporaire Séquentiel Chainé et créer une Unité d'Accès à ce fichier.	

Appel RA : =FCB ; SVC (FMS) ; << FMS = '38

FCB



Description des paramètres

ABU : est l'adresse (16 bits) du buffer de travail
LBU : est la longueur du buffer de travail :
LBU = LP
NAP : nombre d'articles par poste
LP : longueur du poste en octets
NP : nombre de postes.

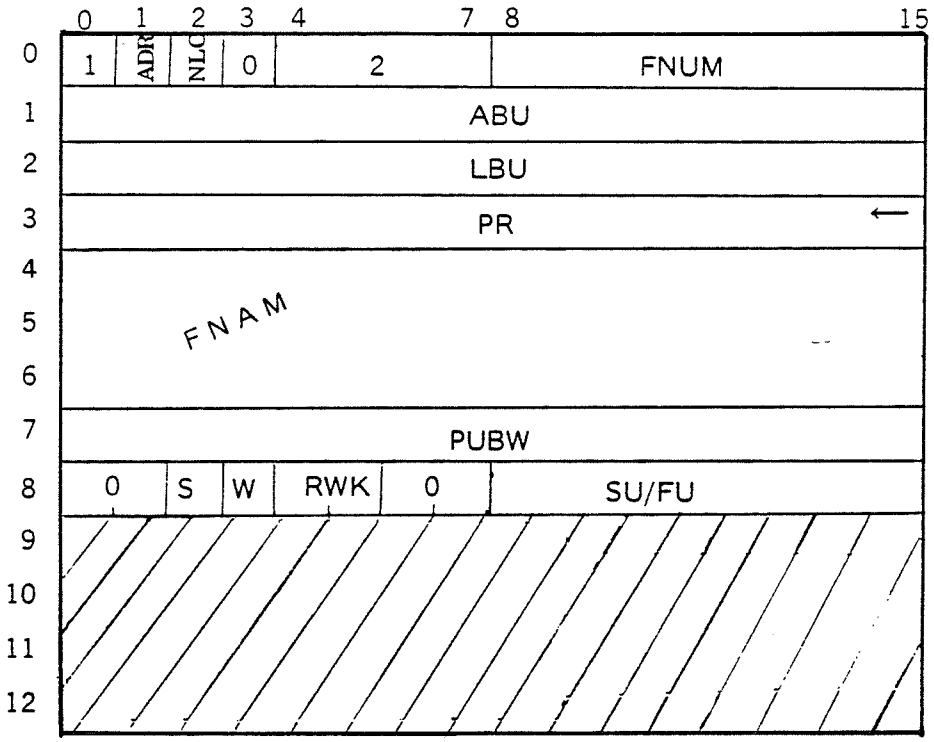
Voir dans le chapitre 9.1.2 : Structure Physique, le domaine de validité des 3 paramètres : NAP, LP, NP.

Comptes Rendus	Valeur du PR	Signification	OPEN NEW Séquentiel Chainé
	0	Primitive correctement exécutée	
	`600B	FAU existante	
	`600D	Fichier existant	
	`6017	Fichier trop long : le fichier ne tient pas sur 128 granules compte tenu de la taille du fichier (LP x NP) et de la taille des granules définie sur le support désigné par SU/FU. La primitive est ineffective.	
	`601F	Primitive en cours	
	`6020	Zone de pavés saturée	
	`6021	FU saturée : le nombre de granules libres sur le support désigné par SU/FU ne permet pas d'allouer la place nécessaire à tout le fichier. La primitive est ineffective.	
	`6028	Erreur de syntaxe : (FCB, FONCT, ABU, LBU, FNAM, FTYP, NAP LP, NP).	
	`6029	Adresse de FCB invalide	
	`602A	SU ou FU non gérée par FMS ou IOCS	
	`602B	Méthode d'accès non gérée	
Erreurs graves	`6032	Informations système invalides	
	`6033	"	
	`6034	"	
	`6035	FU verrouillée par IOCS	
	`4...	Erreur hardware : `4000 + mot d'état PU	

Nom	OPEN OLD	Séquentiel chainé
But	Créer une Unité d'Accès à un fichier permanent.	

Appel : RA : =@FCB ; SVC (FMS) ; << FMS = `38

FCB



Description des paramètres

ABU : est l'adresse (16 bits) du buffer de travail

LBU : est la longueur du buffer de travail
LBU = LP (de la création)

Comptes Rendus	Valeur du PR	Signification	OPEN OLD Séquentiel Chainé
	0	Primitive correctement exécutée	
	`600B	FAU existante	
	`600C	Fichier inexistant	
	`6014	Protection écriture	
	`6015	Permanent de nature différente	
	`601E	Fichier occupé	
	`601F	Primitive en cours	
	`6020	Zone de pavés saturée	
	`6028	Erreur de syntaxe : (@FCB, FONCT, ABU, LBU, FNAM, PUBW, FTYP)	
	`6029	Adresse de FCB invalide	
	`602A	SU ou FU non gérée par FMS ou IOCS	
	`602B	Méthode d'accès non gérée	
Erreurs graves	`6032	Informations système invalides	
	`6034	"	
	`6035	FU verrouillée par IOCS	
	`4 . . .	Erreur hardware : `4000 + mot d'état PU	

9.3.2 - Le niveau article

(a) Généralités

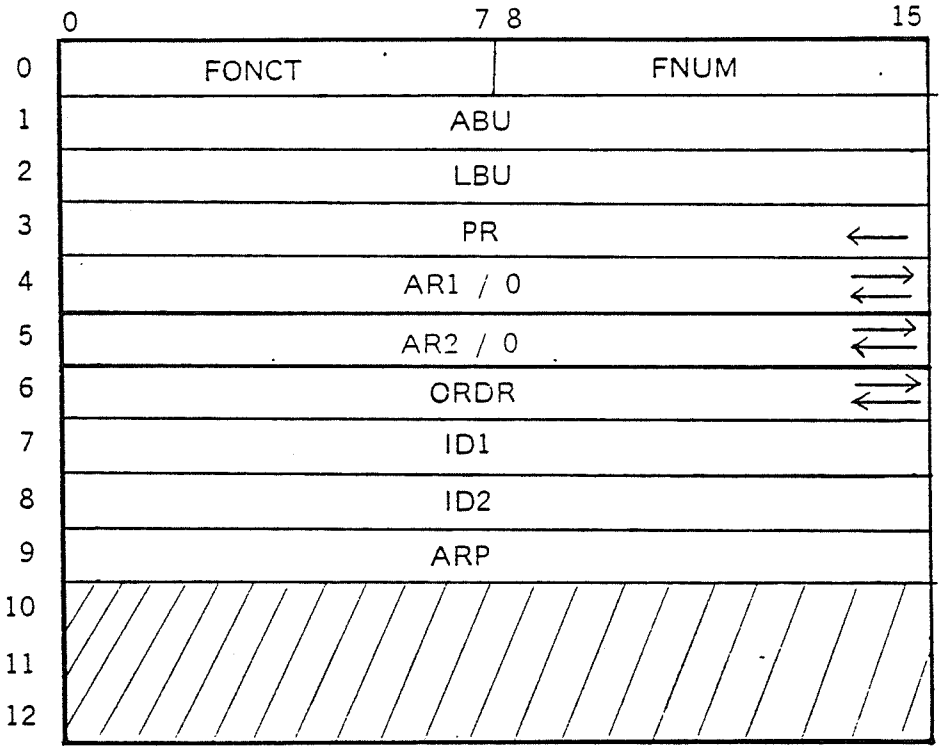
La méthode d'accès : Séquentiel Chainé fournit 4 primitives : SCADD, SCREAD, SCWRIT, SCDREAD.

L'interface d'appel d'une requête du niveau article est en PL1600

```
RA : =@FCB ;
SVC (FMSC) ;          << FMSC = ' 29
```

Description générale des FCB : Séquentiel Chainé

FCB



Conventions :

— les hachures signfient zone inutilisée par FMS
— ⟵ paramètre fourni par FMS
— ⇄ paramètre fourni par l'utilisateur et modifié par FMS
— dans les autres cas le paramètre doit être fournit par l'utilisateur.

Toute requête du niveau article s'adresse à l'Unité d'Accès spécifiée par l'utilisateur (FNUM) et concerne le fichier accessible par celle-ci.

Toute requête du niveau article interdit tout accès au fichier par la méthode d'accès séquentiel sur cette FAU et ceci jusqu'à sa destruction (de la FAU).

Le choix d'une primitive, parmi celles offertes par la méthode d'accès au niveau article, est fait par l'usager à l'aide de l'octet de fonction : FONCT.

(b) SCADD : addition d'un article

Cette primitive permet d'ajouter un article à la fin d'une chaine.
- l'ajout peut se faire dans une chaine existante spécifiée par AR1, AR2, ID1, ID2.
- l'ajout peut se faire dans une chaine à créer de nom ID1, ID2.
- le contenu de l'article est situé dans la zone d'échange spécifiée par ABU, LBU.
  LBU = longueur de l'article, si non erreur de syntaxe.

Addition d'un article avec création de chaine.
Ce que l'utilisateur doit fournir dans le FCB.
- AR1 = AR2 = 0
- ID1, ID2 = le nom de la chaine à créer

Ce que FMS rend à l'utilisateur dans le FCB
- AR1 = AR2 = adresse relative en 1/2 secteur du poste alloué à la chaine.
- ORDR = 1

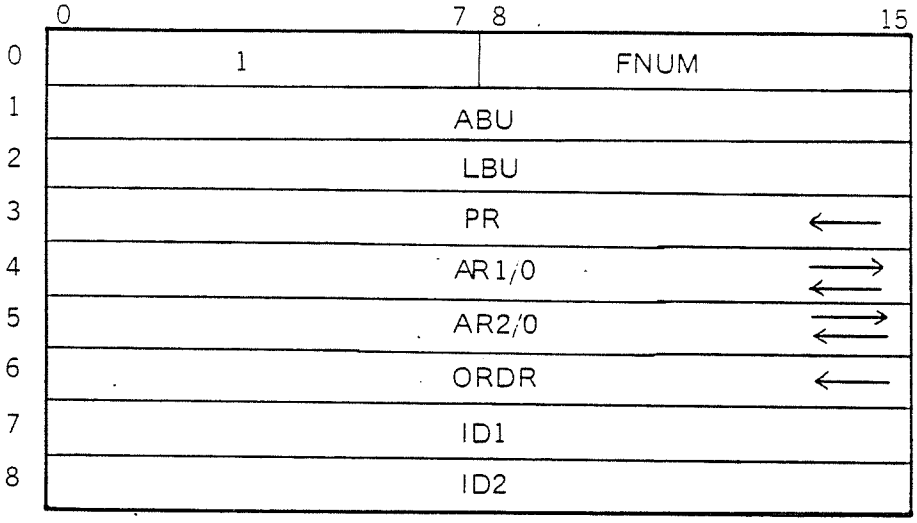
Addition d'un article dans une chaine existante
Ce que l'utilisateur doit fournir dans le FCB
- AR1, AR2 correspondant à la dernière addition dans cette chaine
- ID1, ID2 le nom de la chaine. FMS contrôle qu'à l'adresse AR2 dans le dernier poste, l'identificateur est bien le même que celui fourni par l'utilisateur dans le FCB.

Ce que FMS-E rend à l'utilisateur dans le FCB.
- AR1, AR2 le nouveau couple identifiant la chaine pour FMS. Seul AR2 est modifié lorsqu'il y a allocation d'un poste à la chaine
- ORDR le rang de l'article créé dans la chaine.

Nom	SCADD
But	Ajouter un article dans une chaine existante ou à créer.

Appel : RA : =@FCB ; SVC (FMSC) ; << FMSC = `29

FCB



Description des paramètres

FNUM : numéro de l'unité d'accès au fichier
ABU : adresse 16 bits de la zone d'échange
LBU : Longueur en octets des articles du fichier : chap:9.3
AR1 : 0 ou la valeur fournie lors de la création de la chaine et de chaque ajout d'article dans la chaine.
AR2 : 0 ou la valeur fournie par FMS-E lors de la dernière addition dans cette chaine
ID1, ID2 : identificateur binaire 2 x 16 bits le nom de la chaine pour l'utilisateur.

Comptes Rendus

Valeur du PR	Signification
0	Primitive correctement exécutée
`600A	FAU inexistante
`600E	**Chaine inexistante :** erreur sur ID1, ID2 ou AR1, AR2. Le buffer de travail est chargé par le poste d'adresse AR2. La requête est ineffective sur disque.
`6014	**Protection écriture**
`6016	**Fichier saturé :** il n'y a plus de poste libre dans le fichier
`6018	**Incomptabilité primitive-fichier**

	`601A	**Erreur d'enchainement :** chaine trop longue. Nombre d'article supérieur à 32 767. La primitive est ineffective.
	`601F	**Primitive en cours**
	`6028	**Erreur de syntaxe :** (@FCB, FONCT, ABU, LBU, AR1, AR2, ORDR, ou buffer de travail non fourni).
	`6029	**Adresse de FCB invalide**
	`602B	**Méthode d'accès non gérée**
Erreurs graves	`6032	**Informations système invalides**
	`6034	"
	`6035	**FU verrouillée par IOCS**
	`4 . . .	**Erreur hardware :** `4000 + mot d'état PU

(c) SCREAD : lecture séquentielle d'un article

Cette primitive permet de lire le nième article suivant dans une chaine identifiée par AR1, AR2, ID1, ID2. La zone d'échange spécifiée par ABU, LBU contiendra l'article lu. LBU = LA longueur de l'article sinon erreur de syntaxe.

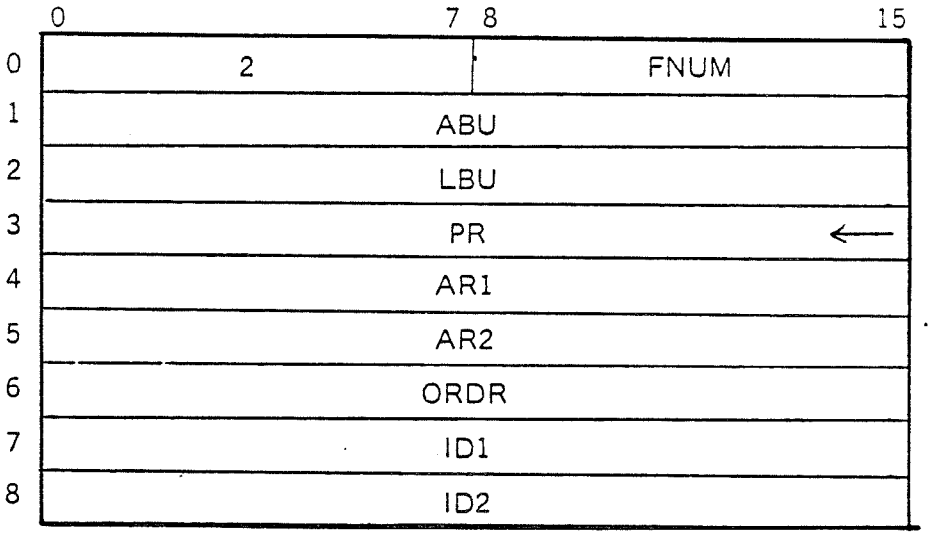
1°) ORDR = 0 réalise sélection et/ou Rewind de la chaine spécifiée par le FCB et la lecture du premier article de la chaine.

2°) Si l'utilisateur continue à lire sur cette même chaine AR1, AR2, ID1, ID2 :
ORDR = n réalise la lecture du nième article suivant.

3°) Si l'utilisateur change de chaine en spécifiant d'autres valeurs pour AR1, AR2, ID1, ID2 :
ORDR = 0 même cas que précédemment (1°))
ORDR = n réalise la lecture du nième article de cette nouvelle chaine. La lecture séquentielle de la fin de la chaine peut se poursuivre comme il est indiqué dans le 2°)

Nom	**SCREAD**
But	Lecture séquentielle du nième article suivant

Appel RA : =@FCB ; SVC (FMSC) ; ≪FMSC = '29



Description des paramètres

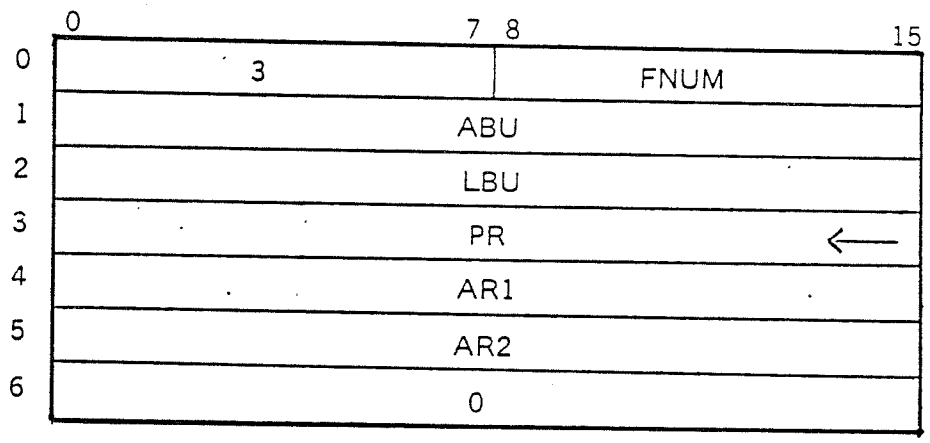
FNUM : numéro de l'unité d'accès au fichier
ABU : adresse 16 bits de la zone d'échange
LBU : longueur en octets des articles du fichier
AR1 : la valeur fournie par FMS lors de la création de la chaine, et de chaque ajout d'article dans la chaine.
AR2 : la valeur fournie par FMS lors de la dernière addition dans cette chaine
ORDR : pas de la lecture séquentielle du nième (ORDR) article suivant
ID1, ID2 : identificateur binaire 2 x 16 bits le nom de la chaine pour l'utilisateur.

Comptes Rendus	Valeur du PR	Signification SCREAD
	0	Primitive correctement exécutée
	`6006	Fin de chaine
	`600A	FAU inexistante
	`600E	Chaine inexistante : erreur sur ID1, ID2 ou AR1, AR2. Le buffer de travail est chargé avec le nième article.
	`6018	Incompatibilité primitive fichier
	`601F	Primitive en cours
	`6028	Erreur de syntaxe : (@FCB, FONCT, ABU, LBU, AR1, AR2, ORDR, ou buffer de travail inexistant)
	`6029	Adresse de FCB invalide
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032	Informations système invalides
	`6034	"
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU

ⓓ SCWRIT : réécriture d'un article

NOM	SCWRIT
But	Réécrire l'article en cours dans la chaine en cours. C'est-à-dire réécriture de l'article que l'on vient de lire SCREAD ou SCDREAD, réécrire SCWRIT ou créer SCADD.

Appel RA : = @FCB ; SVC (FMSC) ; << FMSC = '29



Description des paramètres

FNUM	:	numéro de l'unité d'accès au fichier
ABU	:	adresse 16 bits de la zone d'échange
LBU	:	longueur en octets des articles du fichier
AR1	:	la valeur fournie par FMS lors de la création de la chaine, et de chaque ajout dans la chaine
AR2	:	la valeur fournie par FMS lors de la dernière addition dans cette chaine

Comptes Rendus

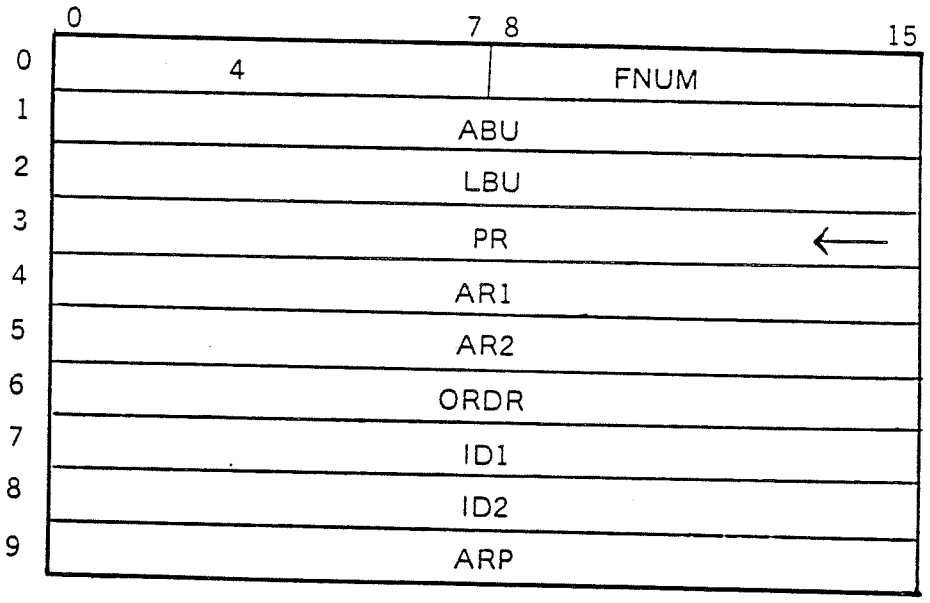
Valeur du PR	Signification
0	**Primitive correctement exécutée**
'600A	**FAU inexistante**
'6014	**Protection écriture**
'6018	**Incompatibilité primitive fichier**
'601A	**Erreur d'enchainement :** la primitive précédente correctement exécutée sur cette FAU n'est pas du niveau article, ou AR1, AR2 ne correspondent pas à la chaine en cours. La primitive est ineffective.

	`601F	Primitive en cours
	`6028	Erreur de syntaxe : (@FCB, FONCT, ABU, LBU, AR1, AR2, ou buffer de travail inexistant)
	`6029	Adresse de FCB invalide
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032	Informations système invalides
	`6033	
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU

(e) SCDREAD : Lecture directe d'un article

Nom	SCDREAD
But	Lire en accès direct le nième article d'une chaine

Appel RA : =@FCB ; SVC (FMSC) ; << FMSC = '29



Description des paramètres

- FNUM : numéro de l'unité d'accès au fichier
- ABU : adresse 16 bits de la zone d'échange
- LBU : longueur en octets de l'article
- AR1 : la valeur fournie par FMS lors de la création de la chaine, et de chaque ajout dans la chaine
- AR2 : la valeur fournie par FMS lors de la dernière addition dans cette chaine.
- ORDR : rang de l'article dans la chaine ou dans le poste, fourni par FMS-E lors de la création de l'article (ORDR).
- ID1, ID2 : identificateur binaire 2 x 16 bits le nom de la chaine pour l'utilisateur
- ARP : l'adresse du poste contenant l'article est fournie par FMS lors de la création de l'article (AR2 du SCADD correspondant)

Comptes Rendus	Valeur du PR	Signification SCDREAD
	0	Primitive correctement exécutée
	`600A	FAU inexistante
	`600E	Chaine inexistante : erreur sur ID1, ID2 ou sur AR1, AR2. Le buffer de travail est chargé et contient le nième article du poste d'adresse ARP
	`6018	Incompatibilité primitive fichier
	`601F	Primitive en cours
	`6028	Erreur de syntaxe : (@ FCB, fonct, ABU, LBU, AR1, AR2, ORDR, ARP ou le buffer de travail est inexistant).
	`6029	Adresse de FCB invalide
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032	Informations système invalides
	`6033	"
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU

(f) La sélection d'article et les comptes rendus

Le tableau suivant indique les exceptions de fonctionnement de la sélection d'article pour les requêtes du Séquentiel Chainé.

OUI : la requête est correctement exécutée. Selon la requête il y a sélection d'article. (Accès Séquentiel Pur Statique interdit).
NON : la requête est ineffective. Une éventuelle sélection en cours peut continuer
NSEL : la sélection est inchangée. Pour certains compte-rendus le buffer de travail est chargé. La requête n'est pas toujours ineffective. Accès Séquentiel Pur Statique interdit.

Le tableau suivant indique également la liste des comptes-rendus de l'ensemble des requêtes du Séquentiel Chainé.

Séq. Chainé Valeur du PR	Signification		Sélection d'article
0	Primitive correctement exécutée		OUI
`6006	Fin de chaine		NSEL (buffer)
`600A	FAU inexistante		NON
`600E	Article inexistant (chaine)		NSEL (buffer)
`6014	Protection écriture		NSEL
`6016	Fichier saturé		NSEL
`6018	Incompatibilité primitive fichier		NON
`601A	Erreur d'enchainement		NSEL
`601F	Primitive en cours		NON
`6028	Erreur de syntaxe	@FCB	NON
		FONCT	NON
		ABU	NON
		LBU	NON
		AR1, AR2	NON
		ORDR	NON
		ARP	NON
`6029	Adresse de FCB invalide		NON
`602B	Méthode d'accès non gérée		NON
`6032	Informations système invalides		NSEL
`6034	Informations système invalides		NSEL
`6035	FU verrouillée par IOCS		NSEL
`4...	Erreur hardware : `4000 + mot d'état PU		NSEL

SOMMAIRE

10 - LE FICHIER DE TYPE DIRECT LONGUEUR VARIABLE (DIRECT V)

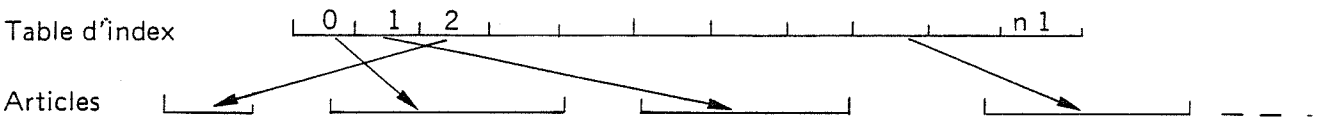
10.1 - ORGANISATION LOGIQUE

10.1.1 - Présentation

La méthode d'accès Direct Longueur Variable gère dynamiquement une suite d'articles numérotés de 0 à N—1 et de longueur variable. La table d'index située au début du fichier contient pour chaque index de rangi, l'adresse et la longueur de l'article de numéro i s'il existe. Les articles sont alloués dynamiquement à la suite de la table d'index et de plus alignés à frontière de secteur.

L'objectif de la méthode DIRECT V est surtout d'être un bibliothécaire de programmes.
L'organisation logique d'un fichier de type Direct V doit être définie à la création du fichier : elle possède le numéro 5.

Organisation logique : Direct Longueur Variable



On distingue deux parties dans un fichier Direct Longueur Variable :

— la table d'index : contenant des informations gérées par la méthode d'accès
— les articles du fichier : contenant l'information de l'utilisateur.

a) Les articles

Un fichier Direct V est composé d'un ou plusieurs articles identifiés par un numéro. Le numéro et la taille de l'article sont définis lors de la création de l'article.
Le numéro de l'article est un nombre de 0 à NART — 1 (NART = nombre maximum d'articles du fichier défini lors de la création cf § b).

FMS-E vérifie la non-homonymie des numéros d'articles : deux articles différents ne peuvent pas avoir le même numéro. La taille des articles est variable d'un article à l'autre ; elle doit cependant être un nombre pair d'octets, inférieure à 32 K mots.
La taille maximale d'un fichier Direct V est de 4 Mega-mots (32 K secteurs).

b) La table d'index

Notation : Un élément de table (relatif à un article donné) sera appelé un **index.**

La table d'index contient la longueur des articles ainsi que leur adresse d'implantation dans le fichier. Chaque index peut éventuellement contenir des informations propres à l'utilisateur.
Lors de la création du Fichier Direct V, l'utilisateur définit le nombre maximum d'articles (NART) que le fichier pourra contenir, et la longueur (LIX) en octets d'un index.

La taille de la table d'index est figée à la création du fichier par la méthode d'accès en fonction des règles suivantes :

- la taille maximum d'une table d'index est de 32 K mots
- la longueur d'un index (LIX) doit être un sous-multiple du secteur et supérieure à 4 octets. Elle est toujours inférieure à 256 octets.
- le nombre d'articles est limité de façon que

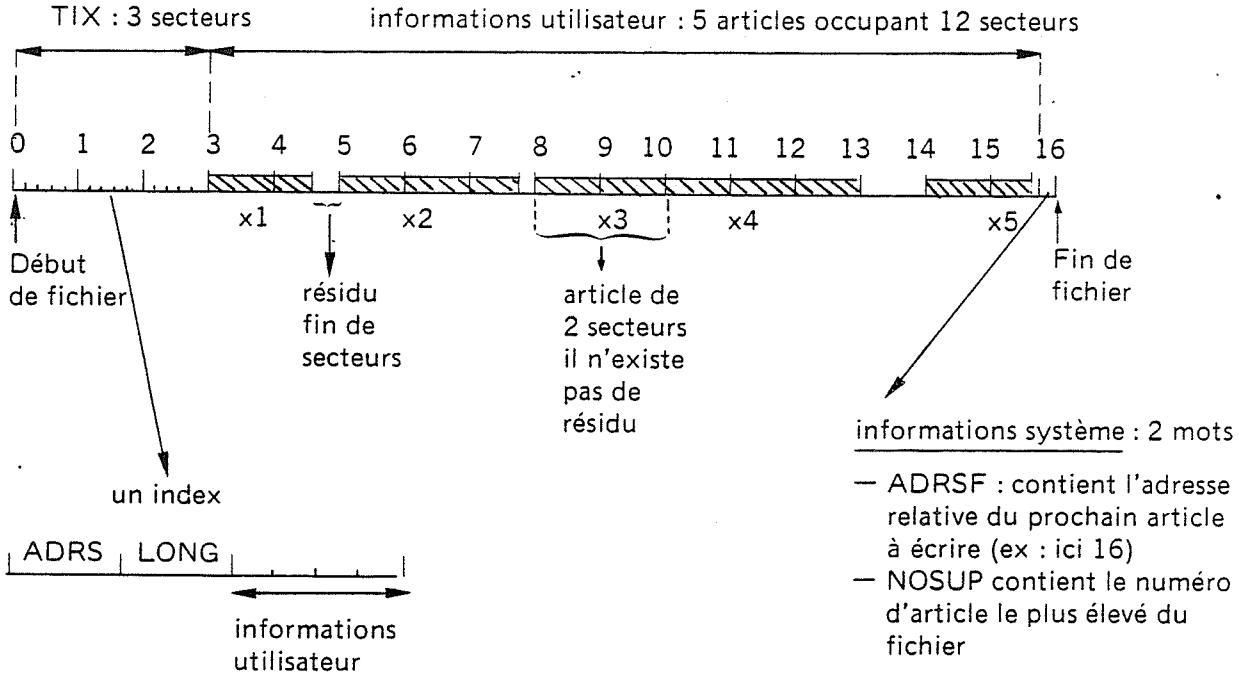
  $1 \leqslant NART < 16\ K$ (16 K = 16.384)
  et $LIX \times NART \leqslant 64$ K octets
- avec la taille standard d'index (4 octets) une table d'index de n secteurs permet de gérer 64 x n articles
- un **index standard à la structure suivante :**

avec

ADRS : adresse relative de l'article dans le fichier. Cette adresse est un nombre de secteurs par rapport au début du fichier.

LONG : longueur de l'article en mots.

10.1.2 - Structure physique d'un fichier Direct Longueur Variable



Soit LSEC la taille du secteur sur disque, en standard 128 mots.

— la table d'index occupe les n premiers secteurs du fichier soit :

. 1 secteur pour LSEC/LIX articles
. 2 secteurs pour 2 (LSEC/LIX) articles
etc...

— chaque article est aligné à une frontière de secteurs d'où l'existence d'un résidu de fin de secteurs

— chaque index est composé de :

. 1 mot pour l'adresse relative ADRS avec la convention
ADRS = 0 si l'article n'existe pas
ADRS < 0 (bit index à 1) si l'article est supprimé

. 1 mot pour la longueur de l'article LONG
$0 \leqslant$ LONG $\leqslant$ 32.767 mots

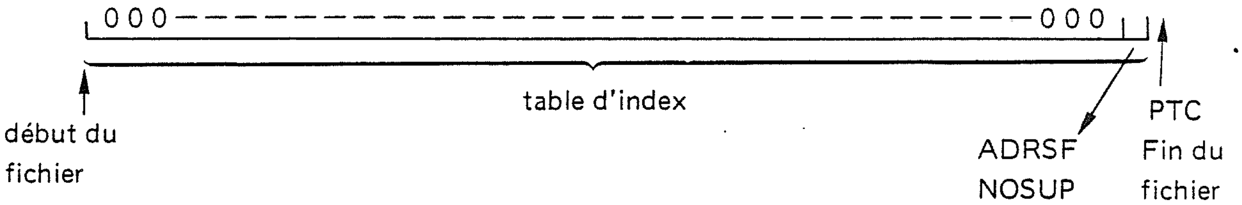
. $\left(\frac{LIX}{2} - 2\right)$ mots pour des informations utilisateurs accessibles dans le buffer de travail (cf 10.2.3 Gestion des mémoires tampons).

NB : La valeur maximale pour NART (16.384) ne peut être atteinte que pour une taille d'index LIX minimale (4 octets).

Fichier Direct V à sa création

A sa création, un fichier Direct V ne contient aucun article. La table d'index est initialisée à zéro.

Fichier Direct V après création :



Avec ADRSF = longueur TIX en secteurs

$$= \frac{\text{LIX x NART'}}{\text{LSEC}}$$

NOSUP = 0

Création d'un article :

A la création de chaque article, la table d'index est mise à jour. Le fichier est agrandi dynamiquement à partir du début de secteur contenant ADRSF, d'un nombre entier de secteurs (immédiatement supérieur ou égal à la longueur de l'article) plus deux mots ADRSF et NOSUP.

Règles :

— au ième élément de la table d'index correspond l'article nº i
— les articles sont implantés dans le fichier d'après l'ordre chronologique de leur création. Ils ne sont pas ordonnés d'après leur numéro.

Recherche d'un poste dans la table d'index :

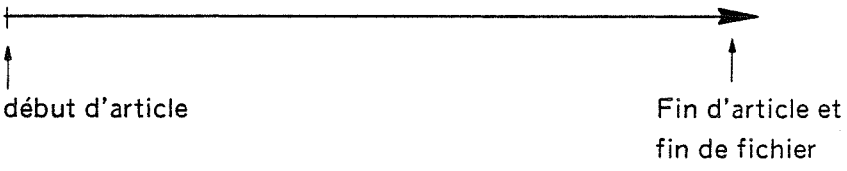
Lors de la création d'un fichier Direct V l'utilisateur fournit un buffer de n secteurs ($n \geqslant 1$) appelé buffer de travail.
Ce buffer est utilisé par l'algorithme de pagination de la table d'index : Lors de la mise à jour de l'index correspondant à l'article i, la page de table d'index contenant i sera chargée en mémoire centrale, si elle n'y était pas déjà. La modification de l'index se fait en mémoire centrale, il est possible de demander une mise à jour immédiate sur le disque. Lors du balayage de la page de table d'index (swap out) cette page sera réécrite sur le disque.

Remarques générales :

D'après le précédent schéma décrivant la structure physique d'un fichier Direct V, on peut remarquer que :

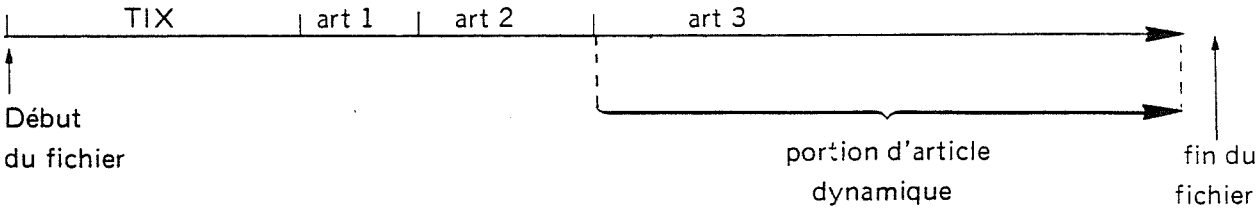
— le fichier est composé d'articles indépendants, c'est-à-dire que chaque article est un vecteur borné de mots.

On peut considérer la table d'index comme étant un article du fichier (pseudo-article ou article système).
Le dernier article, pendant sa création est un vecteur dynamique de mots. Son contenu pourra être crée selon le mode portion d'article dynamique.

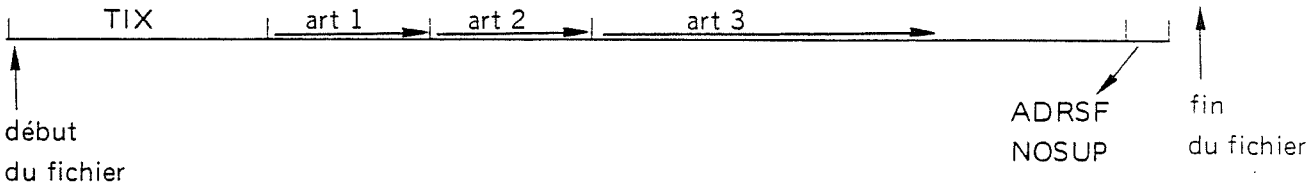


— le fichier lui-même est un vecteur de mots. C'est un vecteur dynamique au sens où on peut l'agrandir par la création d'un nouvel article. C'est un vecteur borné de mots au sens où, en dehors d'une phase de création d'articles, un accès séquentiel au fichier selon le mode séquentiel pur statique est borné par le début et la fin du fichier.

Un fichier Direct V, phase de création d'un article



Un fichier Direct V, figé :



10.2 - LES METHODES D'ACCES

La méthode d'accès Direct V est celle qui gère l'organisation des fichiers Direct V (numéro 5). Elle est entièrement réentrante : si une tâche est en train d'exécuter une requête du Direct V, une autre tâche plus prioritaire faisant appel elle aussi au Direct V pourra exécuter sa requête et se terminer avant la précédente.

10.2.1 - Types d'accès fournis

a) **Accès direct par numéro**

En général, on accède directement à l'article par numéro. Il existe plusieurs requêtes permettant d'accéder à l'article par son numéro ; ce sont :

— VREAD lecture d'un article de numéro donné
— VWRITE écriture d'un article de numéro donné
— VSUP suppression d'un article de numéro donné

Ces requêtes accèdent à l'information utilisateur référencée par le numéro fourni après avoir consulté la table d'index.

Accès direct à l'article de numéro le plus élevé :

Il s'agit en fait d'une utilisation particulière des requêtes précédentes. Le numéro d'article sera remplacé par 'FFFF dans le FCB, dans ce cas, FMS ira lire, écrire ou supprimer l'article de numéro le plus élevé et rendra ce numéro à l'utilisateur.

b) **Accès direct par adresse**

Une seule requête permet l'accès direct aux articles par leur adresse il s'agit de la requête de lecture : VREAD.
L'utilisateur doit alors fournir l'adresse relative par rapport au début du fichier et la longueur en octet de l'article désiré.
Ces deux paramètres sont rendus par FMS lors de l'utilisation des requêtes précédemment citées.

c) Accès à l'article courant

Il s'agit de la requête RENUM, qui effectue un changement de numéro de l'article en cours. Les requêtes VREAD et VWRITE réalisent un positionnement dans le fichier, la requête VRENUM doit être demandée immédiatement après une sélection réalisée par l'une ou l'autre de ces requêtes. La requête VREAD avec accès sur adresse ne sélectionne pas d'article par un VRENUM.

10.2.2 - Type d'accès autorisé

a) Accès séquentiel pur statique

La méthode d'accès autorise l'accès séquentiel pur à tout le fichier. Seule la lecture est alors autorisée. Ceci ne peut se faire qu'à l'aide des primitives de la méthode d'accès séquentiel après l'une des trois primitives (CREAT, OPEN OLD, OPEN NEW) et avant toute requête du Direct Longueur Variable.

Après CREAT et OPEN NEW, le pointeur courant se situe à la fin de la table d'index (mise à zéro). Après OPEN OLD, le pointeur courant adresse le début physique du fichier, c'est-à-dire, le début de la table d'index. Ce type d'accès peut être utilisé pour vérifier le contenu du fichier.

b) Accès séquentiel en portion d'article statique ou dynamique

- le mode portion d'Article statique est autorisé par la requête VREAD pour lire un article en séquentiel
- le mode portion d'Article dynamique est autorisé par la requête VWRITE pour créer un article en séquentiel

10.2.3 - Gestion des mémoires tampons

La méthode d'accès nécessite deux types de buffer pour son fonctionnement :

- un buffer de travail, lié à une unité d'accès du fichier et réservé au système pour consulter la table d'index
- une zone d'échange, liée à une primitive du niveau article et utilisée pour les transferts d'informations entre l'utilisateur et les articles du fichier disque.

a) Le buffer de travail

Ce buffer est donné par l'utilisateur à la création d'une unité d'accès au fichier, c'est-à-dire lors de l'emploi des requêtes CREAT, OPEN NEW et OPEN OLD.
Il devient la propriété de FMS pendant toute la durée de vie de l'unité d'accès, c'est-à-dire jusqu'au CLOSE.

Pendant toute cette période, l'utilisateur ne doit pas modifier les informations système (ADRS, LONG), FMS l'utilise pour ranger tout ou partie de la table d'index. Ce buffer doit être constitué d'un nombre entier de secteurs supérieur ou égal à 1.

Le mécanisme de gestion du buffer est basé sur un principe analogue à celui d'un algorithme de pagination.

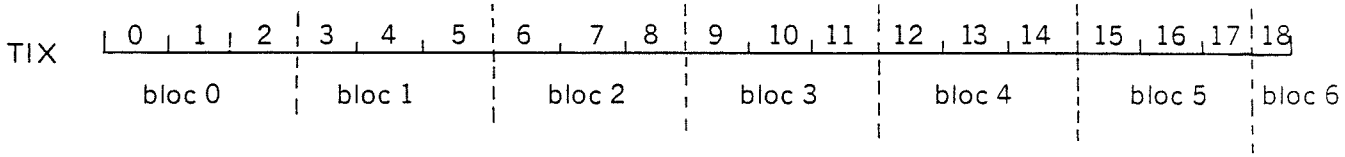
Les temps de réponse globaux sont d'autant meilleurs que le buffer est grand.

Soit une table d'index de p secteurs et un buffer de travail de n secteurs. La table d'index sera alors partagée en k blocs de n secteurs (le dernier bloc peut avoir une longueur inférieure ou égale à n).

Lorsque l'on désire travailler avec l'article i, FMS charge, si celui-ci n'est pas déjà dans le buffer, le bloc de table d'index qui contient cet article.
En cas de modification de ce bloc de table d'index (c'est-à-dire utilisation d'une requête autre que VREAD) la réécriture sur disque peut se faire immédiatement ou se faire de manière retardée (c'est-à-dire lorsque ce bloc n'est plus utile et va être remplacé par un autre).

Exemple :

Soit une table d'index de 18 secteurs et un buffer de travail de 3 secteurs.



Nous considérons donc le bloc 0 formé du secteur 0 au secteur 2, le bloc 1 formé des secteurs 3, 4, 5, etc...
Le dernier bloc n'aura qu'un seul secteur (le secteur 18).

Nous voulons lire l'article numéro 0 ; celui-ci appartient toujours au premier secteur, nous amènerons donc dans le buffer de travail le bloc 0. Si nous désirons par la suite travailler sur l'article j appartenant au secteur 2, FMS ne chargera pas ce bloc 0 en mémoire puisqu'il y est déjà.

Si nous voulons travailler sur l'article p appartenant au secteur 14, le bloc 0 sera réécrit sur le disque s'il a été modifié et le bloc 4 sera mis dans le buffer de travail.

b) **La zone d'échange**

Elle contient les informations que l'utilisateur désire écrire sur disque ou elle recevra les informations que l'utilisateur lit sur le disque.
Elle est caractérisée par sa longueur (LBU en octets) et son adresse (ABU).
Elle doit être fournie par l'utilisateur pour chaque primitive du niveau article.
Elle peut être différente de la taille de l'article : FMS le signale en tant qu'avertissement mais réalise partiellement l'échange.

10.2.4 - Accès à la partie information de l'index

FMS ne fournit pas de requêtes particulières pour accéder à l'information utilisateur d'un index.
Pour cela l'usager dispose de deux méthodes :

— accès à la Table d'Index dans le buffer de travail
— accès à la Table d'Index dans le fichier.

a) Accès à la table d'Index dans le buffer de travail

Le principe en est le suivant ; il faut tout d'abord sélectionner l'article pour lequel on désire lire, créer ou modifier la partie utilisateur de l'index, à l'aide des requêtes VREAD (par numéro), VWRITE, VRENUM (même numéro). Il faut ensuite accéder directement à l'index dans le buffer de travail sans modifier les 2 premiers mots ADRS, LONG qui sont gérés par FMS. L'adresse de l'index de l'article i dans le buffer de travail est égale au reste de la division suivante :

$$\frac{i \times LIX/2}{\text{long buff. Travail (en mots)}}$$

Exemples d'accès à l'index dans le buffer de travail :

— **Lecture** de l'index de l'article i
  - VREAD (i, LBU = 0). FMS charge le bloc contenant l'index i dans le buffer de travail
  - accès au buffer de travail par les instructions machine
— **Création** de l'article i et de son index.
  - VWRITE (i, avec écriture retardée). Après l'exécution de la requête l'index i est chargé dans le buffer de travail
— Chargement de la partie utilisateur de l'index en faisant **attention** à ne pas modifier les 2 premiers mots de l'index. L'index sera mis à jour sur disque par FMS sur le CLOSE ou une prochaine requête du Direct Longueur Variable
— **Modification** de l'index de l'article i
  - VRENUM (i en i, avec écriture retardée). Après l'exécution de la requête l'index i est chargé dans le buffer de travail
  - chargement de la partie utilisateur de l'index comme pour la création d'index expliquée ci-dessus.

b) Accès à la table d'Index dans le fichier

La requête VREAD avec adresse permet d'utiliser les requêtes du séquentiel pour accéder à la table d'index directement sur disque selon le mode portion d'Article statique.

— VREAD avec : ADRS = 0 1er secteur du fichier
  LONG = Longueur de la table d'index
  MAJ = 1 Afin de mettre à jour sur disque les éventuelles modifications précèdemment notées dans le buffer de travail.
— Utilisation des requêtes séquentielles pour lire ou modifier la table d'Index directement sur disque. **Attention** à ne pas modifier les informations système de la table d'index.
  Un REWIND positionne le pointeur courant au début du fichier, et un SKEOA à la fin de la table d'index.

Remarque : le traitement de modification de la table d'index doit être ininterruptible afin de garantir la validité des informations de la table d'index et du fichier.

10.3 - LES FONCTIONS LOGIQUES

10.3.1 - Le niveau fichier

a) **Généralités**

Les requêtes du niveau fichier dans le cas d'un Direct V sont semblables à celles des autres méthodes d'accès. Leur fonctionnement est décrit dans le chapitre 4 du manuel de référence de FMS : l'entité fichier.

Les différentes requêtes CREAT, OPEN NEW, OPEN OLD et CLOSE sont décrites dans les pages suivantes. L'interface d'appel d'une requête au niveau fichier est écrit en PL1600.

RA = @ FCB
SVC (FMS) où FMS = '38

b) **CREAT, OPEN NEW**

Création d'un fichier Direct Longueur Variable.

— **paramètres spécifiques**
  . organisation logique n° 5
  . nombre maximum d'articles que le fichier peut contenir NART tel que $1 \leqslant NART \leqslant 16384$
  . longueur d'un élément de la table d'index LIX tel que $4 \leqslant LIX \leqslant 256$ octets ; $LIX = 2^n$ et $LIX \times NART \leqslant 64$ K octets.

— la taille des articles est spécifiée lors de la création des articles. FMS alloue, à la création du fichier le nombre de secteurs nécessaire à la table d'index, plus deux mots (pour ADRSF et NOSUP). La table d'index est initialisée à zéro, ADRSF contient l'adresse du premier secteur d'information utilisateur, NOSUP contient 0. Après la création de la table d'index le pointeur courant coïncide avec la fin du fichier (FFFS : 3e mot du 1er secteur utilisateur). Un fichier Direct V est crée avec une organisation physique séquentielle (Fichier dynamique).

c) **OPEN OLD**

Cette primitive est utilisée pour ouvrir un fichier permanent, précédemment crée. Elle permet de fournir à FMS les caractéristiques du buffer de travail (ABU, LBU). Le buffer de travail est formé d'un nombre entier de secteurs.

d) CLOSE

Cette primitive détruit une unité d'accès (FAU) à un fichier, elle détruit un fichier temporaire ou ferme un fichier permanent.

Après une phase de modification du fichier, création et suppression d'articles, il est possible de créer une organisation physique directe sur le fichier afin d'accélérer son exploitation en consultation et réécriture, on obtient donc un fichier statique.
Cette modification est effectuée par la requête ALTER Total (paragraphe 4.26) qui devra être programmée juste avant le CLOSE du fichier.
Toute modification de la taille du fichier (création d'articles, suppression avec tassage) lors d'une phase d'utilisation du fichier (après un OPEN) retransforme automatiquement le fichier en dynamique.

L'interface de la requête CLOSE n'est donc pas modifié pour le Direct Longueur Variable.

Nom	**CREAT**
But	Créer un fichier permanent Direct V et créer une unité d'accès à ce fichier

Appel RA ⟵ FCB SVC (FMS) où FMS = `38

FCB

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BUF	ADR	NLC	0	4				FNUM							
1	ABU															
2	LBU															
3	PR ←															
4	FNAM															
5																
6																
7	PUBW															
8	0	1	S	W			0	1	SU/FU							
9	0															
10	LIX															
11	NART															
12	0															

Description des paramètres

BUF = 1 → il existe un buffer de travail

(ABU, LBU) caractéristiques du buffer de travail

LBU = k. 256 octets

LIX : longueur de l'élément de table d'index en octets telle que

$4 \leqslant LIX \leqslant 256$ octets

LIX = 4, 8, 16, 32, 64, 128, 256

NART : nombre maximum d'articles tel que

$1 \leqslant NART \leqslant 16.383$

et $\frac{LIX}{2} \times NART \leqslant 32\,766$ mots

les articles seront numérotés de 0 à NART − 1

Comptes Rendus	Valeur de PR	Signification
	0	**Primitive correctement exécutée**
	`600B	FAU existante
	`600D	Fichier existant
	`601F	Primitive en cours
	`6020	Zone de pavés saturée
	`6021	FU saturée : il n'y a pas assez de place pour la table d'index, requête ineffective.
	`6022	Table de fichiers saturée
	`6028	Erreur de syntaxe : @FCB, FONCT, ABU, LBU, FNAM, FTYP, LIX, NART
	`6029	@ FCB invalide
	`602A	SU ou FU non gérées par FMS
	`602B	Méthode d'accès non gérées
Erreurs graves	`6032	
	`6033	Informations système invalides
	`6034	
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : 4000 + mot d'état PU

Nom	**OPEN NEW**
But	Créer un fichier Direct V temporaire et une unité d'accès à ce fichier.

Appel RA = @FCB SVC FMS où FMS = `38

FCB

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BUF	ADR	NLC	0	1				FNUM							
1	ABU															
2	LBU															
3	PR ←															
4	FNAM															
5																
6																
7	PUBW															
8	0	1	S	W			0	1	SU/FU							
9	0															
10	LIX															
11	NART															
12	0															

Description des paramètres

BUF = 1 → il existe un buffer de travail

ABU, LBU caractéristiques du buffer de travail
LBU = k secteurs (en octets)

LIX : longueur d'un élément de table d'index telle que
$4 \leqslant LIX \leqslant 256$

LIX = 4, 8, 16, 32, 64, 128, 256

NART : nombre maximum d'articles du fichier tel que
$1 \leqslant NART \leqslant 16.383$
et $\frac{LIX}{2} \times NART \leqslant 32.766$ mots
les articles seront numérotés de 0 à NART − 1

Comptes Rendus	Valeur de PR	Signification
	0	**Primitive correctement exécutée**
	`600B	FAU existante
	`600D	Fichier existant
	`601F	Primitive en cours
	`6020	Zone de pavés saturées
	`6021	FU saturée : il n'y a pas assez de place pour la table d'index. Requête ineffective
	`6022	Table de fichiers saturée
	`6028	Erreur de syntaxe : FCB, FONCT, ABU, LBU, FNAM, FTYP, LIX, NART
	`6029	FCB invalide
	`602A	SU ou FU non gérées par FMS
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032 `6033 `6034	Informations systèmes invalides
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU

Nom	**OPEN OLD**
But	Ouvrir un fichier permanent et créer une unité d'accès à ce fichier

Appel RA = @FCB SVC (FMS) où FMS = `38

FCB

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BUF	ADR	NLC	0	2				FNUM							
1	ABU															
2	LBU															
3	PR ←															
4	FNAM															
5																
6																
7	PUBW															
8	0	1	S	W	RWK		0	1	SU/FU							
9	0															
10	0															
11	0															
12	0															

Description des paramètres

BUF = 1 → il existe un buffer de travail

(ABU, LBU) caractéristiques de ce buffer de travail

LBU = k 256 (buffers de k secteurs) en octets

Comptes Rendus	Valeur de PR	Signification
	0	**Primitive correctement exécutée**
	`600B	FAU existante
	`600C	Fichier existant
	`6014	Protection écriture
	`6015	Permanent de nature différente
	`601E	Fichier occupé
	`601F	Primitive en cours
	`6020	Zones de pavés saturées
	`6028	Erreur de syntaxe : @ FCB, FONCT, ABU, LBU, FNAM, PUBW, RWK
	`6029	Adresse de FCB invalide
	`602A	SU ou FU non gérée par FMS
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032	Informations système invalides
	`6034	
	`6035	FU verrouillée
	`4 . . .	Erreur hardware : `4000 + mot d'état PU

10.3.2 - Le niveau article

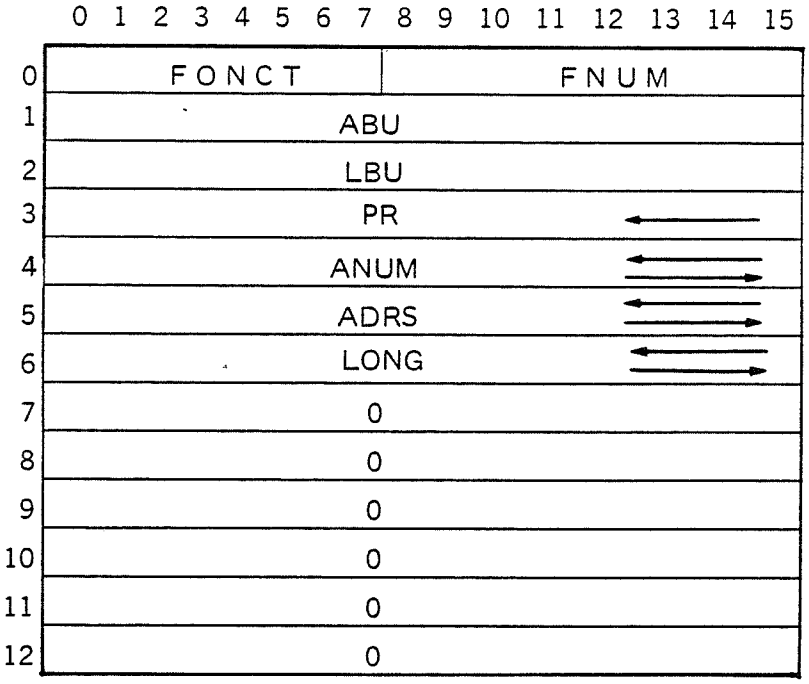
a) Généralités

Le Direct V fournit les primitives suivantes pour accèder directement aux articles : VREAD, VWRITE, VSUP, VRENUM.
L'interface d'appel d'une requête du niveau article pour la méthode d'accès est en PL1600.

RA = @FCB SVC (FMSV) où FMSV = `2A

Description générale des FCB

FCB



Conventions :

— les mots mis à zéro dans le FCB ne sont pas utilisés par FMS

— ⟵ paramètres de retour fournis par FMS

— ⟷ le paramètre peut être précisé ou non par l'utilisateur. S'il n'est pas précisé, il sera rendu par FMS

— dans tous les autres cas, les paramètres sont à fournir.

Toute requête du niveau article s'adresse à l'unité d'accès spécifiée par l'utilisateur (FNUM) et concerne le fichier accessible par celle-ci. Le choix d'une primitive parmi celles offertes par la méthode d'accès est fait par l'usager grâce à l'octet FONCT.

b) VREAD : lecture d'un article

Cette primitive permet de lire soit :

— un article désigné par son numéro (ANUM)
— un article désigné par son adresse (ADRS)
— l'article de numéro le plus élevé (ANUM = `FFFF).

Cet article est transféré dans la zone d'échange spécifiée par l'utilisateur.

Cette primitive permet de sélectionner un article pour lui changer son numéro (utilisation de la requête VRENUM) ou pour y accèder de façon séquentielle selon le mode portion d'article statique.

Attention : la lecture d'un article désigné par son adresse permet la portion d'article statique mais interdit le VRENUM.

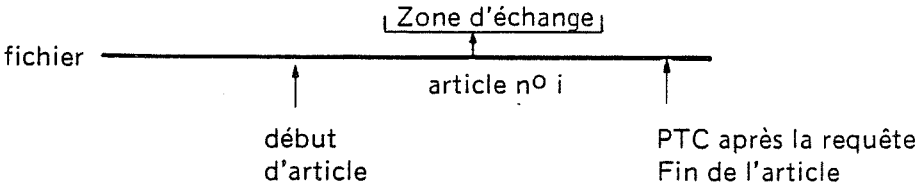
Si l'article est désigné par son adresse, l'information correspondante sera chargée directement dans la zone utilisateur.

Dans les autres cas, FMS charge en mémoire la page (n secteurs, n a été défini lors de la création ou de l'ouverture) de la table d'index correspondant au numéro désiré, et obtient ainsi l'adresse et la longueur de l'article concerné.

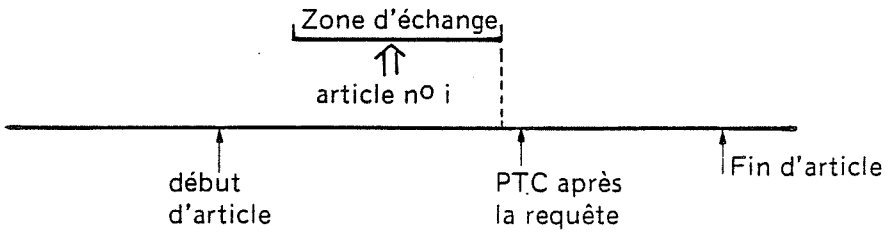
L'article est cadré à gauche dans la zone d'échange.

On peut avoir les cas suivants :

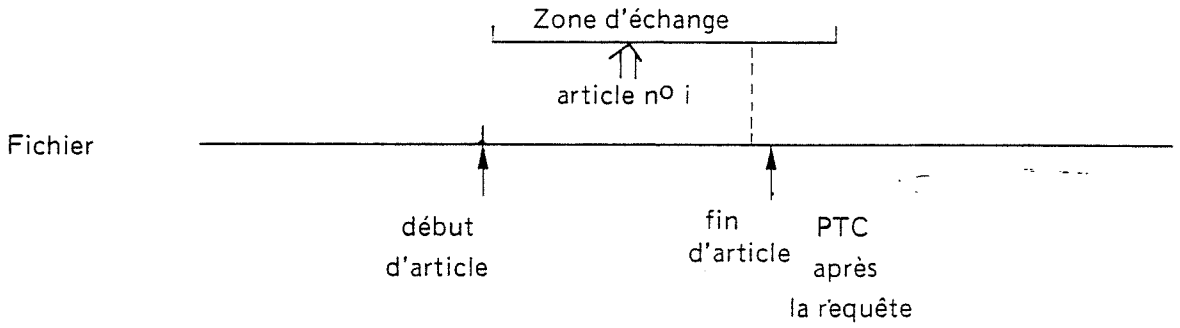
* PR = 0 primitive correctement exécutée



* PR = `6003 Article plus long que la zone d'échange



* PR = `6004 Article plus court que la zone d'échange



Article désigné par son numéro i :

— paramètres fournis par l'utilisateur :

FONCT = 01
FNUM = x
ABU
LBU
ANUM = i ADRS = 'FFFF LONG = 0

— paramètres rendus par FMS après l'exécution de la requête :

PR
ADRS
LONG

Article de nº le plus élevé :

— paramètres fournis par l'utilisateur :

FONCT, FNUM
ABU, LBU
ANUM = 'FFFF
ADRS = 'FFFF
LONG = 0

— paramètres rendus par FMS après l'exécution de la requête :

PR
ANUM = i
ADRS
LONG

Article désigné par son adresse relative :

— paramètres fournis par l'utilisateur :
FONCT, FNUM
ABU, LBU
ADRS, LONG
ANUM quelconque

— paramètres rendus par FMS après l'exécution de la requête : PR

Nom VREAD lecture d'un article

Appel RA = @ FCB SVC (FMSV) où FMSV = `2A

FCB

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	MAJ		01						FNUM							
1	ABU															
2	LBU															
3	PR															←
4	ANUM															↔
5	'FFFF/ADRS															↔
6	0/LONG															↔

Description des paramètres

— (ABU, LBU) définition de la zone d'échange
En mode maître FMS contrôle par rapport à la taille de la mémoire. En mode esclave, ils seront contrôlés par rapport à SLO et SLE.
LBU est exprimé en octets, c'est un nombre pair
$0 \leqslant LBU \leqslant 65.534$ octets

— ANUM nº d'article à lire $0 \leqslant ANUM < NART$
Si ANUM ='FFFF , lecture de l'article de nº le plus élevé

— ADRS adresse relative de l'article à lire (adresse secteur)
$0 \leqslant ADRS \leqslant 32.767$ secteurs
Lorsque ADRS est fourni par l'utilisateur, ANUM est quelconque
ADRS = 'FFFF, lecture par numéro

— LONG longueur de l'article en octets
Cette quantité doit être fournie lorsque l'on précise l'adresse. Sinon, elle est indifférente
$0 \leqslant LONG \leqslant 65.534$ octets
LONG : nb pair d'octets

— MAJ : si MAJ = 1 FMS met à jour (si nécessaire) la table d'Index sur disque avant d'exécuter la requête.

Compte Rendus	Valeur de PR	Signification
	0	**Primitive exécutée correctement**
	`6003	Article du fichier plus long
	`6004	Article du fichier plus court
	`600A	FAU inexistante
	`600E	Article inexistant
	`6018	Incomptabilité primitive fichier
	`601F	Primitive en cours
	`6028	Erreur de syntaxe : @FCB, ABU, LBU, ANUM
	`6029	Adresse de FCB invalide
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032	Informations système invalides
	`6034	
	`6035	FU verrouillée par IOCS
	`4...	Erreur hardware : `4000 + mot d'état PU

Attention :

— si on effectue une lecture par ADRS et LONG, les comptes-rendus 6003 et 6004 signalent seulement le fait que la zone d'échange (LBU) et la longueur que l'on veut lire (LONG) ne sont pas identiques.

— lorsque le compte-rendu 6004 (article plus court) est rendu après un VREAD par numéro il se peut que cela provienne d'un mauvais enchainement des requêtes :
- VRENUM sans suppression d'index
- VSUP avec suppression d'information.

Dans ce cas l'information d'un article n'existe plus dans le fichier bien que des index s'y réfèrent. Pour remettre en ordre le fichier, utiliser la requête VSUP (d'index seulement).

c) VWRITE : création d'un article

Cette primitive permet de créer un article désigné par son numéro ANUM, ou un article dont le numéro est immédiatement supérieur au numéro le plus élevé. Le contenu de ce nouvel article est dans la zone d'échange (ABU, LBU).

FMS vérifie que le numéro désiré par l'utilisateur n'a pas déjà été attribué.

Cette primitive s'utilise pour créer un article selon deux modes différents :

— création au niveau article seulement
— création au niveau article et portion d'article.

Allocation dynamique à la fin du fichier

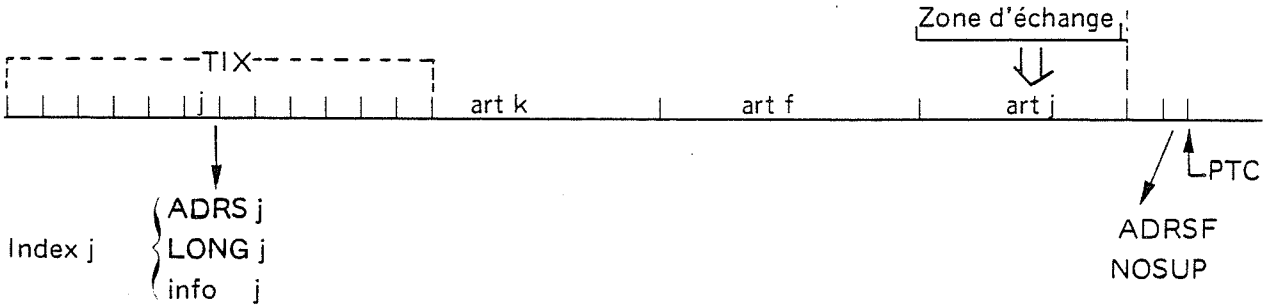
Tout nouvel article est écrit en fin de fichier. FMS alloue le nombre de secteurs égal ou immédiatement supérieur à la taille de l'article plus deux mots (ADRSF et NOSUP).

Création au niveau article seulement

La zone d'échange associée à la requête contient toute l'information de l'article. La longueur de l'article (LBU) est spécifiée par un nombre d'octets pair tel que
$0 \leqslant LBU \leqslant$ 'FFFƆ

Si le nombre d'octets est impair, il sera automatiquement appairé par FMS.
Il est déconseillé, mais non interdit, de créer des articles de longueur nulle.
L'index correspondant au numéro de l'article sera mis à jour lors de la prochaine requête du Direct Longueur Variable.

Exemple : VWRITE article nº j



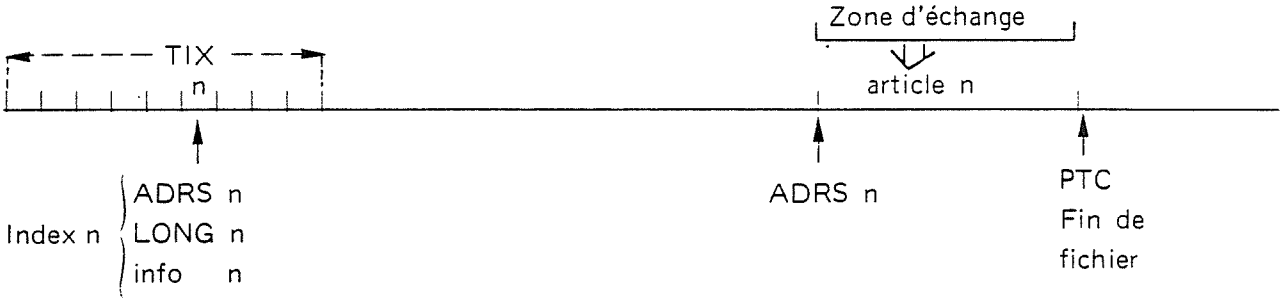
Remarque :

L'article de numéro zéro doit être créé explicitement avec ANUM = 0 et non par numéro supérieur ANUM = `FFFF.

Création au niveau article et portion d'article :

L'article est crée de la même façon que précèdemment. Dans ce cas, il est souhaitable que l'article est une longueur d'un nombre pair d'octets.
La portion d'article permettra d'agrandir l'article par la fin.

Exemple : VWRITE article n



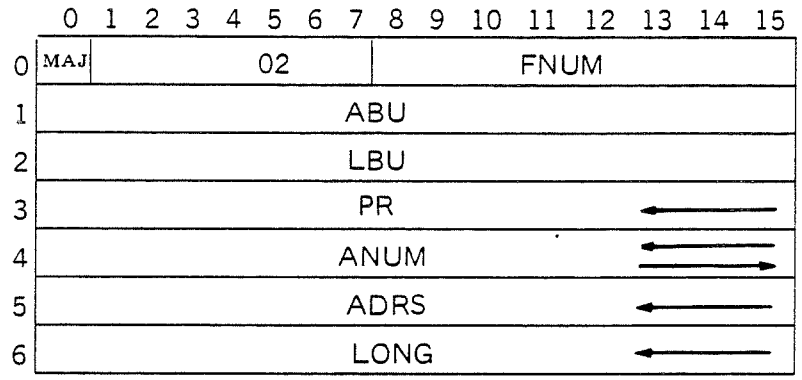
On peut se servir de la portion d'article dynamique (WRITE, READ. . . . . .) jusqu'à la rencontre d'une requête du Direct V. Dès lors la taille de l'article n sera figée, ADRSF sera mis à jour ainsi que la longueur de l'article dans la table d'index.

Si on utilise la portion d'article dynamique, il n'est pas possible de demander la mise à jour immédiate de ADRSF et NOSUP.

Nom VWRITE

Appel RA = @FCB SVC (FMSV) où FMSV = `2A

FCB



Description des paramètres

— (ABU, LBU) caractéristiques de la zone d'échange
0 ⩽ LBU ⩽ 65.534 octets

— ANUM numéro de l'article à créer
0 ⩽ ANUM ⩽ NART

Si ANUM = 'FFFF, l'article créé aura le numéro immédiatement supérieur au plus élevé. Dans ce cas ANUM sera rendu par FMS-E

— ADRS adresse relative de l'article que l'on vient de créer.
Cette adresse est rendue par FMS

— LONG longueur de l'article en octets
Rendu par FMS.

— MAJ booléen de mise à jour immédiate de la table d'index et de (ADRSF, NOSUP)
MAJ = 1 si mise à jour immédiate, 0 sinon

Comptes Rendus	Valeur de PR	Signification
	0	**Primitive correctement exécutée**
	`600A	FAU inexistante
	`600F	Article existant
	`6014	Protection écriture
	`6016	Fichier saturé : La table d'index est saturée : cette requête est ineffective
	`6017	Fichier trop long : supérieur à 32 K secteurs : la requête est ineffective
	`6018	Incompatibilité primitive fichier
	`601F	Primitive en cours
	`6021	FU saturée
	`6028	Erreur de syntaxe : FCB, FONCT, ABU, LBU, ANUM
	`6029	Adresse de FCB invalide
	`602B	Méthode non gérée
Erreurs graves	`6032	Informations système invalides
	`6034	
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU

d) VSUP : Suppression d'un article

Cette primitive permet de supprimer soit un article désigné par son numéro (ANUM) soit l'article de numéro le plus élevé (ANUM = 'FFFF).

La suppression est logique : seul l'index correspondant au numéro de l'article est modifié ; le bit index du mot ADRS est forcé à 1.

En général, la place occupée par l'information de l'article n'est pas récupérée par la primitive VSUP. On récuperera la place correspondant à des suppressions d'articles par tassage du fichier, à l'aide d'un traitement différé.

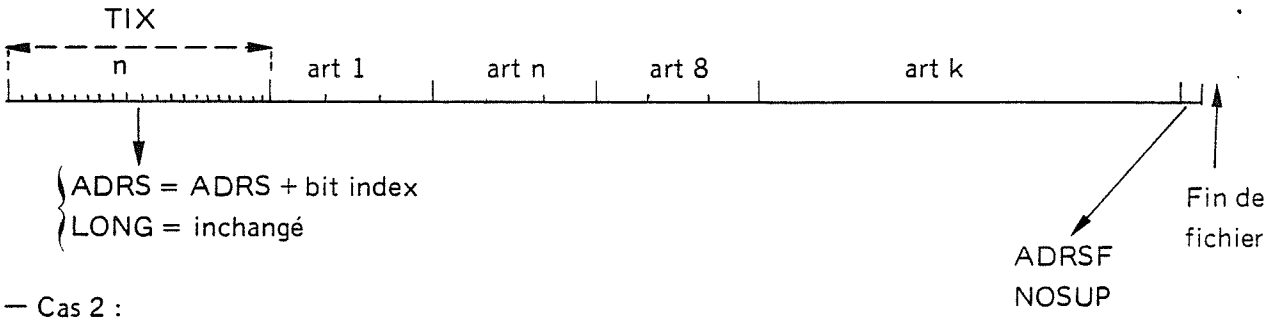
Toutefois, si l'article supprimé est le dernier créé par ordre chronologique, c'est-à-dire que son information se trouve à la fin du fichier, le fichier sera réduit de la longueur correspondant à l'article. Le pointeur ADRSF sera mis à jour ainsi que l'index correspondant à l'article (il y a dans ce cas et dans ce cas seulement récupération de la place).
Si l'on veut éviter cette suppression physique, il est possible de le préciser dans le FCB

Exemple : Suppression de l'article n

— Cas 1 :

L'information n'est pas en queue de fichier

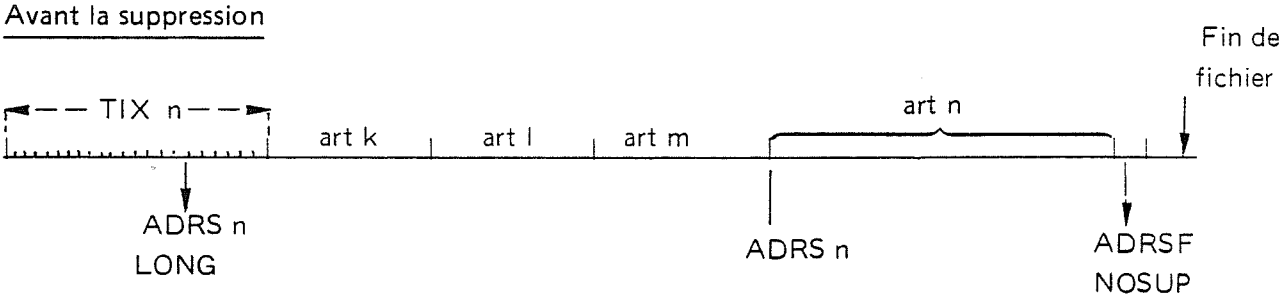
Après la suppression :



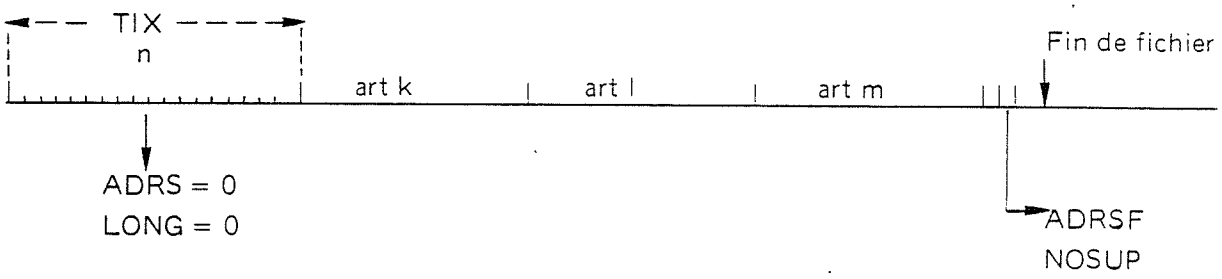
— Cas 2 :

L'information est en fin de fichier

Avant la suppression



Après la suppression



La primitive VSUP interdit tout accès séquentiel sur l'unité d'accès concernée.

NB :

Lors de la suppression physique d'un article, l'index correspondant est remis à zéro. Dans les autres cas, on force le bit index du mot ADRS. Le fait de forcer à 1 le bit index de ADRS signifie qu'il y a de la place perdue dans le fichier.
Cette remarque peut être utilisée pour déterminer le taux de remplissage réel d'un fichier.

Nom VSUP : suppression d'un article

Appel RA = @FCB SVC (FMSV) où FMSV = `2A

FCB

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	MAJ			03					FNUM							
1	ABU = 1/0															
2	0															
3	PR															←
4	ANUM															←→
5	ADRS															←
6	LONG															←

Description des paramètres

— ANUM : n° de l'article à supprimer
Si ANUM = `FFFF l'article de numéro le plus élevé sera supprimé
$0 \leqslant ANUM \leqslant NART$

— ADRS : adresse relative de l'article qui vient d'être supprimé

— LONG : longueur de l'article que l'on vient de supprimer
Ces deux paramètres valent 0 s'il y a eu suppression physiques (récupération de la place).

— MAJ : booléen de mise à jour immédiate de la table d'index et de ADRSF et NOSUP
MAJ = 1 il y-a maj immédiate
= 0 sinon

— ABU = 0 **suppression physique :** suppression de l'index et du contenu de l'article quand ce dernier est en fin de fichier, avec donc un raccourcissement du fichier
= 1 **suppression logique :** suppression de l'index seulement.

Comptes Rendus	Valeur de PR	Signification
	0	**Primitive correctement exécutée**
	`600A	FAU inexistante
	`600E	Article inexistant
	`6014	Protection écriture
	`6018	Incompatibilité primitive fichier
	`601F	Primitive en cours
	`6028	Erreur de syntaxe : FCB, FONCT, ANUM
	`6029	Adresse de FCB invalide
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032	Informations système invalides
	`6034	
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU

e) VRENUM : Rénuméroter un article

Cette primitive permet de changer le numéro de l'article en cours par celui fourni dans le FCB. L'article dont on veut changer le numéro aura préalablement été sélectionné par une requête VWRITE ou VREAD (attention : la requête VREAD appelé par ADRS et LONG ne modifie pas la sélection pour le VRENUM mais sélectionne la protion d'article statique définie par ADRS et LONG.

FMS contrôle que le numéro n'a pas déjà été attribué. La table d'index est mise à jour : les informations contenues dans l'index de l'article en cours sont transférées dans l'index correspondant au nouveau numéro.

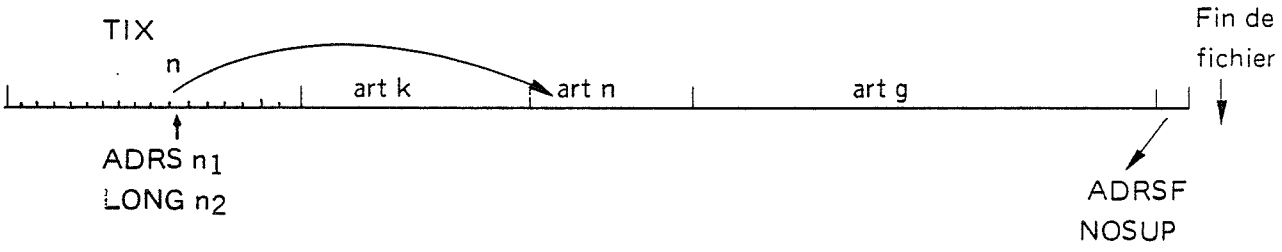
Il y a possibilité de conserver l'ancien index (deux index pointeront alors sur la même partie information)
Toutefois, si l'utilisateur utilise cette possibilité, la suppression de tous les articles de même information sera à sa charge lors de la suppression de la partie information correspondante.

Dans le cas général, on supprime l'ancien index, FMS le met alors à zéro.

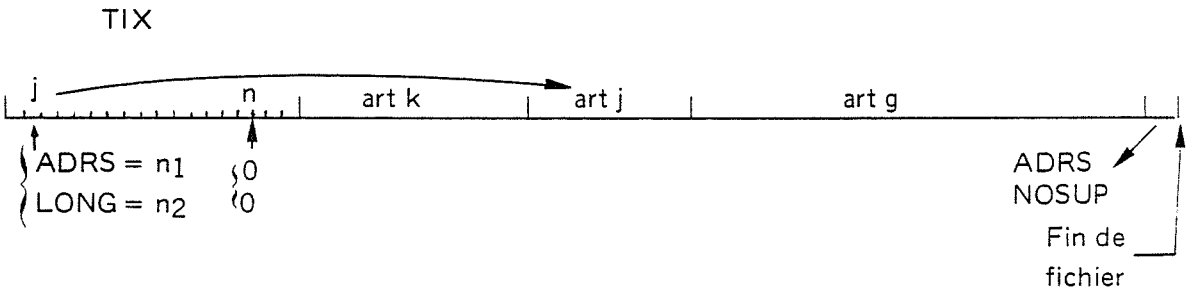
Exemples :

— Cas 1 : Rénumérotation avec suppression de l'ancien index.

Avant :

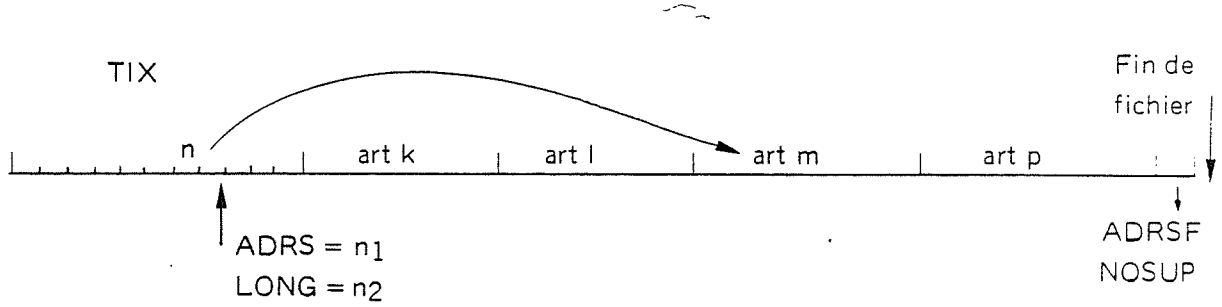


Après :

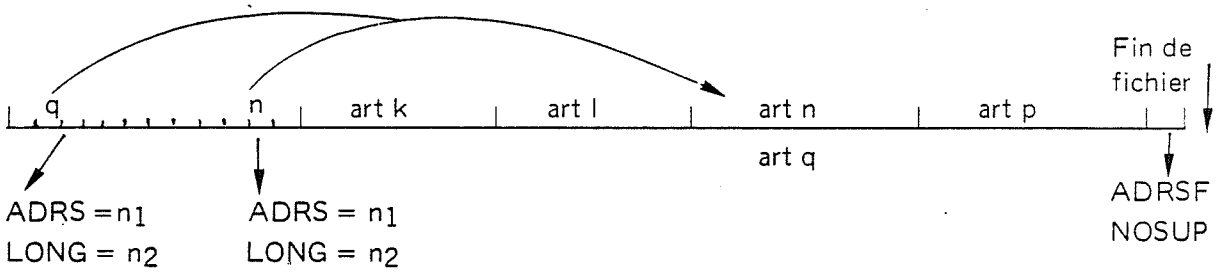


— Cas 2 : Rénumérotation sans suppression de l'index

Avant :



Après :



Attention :

— un VRENUM identité (i en i) avec ABU = 0 à comme résultat final la suppression de l'index i sans toucher à l'information de l'article. Ce fonctionnement est équivalent à celui du VSUP d'index

— un VRENUM (i en j) ne transfère pas la partie utilisateur de l'index i dans l'index j.

Nom VRENUM : rénumérotation d'un article

Appel RA = @FCB SVC (FMSV) où FMSV = `2A

FCB

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	MAJ				04				FNUM							
1	ABU = 0 ou 1															
2	0															
3	PR															←
4	ANUM															↔
5	ADRS															←
6	LONG															←

Description des paramètres

— ANUM : n° à donner à l'article en cours
$1 \leqslant ANUM \leqslant NART$
FMS vérifie que ANUM ne correspond pas à un article existant
Si ANUM = 'FFFF, le n° attribué sera le numéro le plus élevé − 1

— ABU = 0 supprimer l'ancien numéro
ABU = 1 garder l'ancien numéro (2 articles de n° différents auront la même information)

— ADRS adresse de la partie information

— LONG longueur de la partie information

— MAJ booléen de mise à jour immédiate de la table d'index et de (ADRSF, NOSUP)
MAJ = 1 si maj immédiate, 0 sinon.

Comptes Rendus	Valeur de PR	Signification
	0	**Primitive correctement exécutée**
	`600A	FAU inexistante
	`600F	Article existant
	`6014	Protection écriture
	`6016	Fichier saturé : la table d'index est saturée ; la requête est ineffective
	`6018	Incompatibilité fichier primitive
	`601A	Erreur d'enchainement : article non sélectionné par VREAD ou VWRITE
	`601F	Primitive en cours
	`6028	Erreur de syntaxe : FCB, FONCT, ANUM, ABU
	`6029	Adresse de FCB invalide
	`602B	Méthode d'accès non gérée
Erreurs graves	`6032	Informations système invalides
	`6034	
	`6035	FU verrouillée par IOCS
	`4 . . .	Erreur hardware : `4000 + mot d'état PU

ANNEXES

A — LEXIQUE

B — SYNOPTIQUE

C — INDEX

A — LEXIQUE

Apairer — Apairer un nombre signifie rendre ce nombre pair. Apairer un compte d'octets permet, par exemple, de cadrer une E/S à une frontière de mot.

Article — L'article ou enregistrement logique est l'élément générique d'un fichier. Un fichier est un ensemble d'articles. Dans FMS, la taille d'un article peut être définie indépendamment des caractéristiques physiques du support choisi pour le fichier et en particulier de la taille de l'unité d'enregistrement physique (UEP).

Catalogue — Sur une FU, l'ensemble des fichiers permanents qui ont le même mot de passe public (PUBW).

Enregistrement Physique — Un enregistrement physique est la quantité d'informations qui peut être échangée entre la mémoire centrale et un périphérique en une seule demande d'E/S adressée au périphérique.

Maximum sur disque avec FMS : la taille du granule. Un enregistrement physique est constitué d'une ou plusieurs unités d'enregistrement physique (voir UEP).

F A U — Unité d'Accès à un Fichier. Une FAU est :

- Un chemin d'accès au fichier considéré comme une ressource logique
- Une Table Système de pointeurs gérés par FMS permettant l'accès au fichier.

Dans FMS, un usager peut créer et détruire des FAU, elles sont identifiées par un numéro concaténé avec celui de l'usager.

numéro de FAU pour FMS = USR ⊕ FNUM
numéro de FAU pour l'usager = FNUM.

F C B — File Control Block. Un FCB est une Table contenant les paramètres d'une primitive envoyée à FMS.

Fichier — Un fichier désigne soit un contenu d'information, soit tout ce qui le caractérise : identification statique du fichier (dans FMS : FNAM, PUBW, SU ou FU),type du fichier (dans FMS : FTYP), description de la structure du fichier en articles, etc. et que l'on peut appeler le contenant.

FU — Unité Fonctionnelle. La notion de FU utilisée dans FMS est identique à celle utilisée dans IOCS. Une FU est un espace d'adressage d'une Unité Physique, elle permet de choisir un support pour un fichier.

Granule — Quantum d'allocation de mémoire secondaire adressable. FMS alloue un ou plusieurs granules à un fichier. Un fichier est donc constitué d'une chaine de granules. La Taille d'un granule est paramétrable sur chaque support. Taille standard : 2 K mots de 16 bits.

Interface — L'interface entre deux modules est défini par l'ensemble des primitives, que ces deux modules s'échangent mutuellement.

IOCS — Input Output Control System. Partie d'un superviseur gérant les Entrées/Sorties au niveau physique. Voir Manuel de Référence IOCS.

Ligatures de granule — Technique de doubles pointeurs permettant de chainer entre eux les granules d'un même fichier.

Module — Programme assurant une fonction de manière aussi autonome et exclusive que possible. Un module est fortement caractérisé par les primitives permettant de l'appeler.

Mono-accès — Un fichier est dit en mono-accès si une seule FAU est autorisée sur ce fichier à un instant donné.

Partageable — Un fichier est dit partageable entre des usagers si ceux-ci peuvent théoriquement y accéder. Dans FMS tous les fichiers permanents sont partageables.

Primitive — Désigne un appel standard de module caractérisé par un nom, des paramètres d'entrée, des paramètres de retour. Une primitive d'appel à FMS est une requête programmée (SVC).

Radix 40

Technique de codage (FNAM, PUBW, ANAM) en base 40 de 3 caractères ASCII sur un mot de 16 bits:
Règle : Radix 40 (ABC) = A + B x 40 + C x 40^2
Avec Nul = 0, A = 1, ..,Z = 26, 0 = 27, ...,9 = 36, : = 37, ; = 38, <= 39.
Exemple Radix 40 (:< B) = '12BD

Simultané

Un fichier Permanent est dit partageable simultané ou en abrégé simultané si à un instant donné, plusieurs usagers peuvent chacun posséder une FAU accédant à ce fichier.

SU

Unité Symbolique. La notion de SU utilisée dans FMS est un sous-ensemble de celle utilisée dans IOCS. Dans FMS, la SU permet de choisir un support pour un fichier indépendamment du périphérique utilisable au moment de l'exécution. Juste avant l'exécution, l'affectation SU⟶FU permet de choisir le périphérique à utiliser.

Superviseur

Le superviseur est la partie du système (ou système d'exploitation) qui :

— permet l'exploitation des tâches ou programmes utilisateur
— supervise le fonctionnement des différents composants du système.
moniteur de fichiers : FMS
moniteur d'E/S : IOCS

Exemple : le superviseur de BOS16

SVC

Supervisory Call : instruction permettant d'appeler le superviseur pour lui demander l'exécution d'une fonction particulière.
ex : SVC d'E/S à IOCS
ex : SVC d'E/S sur fichier à FMS

Système

Système d'Exploitation, Operating System, ensemble de programmes permettant l'exploitation d'un calculateur.

— superviseur et moniteurs
— processeurs système : FORTRAN, PL1600, Editeur de liens, Bibliothèques système.

Les Systèmes d'Exploitation se distinguent par les différents services qu'ils fournissent à l'utilisateur.

— traitement par lot en mono-programmation : BOS16
— multi-tâche Temps Réel : RTES16
— Temps Partagé : MUTEX.
— Système Multi-fonctions : MPES16.

T I X	Table d'Index d'un fichier Indexé ou Séquentiel Indexé.
U E P	Unité d'enregistrement Physique. Sur disque quantum adressable : le secteur (128 mots)
Usager	Désigne soit une personne, soit plus souvent un programme en cours d'exécution.

B — SYNOPTIQUE

(voir détails page 3-7 (MR))

Les Compte-Rendus de FMS	0	Primitive correctement exécutée
	0 ⩽ PR ⩽ `3FFE	Primitive correctement exécutée : READ, WRITE, SKIPB, SKIPF .
Avertissements	`6001	Fin d'Article
	`6002	Début d'Article
	`6003	Article du fichier plus long
	`6004	Article du fichier plus court
	`6005	Article incorrect
	`6006	Fin de chaîne
	`6007	Début de chaîne
Erreurs	`600A	FAU inexistante
	`600B	FAU existante
	`600C	Fichier inexistant
	`600D	Fichier existant
	`600E	Article inexistant
	`600F	Article existant
	`6014	Protection écriture
	`6015	Permanent de nature différente
	`6016	Fichier saturé
	`6017	Fichier trop long
	`6018	Incompatibilité Primitive Fichier
	`601A	Erreur d'enchaînement
	`601E	Fichier occupé
	`601F	Primitive en cours
Erreurs filtrées par les superviseurs RBOS/D RTES/D	`6020	Zone de pavés saturée
	`6021	FU saturée
	`6022	Table des fichiers saturée
	`6028	Erreur de Syntaxe
	`6029	Adresse de FCB invalide
	`602A	SU ou FU non gérée par FMS ou IOCS
	`602B	Méthode d'Accès non gérée
Erreurs graves	`6032, `6033, `6034	Informations Système Invalides
	`6035	FU Verrouillée par IOCS
	`4000	Erreur hardware : `4000 mot d'état PU

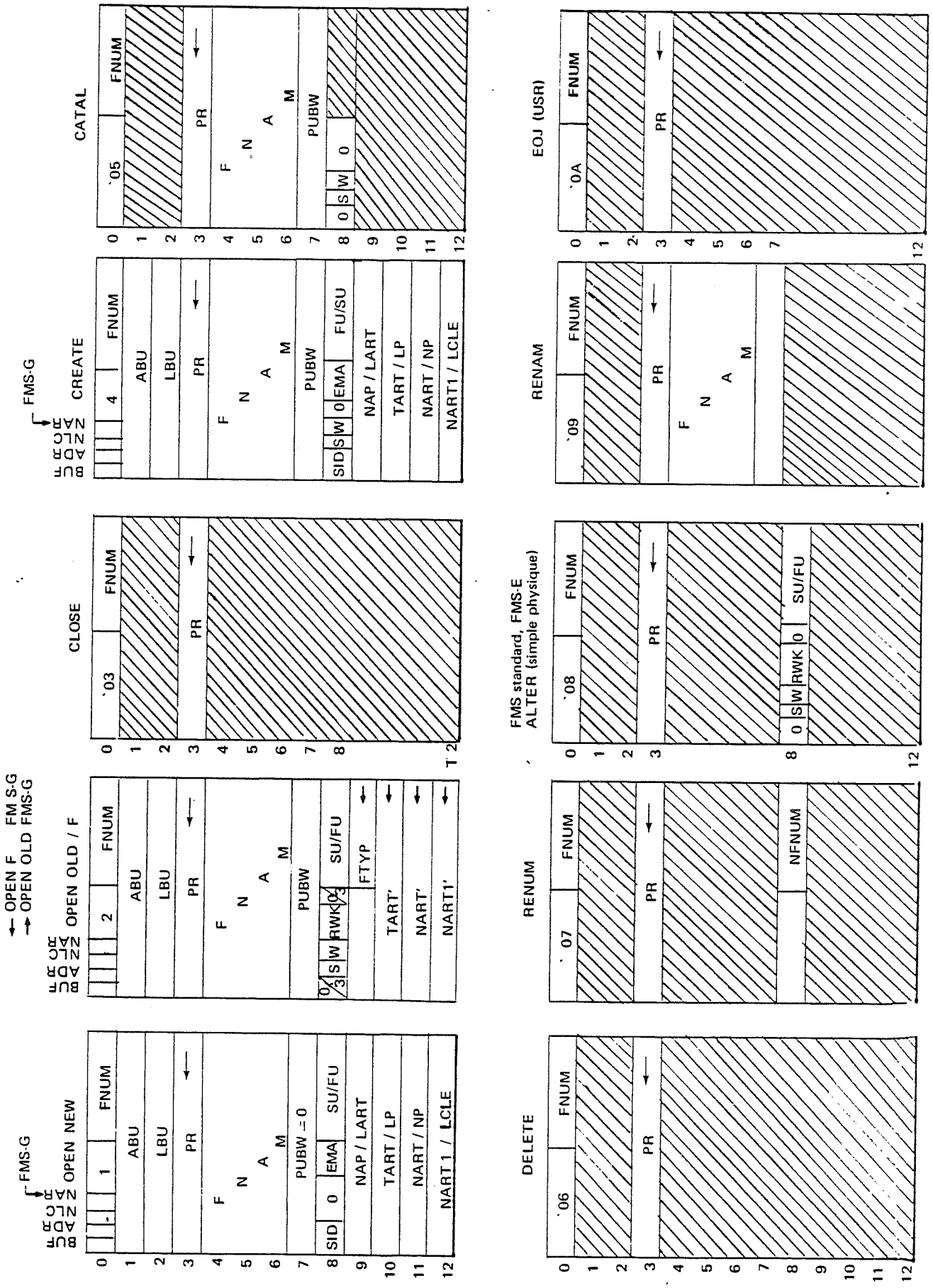
CORRESPONDANCE PRIMITIVE COMPTE-RENDU

Compte-rendu / Primitive	0	LBU	'1	'2	'3	'4	'5	'6	'7	'A	'B	'C	'D	'E	'F	'14	'15	'16	'17
OPEN NEW	X										X		X						X
OPEN OLD	X										X	X				X	X		
CLOSE	X									X									
CREAT	X										X		X						X
CATAL	X									X			X						
DELET	X									X						X			
RENUM	X									X	X								
ALTER	X									X						X			
RENAM	X									X			X			X			
EOJ	X																		
READ	X	X	X							X									
WRITE	X	X	X							X						X			X
SKIPB	X	X		X						X									
SKIPF	X	X	X							X									
REWIND	X									X									
SKEOA	X									X									
IREAD	X				X	X				X				X					
IWRITE	X									X					X	X		X	X
ISUP	X									X				X		X			
IRNAM	X									X					X	X			
IRWRITE	X				X	X				X				X		X			
IRTIX	X				X	X				X									
DREAD	X				X	X				X				X					
DWRITE	X				X	X				X				X		X			
DCRE	X				X	X				X				X		X		X	
DSUP	X									X				X		X			
SIREAD	X				X	X				X				X					
SIRIS	X		X	X	X	X		X	X	X									
SIWRIT	X				X	X	X			X						X			
SIADD	X				X	X				X					X	X		X	
SISUP	X						X			X						X			
SCADD	X									X				X		X		X	
SCREAD	X							X		X				X					
SCWRIT	X									X						X			
SCDREAD	X									X				X					

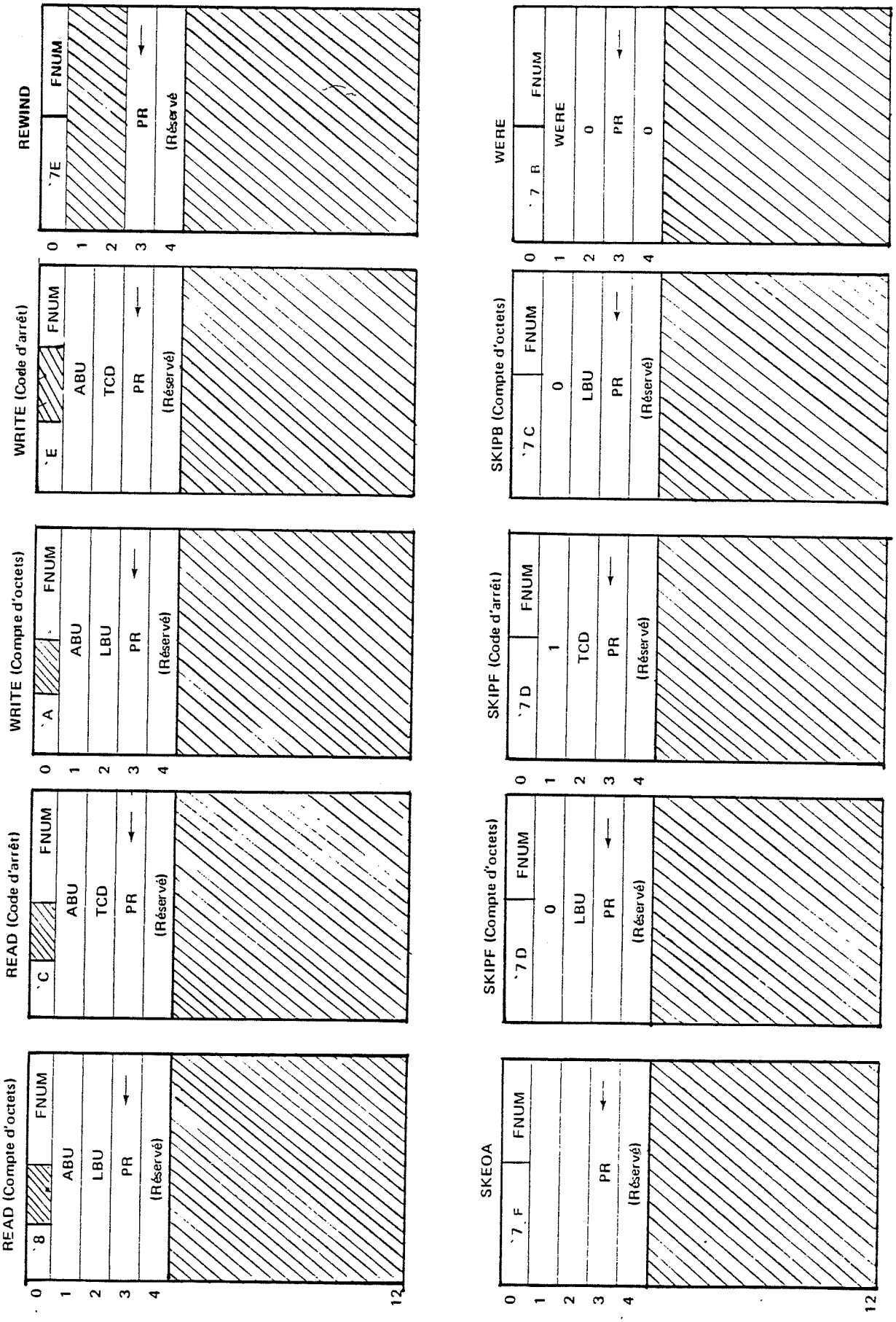
CORRESPONDANCE PRIMITIVE COMPTE-RENDU

Compte-rendu / Primitive	'18	'1A	'1E	'1F	'20	'21	'22	'28	'29	'2A	'2B	'32	'33	'34	'35	'36	'37	Hard
OPEN NEW				X	X	X		X	X	X	X	X	X	X	X	X	X	X
OPEN OLD		X	X	X	X			X	X	X	X	X		X	X	X	X	X
CLOSE				X				X	X		X	X	X	X	X	X	X	X
CREAT				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CATAL	X			X			X	X	X		X	X		X	X	X	X	X
DELET			X	X				X	X		X	X	X	X	X	X	X	X
RENUM				X				X	X		X					X	X	
ALTER	X		X	X				X	X	X	X			X	X	X	X	X
RENAM	X		X	X				X	X		X	X		X	X	X	X	X
EOJ								X	X		X	X	X	X	X	X	X	X
READ		X		X				X	X		X	X		X	X	X	X	X
WRITE		X		X		X		X	X		X	X	X	X	X	X	X	X
SKIPB		X		X				X	X		X	X		X	X	X	X	X
SKIPF		X		X				X	X		X	X		X	X	X	X	X
REWIND		X		X				X	X		X	X		X	X	X	X	X
SKEOA		X		X				X	X		X	X		X	X	X	X	X
IREAD	X			X				X	X		X	X		X	X	X	X	X
IWRITE	X			X		X		X	X		X	X		X	X	X	X	X
ISUP	X			X				X	X		X	X		X	X	X	X	X
IRNAM	X	X		X				X	X		X	X		X	X	X	X	X
IRWRITE	X			X				X	X		X	X		X	X	X	X	X
IRTIX	X			X				X	X		X	X		X	X	X	X	X
DREAD	X			X				X	X		X	X		X	X	X	X	X
DWRITE	X			X				X	X		X	X		X	X	X	X	X
DCRE	X			X				X	X		X	X		X	X			X
DSUP	X			X				X	X		X	X		X	X			X
SIREAD	X			X				X	X		X	X		X	X			X
SIRIS	X	X		X				X	X		X	X		X	X			X
SIWRIT	X	X		X				X	X		X	X		X	X			X
SIADD	X			X				X	X		X	X		X	X			X
SISUP	X	X		X				X	X		X	X		X	X			X
SCADD	X	X		X				X	X		X	X		X	X			X
SCREAD	X			X				X	X		X	X		X	X			X
SCWRIT	X	X		X				X	X		X	X		X	X			X
SCDREAD	X			X				X	X		X	X		X	X			X

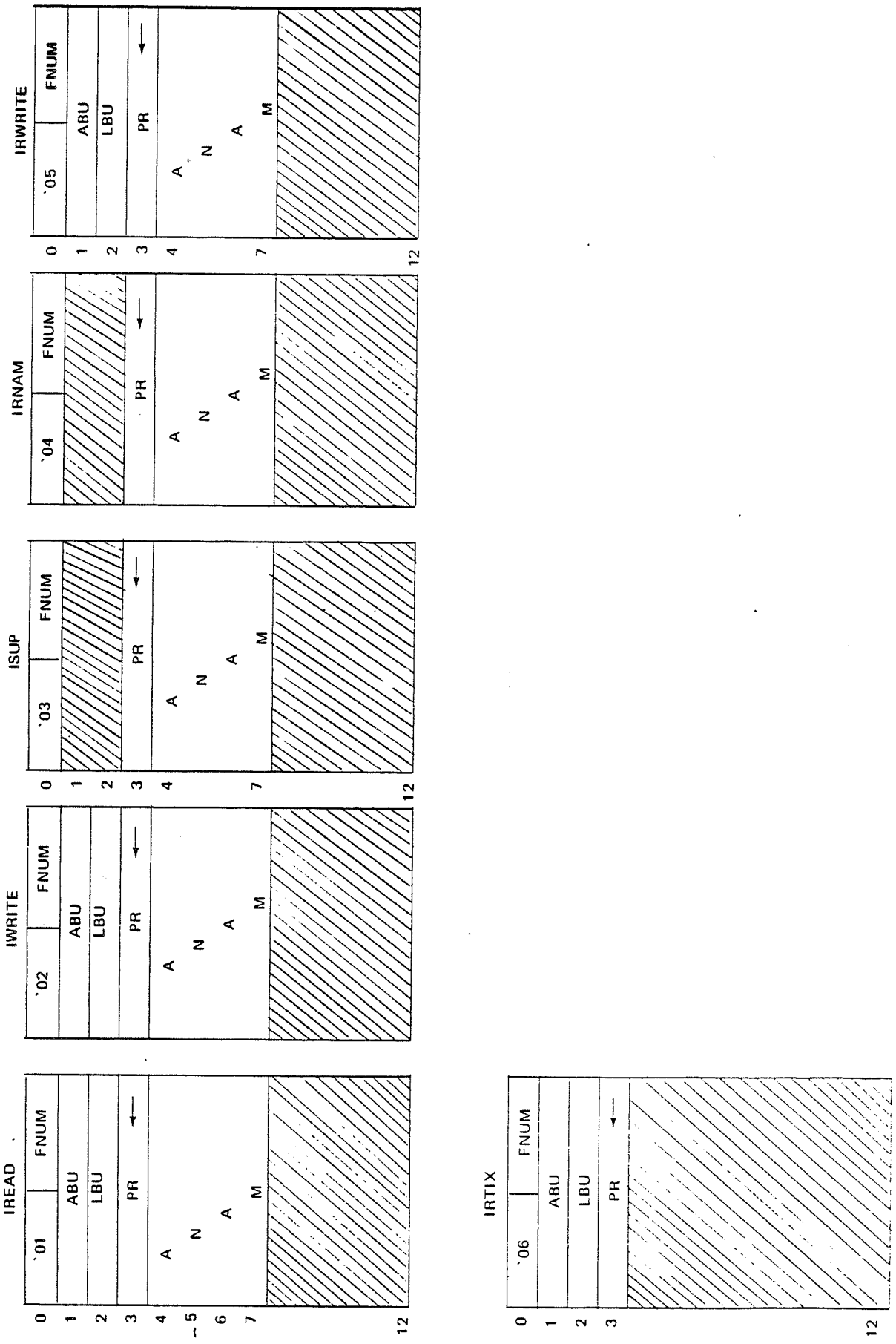
NIVEAU OPEN-CLOSE



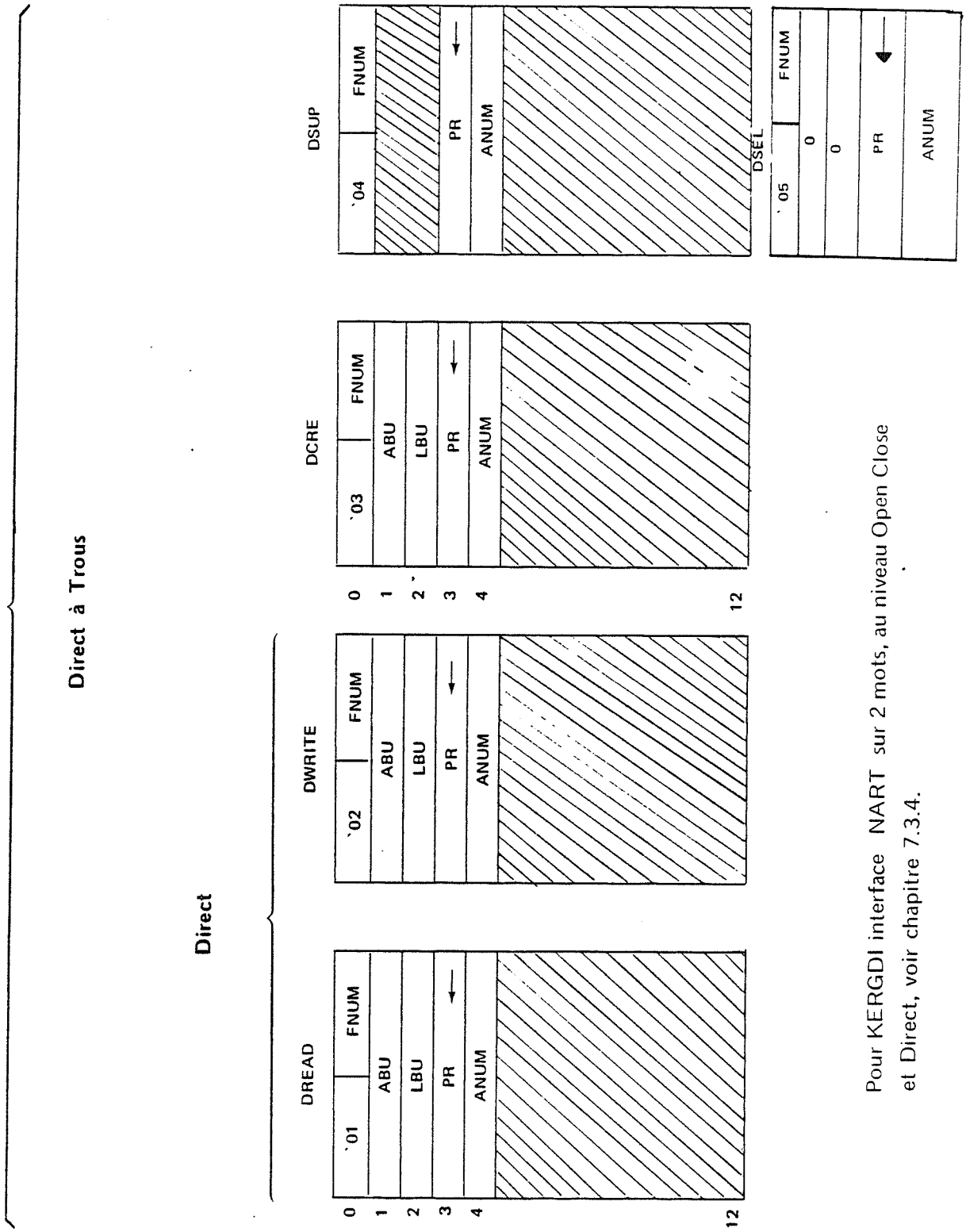
Niveau portion d'article



Niveau article d'un fichier indexé



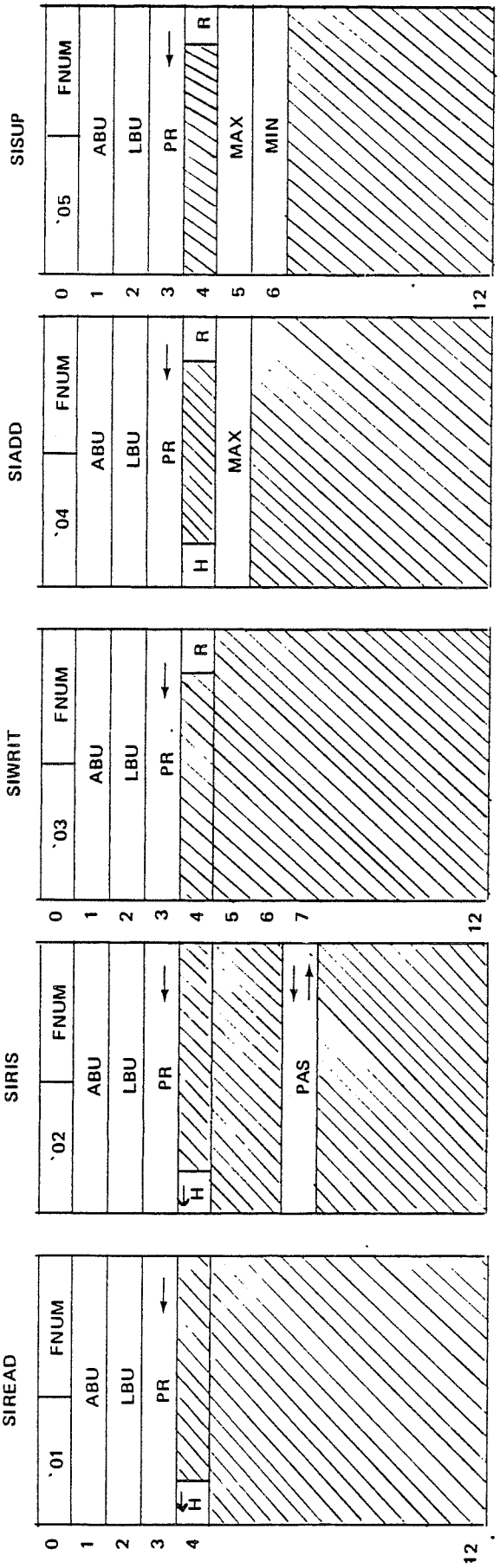
Niveau article de fichiers Direct et Direct à Trous



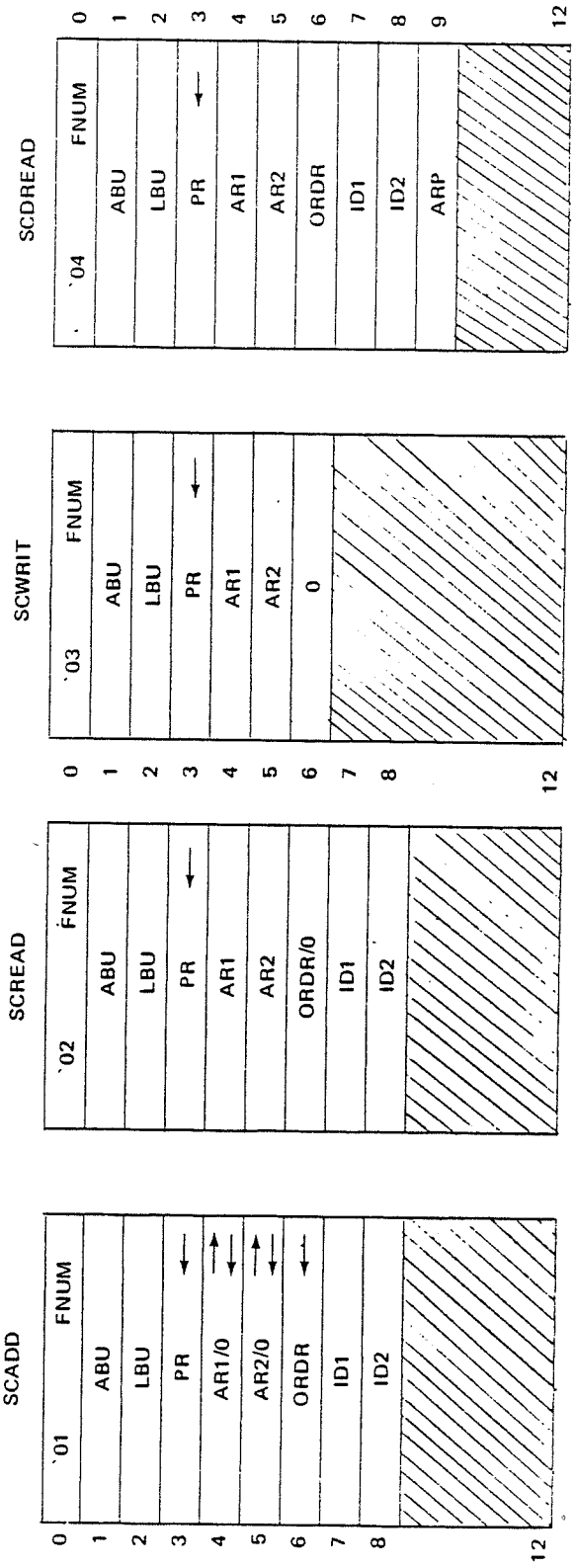
Pour KERGDI interface NART sur 2 mots, au niveau Open Close et Direct, voir chapitre 7.3.4.

Bull

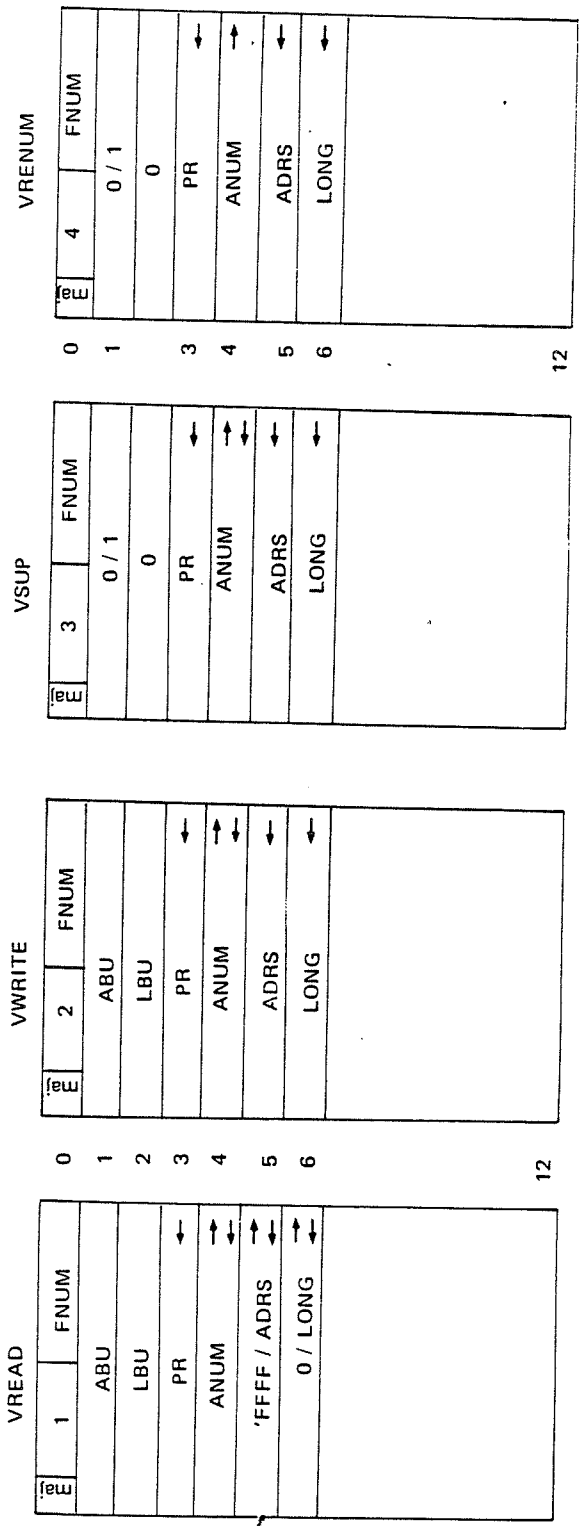
Niveau article d'un fichier Séquentiel Indexé



Niveau article d'un fichier Séquentiel Chaîné



Niveau article d'un fichier Direct Longueur Variable



Valeurs limites liées à la gestion des disques par les noyaux de FMS

	Taille minimum	Taille maximum
E/S IOCS	1 secteur Remarque : R/W du 1er mot d'un secteur, avec en écriture RAZ de la fin du secteur	Limite IOCS : ou 8 K mots - 1 6 K sur disque cartouche 5 M. oct.
E/S Article	0 ou 1 mot	KERADR { 32 K mots - 1 si TG < limite IOCS limite IOCS si TG > limite IOCS KERGDI 32 K mots - 1
E/S Portion d'article	0 ou 1 mot	KERADR limite IOCS KERGDI 8 K mots - 1
Buffer de travail	1 secteur	voir méthode d'accès ou limite IOCS 32 K mots - 1
E/S demandée à IOCS par FMS	1 secteur ou le 1er mot d'un secteur	INF { - ou [demande utilisateur / buffer de travail] - Taille utile du granule - limite IOCS
FIFI	3 secteurs	KERADR 192 secteurs KERGDI ≃ 400 K mots
TAG	- occupation 1 secteur - LTAG = 1 mot chaîne de bit KERADR + 3 mots système KERGDI + 7 mots système	KERADR 1 secteur LTAG = 125 mots KERGDI 2048 mots LTAG = 2041 mots

Valeurs limites liées à la gestion des disques par les noyaux de FMS

	Taille minimum	Taille maximum	Nombre minimum	Nombre maximum
FU IOCS	1 cylindre	NBCYL Nombre total de cylindres du disque	2 D1 D2	29 D1 à DF, E1 à EB, ED, EF
Secteur	LSEC = 128 mots	128 mots		32 767 par FU 2^{31} / LSEC
FU FMS	FU IOCS	KERADR { NBCYL - 1 ; 4 méga-mots } KERGDI { NBCYL ; 2 milliards de mots }	1 D2	28 D2 à DF, E1 à EB, ED, EF
Granule	3 secteurs	KERADR 256 Sect. KERGDI 32 K - 1 Sect.	KERADR 2 KERGDI 1	KERADR 2000 KERGDI 32656
Fichiers	Logiquement : 0 mot Physiquement : 1 granule	**Log.** voir Méth. d'accès KERADR :4 méga-mots KERGDI :2 milliard de mots **Phys.** 128 granules pour les fichiers statiques	1 par FU (ou espace disque)	KERADR 1999 KERGDI 32656

Valeurs limites des fichiers dues au logiciel KERADR

NOL	Méthodes d'accès Statique ou Dynamique	3 chiffres dans l'ordre Min, Raisonnable, Max		
		Taille d'article	Nombre d'articles	Taille du fichier
0	Séquentiel Dyn.	Portion d'article - 0 mot - n. secteurs - 8 K mots ou limite IOCS si TG lui est supérieur	1 " "	0 mots illimité "
1	Indexé Dyn.	1 mot 6 K mots 64 K mots	0 600 8159	Long. TIX 100 K mots 2 M mots ÷ LTX
2	Direct Stat.	1 mot 32 K mots "	1 65535 "	1 mot 4 méga-mots "
2	Direct à trous Stat.	Idem DIR	Idem DIR	Idem DIR
3	Séquentiel indexé Stat.	1 mot 100 mots 2 K Mots	1 100 000 200 000 ≃	3 secteurs 4 méga-mots 32 767 postes de 128 mots
4	Séquentiel chaîné Stat.	1 mot 10 mots 12 K mots	Nombre de chaînes 1 20 000 32 767	2 secteurs 2 M. mots (32 767 postes de 64 mots, ou 344 postes de de 12 K mots)
5	Direct Longueur variable Dyn.	1 mot 6 K mots 32 K mots	0 1 000 16 384 (LIX = 4)	Long TIX 200 K mots 32 K sect. + LTX

Valeurs limites des fichiers dûes au logiciel KERGDI

Méthodes d'accès NOL — Statique ou Dynamique	3 chiffres dans l'ordre Min, Raisonnable, Max Taille d'article	Nombre d'articles	Taille du fichier
0 Séquentiel Dyn.	Portion d'article 0 mot 4 secteurs 8 K mots	1 " "	0 mot illimité "
1 Indexé Stat.	1 mot 6 K mots 64 K mots	0 600 8 159	Long. TIX 100 K mots 2 M mots + LTX
2 Direct Stat.	1 mot 32 K mots "	1 65 535 "	1 mot 2 milliards mots "
2 Direct à trous Stat.	Idem DIR	1 2 milliards "	Idem DIR
3 Séquentiel indexé Stat.	1 mot 100 mots 2 K mots	1 100 000 200 000 ~	3 secteurs 145 M mots (65 535 postes de 6 K mots)
4 Séquentiel chaîné Stat.	1 mot 10 mots 12 K mots	Nombre de chaînes 1 20 000 32 767	2 secteurs 2 M mots (32 767 postes de 64 mots, ou 344 postes de 12 K mots)
5 Direct Longueur variable	1 mot 6 K mots 32 K mots	0 1 000 16 384	Long. TIX 200 K mots 32 K sect. + LTIX

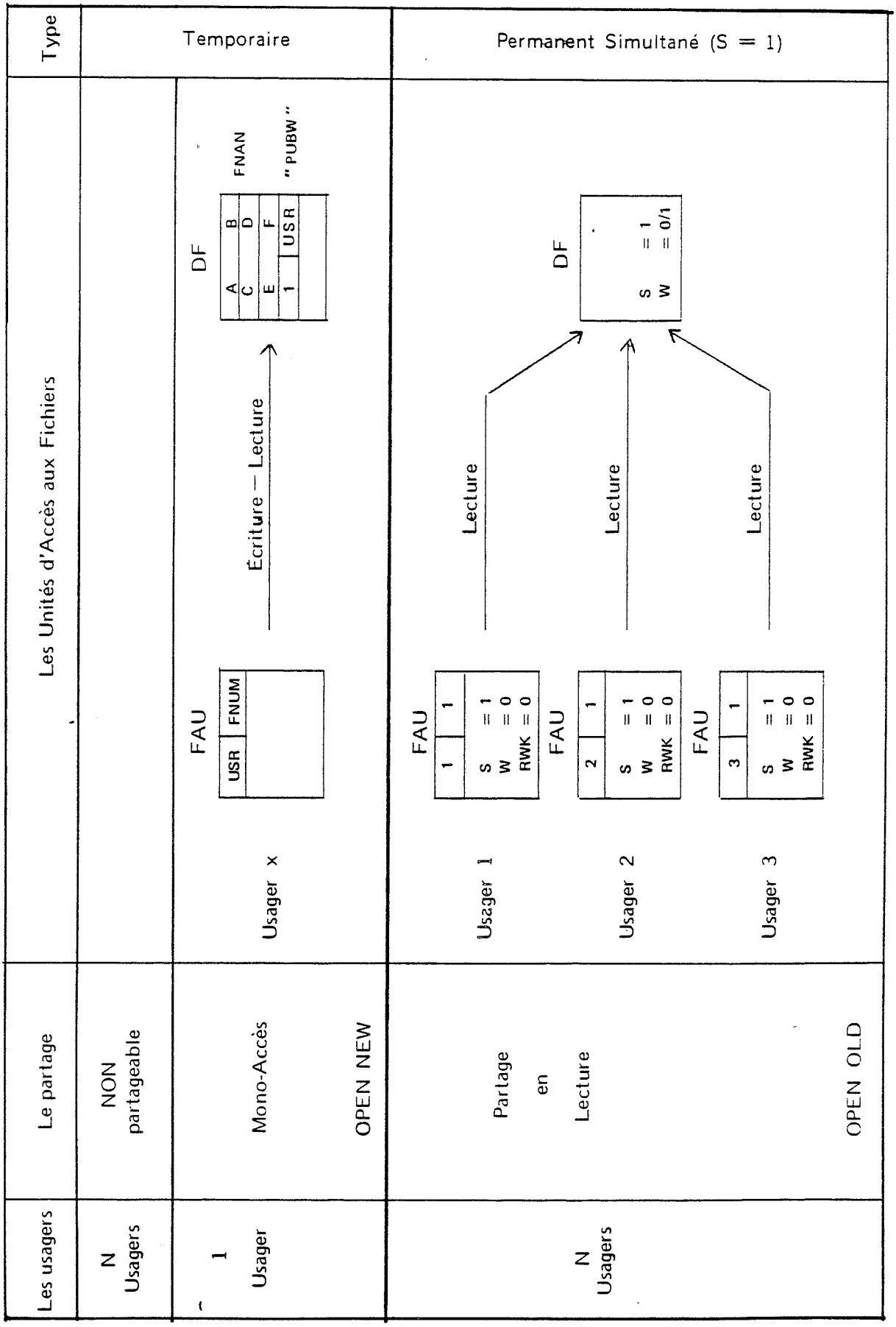
Taille des fichiers en nombre de secteurs, pour des secteurs de 128 mots

NBG / TG	1	2	4	5	8	10	16	20	32	50	64	100	128	200	256	500	512	1 000	1 024
1 K	7	14	28	35	56	70	112	140	224	350	448	700	896	1 400	1 792	3 500	3 581	7 000	7 168
2 K	15	30	60	75	120	150	240	300	480	750	960	1 500	1 920	3 000	3 840	7 500	7 680	15 000	15 360
3 K	23	46	92	115	184	230	368	460	736	1 150	1 472	2 300	2 944	4 600	5 888	11 500	11 776	23 000	23 552
4 K	31	62	124	155	248	310	496	620	992	1 550	1 984	3 100	3 968	6 200	7 936	15 500	15 872	31 000	31 744
6 K	47	94	188	235	376	470	752	940	1504	2 350	3 008	4 700	6 016	9 400	12 032	23 500	24 064	47 000	48 128
8 K	63	126	252	315	504	630	1 008	1 260	2 016	3 150	4 032	6 300	8 064	12 600	16 128	31 500	32 256	63 000	64 512
12 K	95	190	380	475	760	950	1 520	1 900	3 040	4 750	6 080	9 500	12 160	19 000	24 320	47 500	48 640	95 000	97 280
16 K	127	254	508	635	1 016	1 270	2 032	2 540	4 064	6 350	8 128	12 700	16 256	25 400	32 512	63 500	65 024	127 000	130 048
24 K	191	382	764	955	1 528	1 910	3 056	3 820	6 112	9 550	12 224	19 100	24 448	38 200	48 896	95 500	97 792	191 000	195 584
32 K	255	510	1 020	1 275	2 040	2 550	4 080	5 100	8 160	12 750	16 320	25 500	32 640	51 000	65 280	127 500	130 560	255 000	261 120

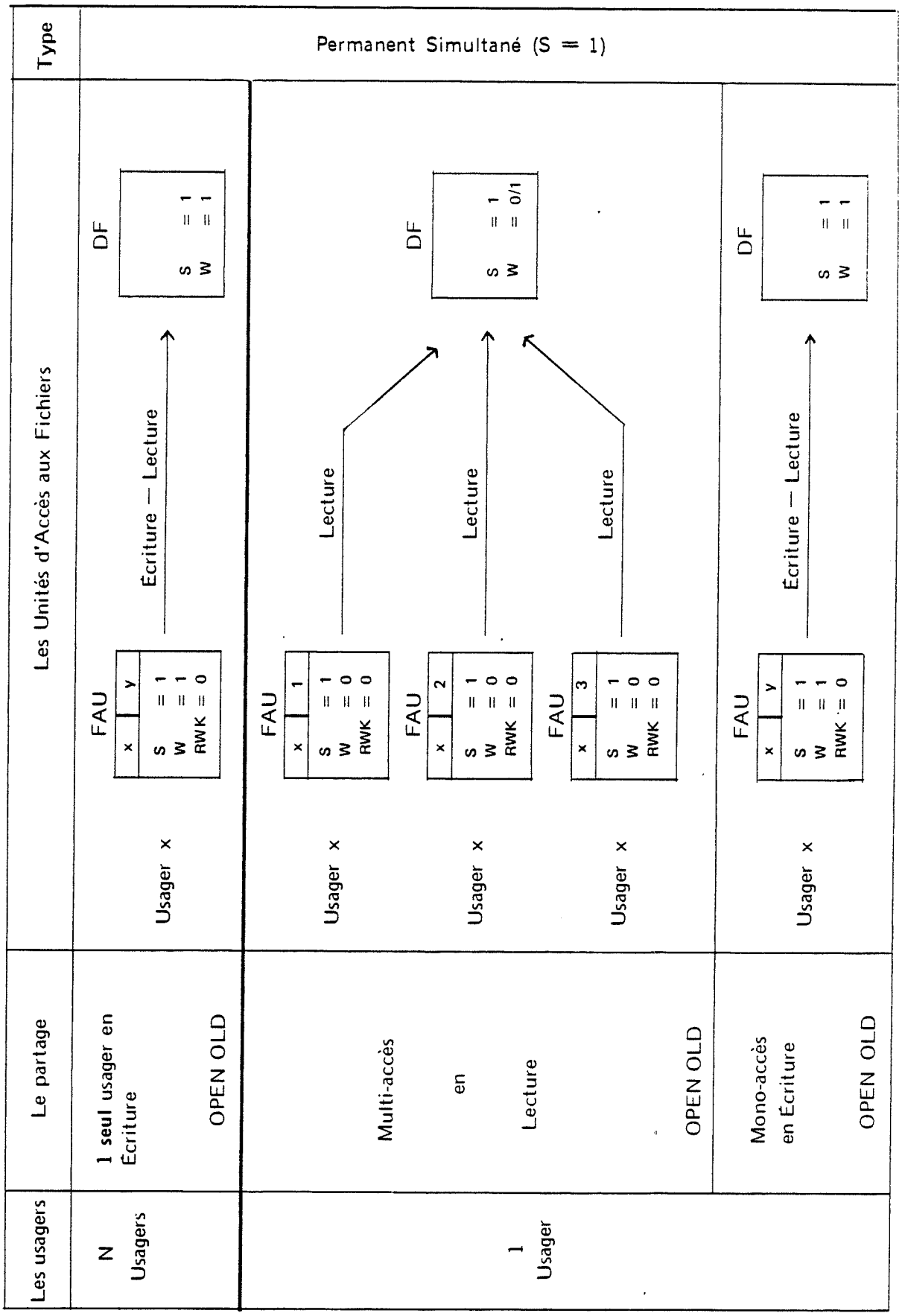
Limite de 399 cylindres de 6Kmots

Limite de 128 granules (fichiers statiques)

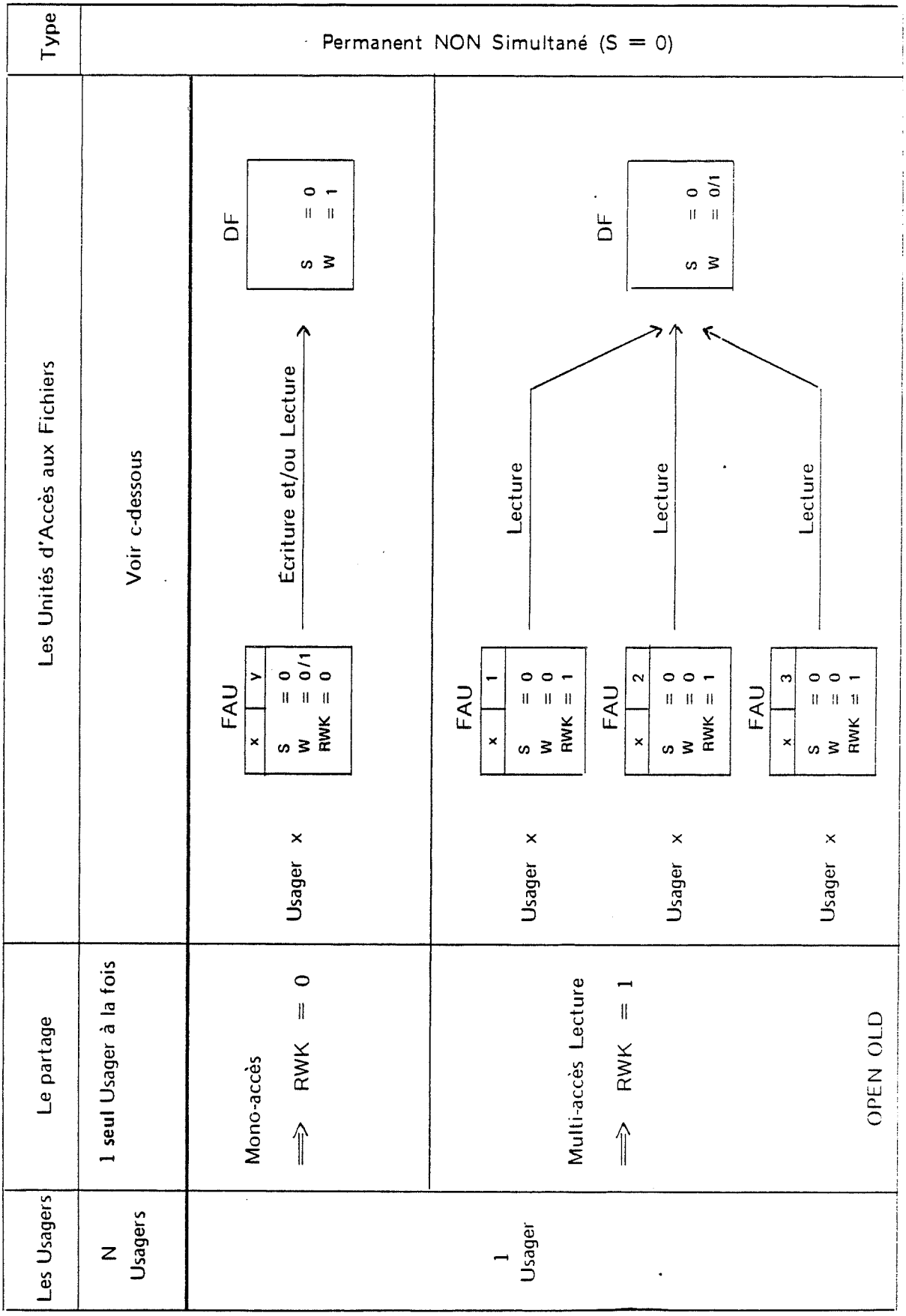
Le Partage des Fichiers

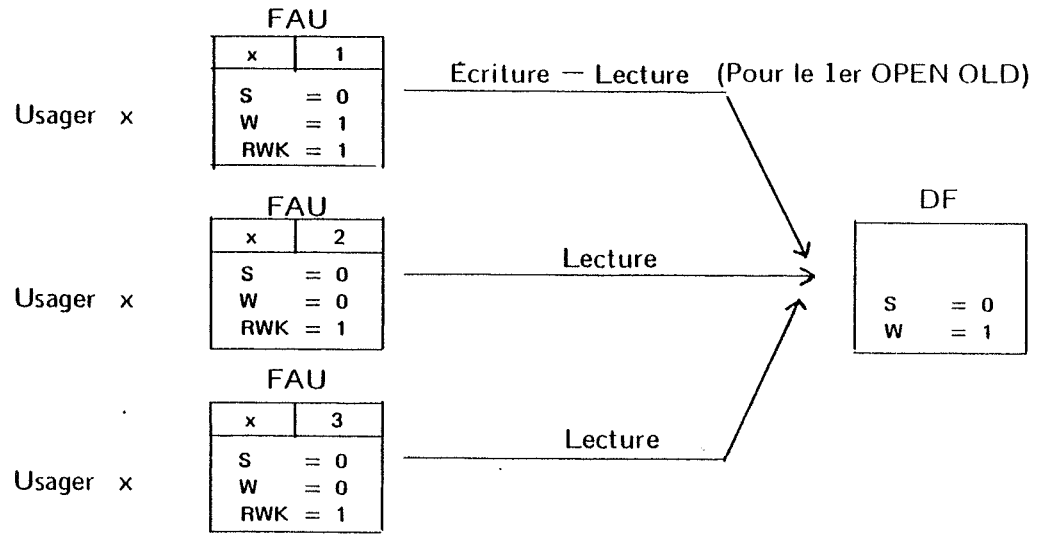
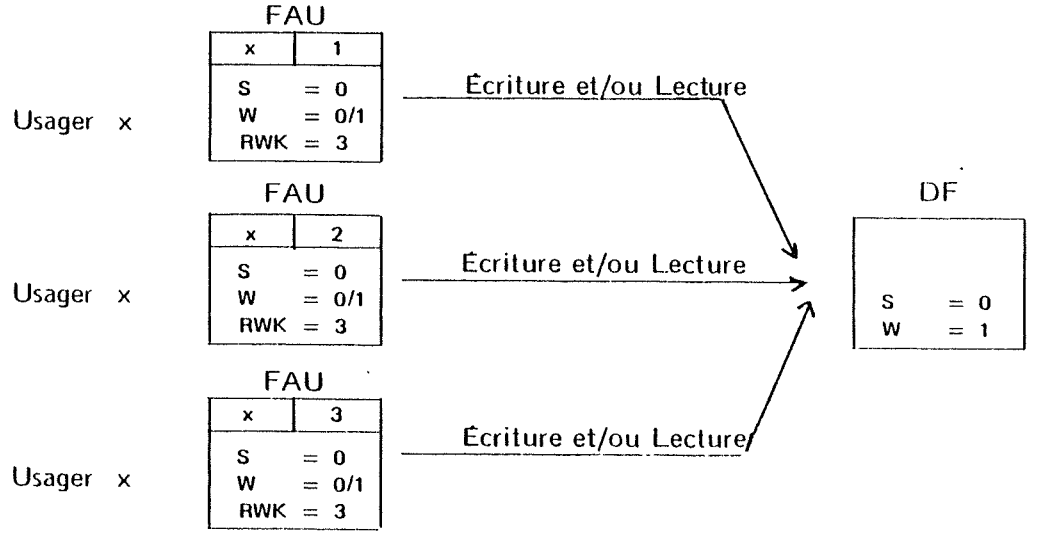


La Partage des Fichiers



Le Partage des Fichiers



Les Usagers	Le partage	Les Unités d'Accès aux Fichiers	Type
1 Usager	**Pour** : Direct et Direct à Trous Multi-accès Lecture ⟹ RWK = 1 Avec possibilité d'écriture, réservée à la 1ère ouverture dans le temps. OPEN OLD		Permanent NON Simultané (S = 0)
	Pour : Direct Multi-accès Écriture ⟹ RWK = 3 OPEN OLD		

C — INDEX

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W

Z

Bull Sems

VOS REMARQUES SUR CE DOCUMENT

TITRE

FMS16 ADDENDUM A

N° DE REFERENCE

Bull-Sems : 1 164 226 00 036 01

DATE

DECEMBRE 1985

ERREURS DETECTEES

AMELIORATIONS SUGGEREES

*→ Vos remarques et suggestions seront attentivement examinées.
si vous désirez une réponse écrite, veuillez indiquer ci-après
votre adresse postale complète.

NOM : ..... DATE .....

SOCIETE : .....

ADRESSE : .....

*→ Remettez cet imprimé à un responsable Bull-Sems ou envoyez le
directement à

Bull-Sems
Méthodes G.I.
Rue de Provence
38 130 - ECHIROLLES