Réalisation d'un système de gestion de fichiers en langage Pascal pour MODULECO

Robert Gardien

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE
DE GRENOBLE (C.U.E.F.A)

MEMOIRE

présenté en vue d'obtenir
le diplôme d'ingénieur

en
informatique

par

ROBERT GARDIEN

REALISATION D'UN SYSTEME DE GESTION DE FICHIERS EN LANGAGE PASCAL POUR MODULECO

SOUTENU LE :

JURY

Président : Professeurs L. BOLLIET
Membres : J.J. COMBIER
B. OUDET
F. RECHENMANN

Je tiens à remercier,

Monsieur le Professeur du Conservatoire National des Arts et Métiers qui a bien voulu présider le jury de ce mémoire,

Monsieur le Professeur L. BOLLIET, Directeur du GIS de Mini et Micro Informatique de Grenoble et Professeur à l'Université de Grenoble 2, pour le soutien qu'il m'a apporté et l'honneur qu'il me fait de participer au jury,

Monsieur B. OUDET, Professeur à l'Université Scientifique et Médicale de Grenoble, qui ne m'a pas ménagé ses encouragements et qui a accepté d'être membre du jury,

Monsieur F. RECHENMANN, Chercheur à l'INRIA pour l'aide et les conseils qu'il m'a donné, et qui a bien voulu être membre du jury,

Monsieur J.J. COMBIER, Ingénieur en Chef à CAP SOGETI LOGICIEL, qui a bien voulu faire partie du jury.

Je remercie également tous les membres de l'équipe MODULECO qui m'ont apporté leurs conseils tout au long de la réalisation de ce travail.

Je voudrais remercier mon épouse Michelle, pour la patience dont elle a fait preuve pendant l'année de réalisation de ce mémoire.

Je voudrais aussi remercier D. IGLESIAS et le Service Reprographie, pour le soin apporté au tirage de ce texte.

PLAN

INTRODUCTION

MODULECO est un logiciel de construction et d'utilisation de modèles macroéconomiques. Il a été développé dans le cadre d'une Association de loi 1901 à but non lucratif : le Club MODULECO. L'Association créée en février 1979 regroupe à l'heure actuelle des représentants des Laboratoires de recherches INRIA et IMAG, des représentants de l'INSEE, de la Direction de la Prévision, du Parlement (Assemblée Nationale et Sénat), du Centre Interuniversitaire de Calcul de Grenoble et de l'Agence de l'Informatique.

MODULECO offre un ensemble intégré de fonctions qui permettent à l'économiste d'effectuer les différentes tâches de la modélisation : gestion de séries chronologiques, entrée et modification du texte des équations d'un modèle, estimation des coefficients, impression ou visualisation des résultats.

Les logiciels de modélisation économétrique existants ont été étudiés dans <9> et <4>. MODULECO s'en distingue par le fait qu'il est disponible sur plusieurs machines et systèmes d'exploitation : HB68 (MULTICS), IBM série 370, 3032 (OS/MVS).

Le travail décrit dans ce mémoire s'est déroulé dans le cadre du projet MODULECO. Il s'agissait de fournir un module de gestion

de fichiers à l'équipe chargée de l'impléementation du logiciel.

Deux problèmes principaux se posaient pour la gestion des fichiers de MODULECO :

- éviter la multiplicité de ceux-ci dans le système hôte, ce qui pouvait provoquer une saturation des tables du système.

- résoudre les problèmes posés par PASCAL lequel étant un langage typé obligeait à une prolifération de procédures de lecture-écriture. Il aurait fallu un couple de procédures par fichier différent.

- avoir un ensemble de méthodes d'accès qui sont nécessaires au projet et qui ne sont pas standard sur tous les systèmes.

Pour présenter la solution retenue, le plan suivant est adopté:

- le chapitre 1 présente le logiciel MODULECO avec les différentes couches. Ce découpage permet d'assurer un minimum de travail à l'équipe chargée de porter le logiciel sur un autre site que celui sur lequel il a été développé initialement.

- Dans le chapitre 2 sont étudiés les besoins de chaque sous-système de MODULECO pour la gestion des différents objets à gérer ainsi que la solution proposée.

- le chapitre 3 montre les avantages de cette solution par rapport à la portabilité du logiciel.

- le chapitre 4 présente les améliorations apportées au système de gestion de fichier, pour le rendre utilisable d'une manière autonome.

- En annexe nous trouvons les Spécifications de Réalisation du système de gestion de fichier.

1. PRESENTATION DU LOGICIEL MODULECO

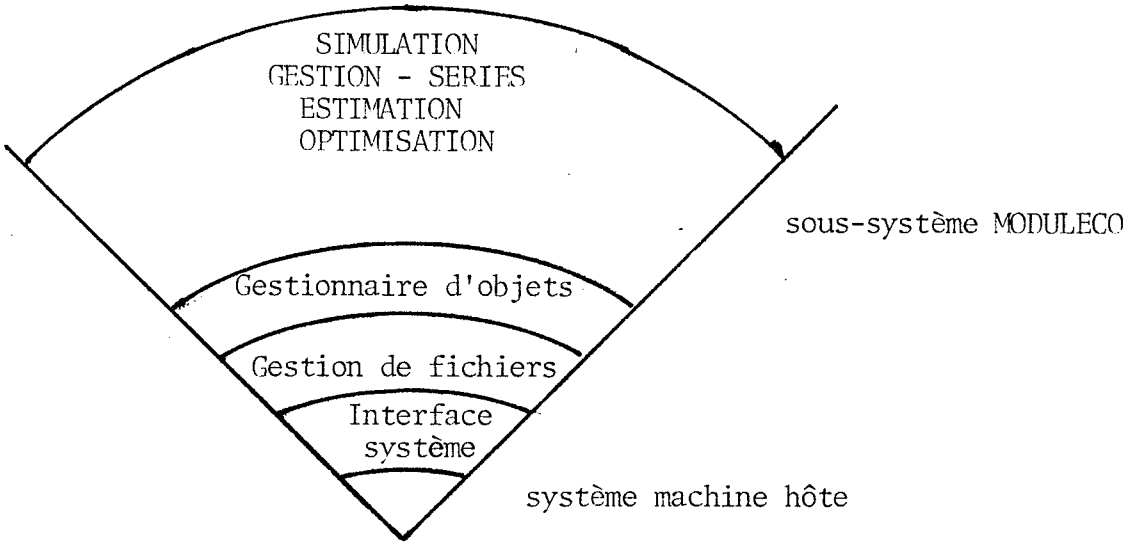
Comme beaucoup de systèmes informatiques MODULECO peut être considéré sous deux aspects principaux <3> :

- Interpréteur d'un langage. Le système MODULECO est l'interpréteur du "langage MODULECO" <14> <13> et du langage externe d'écriture des équations (L3E) <19> à travers lesquels les fonctions de modélisation économiques, de recherche et de transformation de données dans la base, de compilation,etc..., sont exécutées.

- Allocateur de ressources. Pour le système MODULECO les ressources sont constituées des annuaires, des segments et des liens gérés par le Gestionnaire des Objets MODULECO <18>.

MODULECO a été réalisé selon la technique des logiciels en couche. Une couche, vue de l'extérieur est une boîte noire qui offre des services. Chaque couche peut utiliser les services de la couche immédiatement inférieure, en plus de ses fonctions propres, pour créer de nouveaux services utilisables par la couche immédiatement supérieure.

L'ensemble des règles et des formats (syntaxiques ou sémantiques) au moyen desquels une couche utilise les services offerts par la couche immédiatement inférieure est appelée une interface.



Le schéma ci-dessus montre le découpage en couches de MODULECO. Cette réalisation en couches est très importante car elle simplifie énormément le travail de réalisation, de maintenance et de transport par la suite. Lors des spécifications du logiciel, il faut préciser les interfaces entre les différentes couches. Ceci permet par la suite de pouvoir réaliser une couche sans attendre les autres ou même plus, de remplacer une couche. Pourvu que l'interface soit respectée, ce changement n'affecte en rien les autres couches.

Au niveau le plus bas se trouve le système d'exploitation de la machine hôte; puis vient le module d'Interface avec le système hôte; ensuite le système de Gestion de fichier que nous allons détailler dans le chapitre 2 puis le gestionnaire d'Objets. Enfin

au-dessus de ces couches qui peuvent être considérées comme purement systèmes, sont réalisés les différents sous-systèmes qui permettent à l'économiste d'utiliser ses modèles macroéconomiques:

- Simulation et gestion de Modèle

Le sous-système de simulation <22> <23> <17> permet de construire des Modèles et de calculer, par résolution des équations de ceux-ci, les valeurs que prennent les variables endogènes pour des valeurs données des variables d'entrée.

Pour entrer le texte des équations d'un modèle, l'économiste dispose d'un langage particulier, le Langage Externe d'Ecriture des Equations, LEEE <19>.

Un exemple d'équation économique est celle de la consommation :

Ct = C1 * Yt + C2 * Yt-1

qui exprime que la consommation est fonction du revenu aux temps t et t-1.

Le texte source en LEEE est transformé en une représentation des équations sous forme d'arbre. Pour le calcul des équations, c'est cette représentation qui est fournie à l'interpréteur du langage LEEE.

- Estimation

Le sous-système d'estimation <21> permet d'obtenir la valeur des coefficients des équations du modèle à partir des valeurs contenues dans les séries chronologiques.

L'estimation est constituée d'un ensemble d'algorithmes classiques (moindres carrés simples et généralisés, corrections des corrélations des résidus, retards échelonnés d'Almon) d'estimation équation par équation. L'estimation se fait à partir du texte en LEEE d'une équation.

- Optimisation

Le sous-système d'optimisation <20> permet d'inclure à MODULECO des programmes d'optimisation capables de traiter des modèles de grande taille.

- Gestion des séries

Le sous-système de gestion de séries <24> <25> <12> met à la disposition des autres sous-systèmes des fonctions permettant de consulter, mettre à jour ou créer des séries chronologiques.

Rappelons qu'une série chronologique est la suite des valeurs (observations) que prend une variable économique pour un certain ensemble de dates. La suite de valeurs

PNB(1975), PNB(1976), PNB(1977) est la série du produit national brut de calendrier annuel.

Une originalité de MODULECO est que les séries peuvent avoir de une à trois dimensions en plus de la dimension temporelle. Ainsi, par exemple, dans un modèle mondial, la série PNB aura pour chaque date autant de valeurs qu'il y a de pays dans le modèle : PNB(FRANCE,1979),PNB(ITALIE,1979), etc... Ce sont ces noms de pays qui peuvent constituer une nomenclature.

PAYS = (FRANCE,ITALIE,JAPON)
est la nomenclature de nom PAYS.

Toutes les interfaces système <15> qui constituent la partie non portable du système sont actuellement réalisées sur HB68 (MULTICS) et sur IBM (OS/MVS).

C'est à travers le langage MODULECO <16> que l'utilisateur exprime ses besoins et les tâches qu'il veut voir accomplir. Le langage MODULECO propose un ensemble initial de commandes, mais l'utilisateur peut étendre la puissance du langage de deux manières :

- en combinant plusieurs commandes entre elles pour en former de nouvelles. (ceci est particulièrement aisé car MODULECO est un langage interprété).

- ou en créant de nouvelles commandes personnelles et en "ajoutant" le programme correspondant à la bibliothèque des programmes de MODULECO. Outre le fait que ceci permet à l'utilisateur de développer ses propres algorithmes et de les exécuter dans le contexte MODULECO, il est à signaler que les programmes ajoutés peuvent être écrits dans un autre langage que PASCAL sans que cela n'entraine aucune recompilation ou édition de liens de MODULECO.

Ce dernier point constitue une des principales originalités de MODULECO.

Signalons enfin, le fait que le langage est entièrement documenté en interactif, afin de guider l'utilisateur pas à pas.

2. ANALYSE DU SYSTEME DE GESTION DE FICHIER

Les sous-systèmes présentés dans le chapitre précédent ont tous besoin soit pour lire les données, soit pour stocker les résultats d'utiliser les fichiers magnétiques. Le langage PASCAL standard (norme ISO) choisi pour implémenter MODULECO apporte de nombreux avantages <5> <10> :

- PASCAL possède des structures de données très évoluées
- il utilise très largement la notion symbolique (scalaires, ensembles), ce qui le rend relativement indépendant de la machine.
- il a une syntaxe très simple
- sa traduction est simple et par là-même efficace
- il est très lisible et facilite ainsi la maintenance et le transport des programmes
- étant typé, il permet à son traducteur de détecter un maximum d'erreurs avant toute exécution

Ces avantages permettent une expression et une mise au point plus rapide des programmes et assure la portabilité du logiciel. Cependant ce langage s'il apporte des facilités a aussi des insuffisances <11> :

- absence de tableaux à bornes dynamiques, obligation est faite de figer à la déclaration les bornes des tableaux.

- impossibilité d'initialiser des variables à la déclaration

- absence de l'opérateur d'exponentiation

- faiblesse des procédures d'entrées-sorties : PASCAL écrit dans un fichier des enregistrements de structure homogène et de même longueur, on ne peut écrire dans un fichier des enregistrements de types différents et/ou de longueurs distinctes. Comme nous le verrons plus loin ceci est très important dans le cadre de notre réalisation.

  Dans le PASCAL standard l'accès direct n'est pas prévu, or il était indispensable pour réaliser MODULECO. De plus, en PASCAL norme SOL, qui comporte des primitives pour l'accès direct, l'occupation d'espace disque n'est pas optimisée : PASCAL écrit des enregistrements de longueur X dans des enregistrements physiques de longueur Y. Si X est petit devant Y, la place perdue est importante.

D'un autre côté, une gestion des Entrées-Sorties faite directement par les utilisateurs aurait posé un autre problème: la prolifération des fichiers qui peuvent être très nombreux (Fichiers de séries chronologiques de la base de données, modèles, différentes variantes, etc...), ceci aurait pu être gênant dans certains systèmes d'exploitation.

2.1. Analyse des besoins des différents sous-systèmes

Dans ce paragraphe nous étudions les besoins des différents sous-systèmes afin de faire une synthèse, pour approcher la meilleure solution et faciliter ainsi au maximum le travail de développement du logiciel.

2.1.1. Les objets de MODULECO

Pour la gestion des objets et des utilisateurs, tout MODULECO est chapeauté par le fichier système SYSFIC. Dans ce fichier il est nécessaire de stocker les descripteurs utilisateurs. Il faut pouvoir retrouver ce descripteur lors du "Login", mais aussi créer de nouveaux utilisateurs et supprimer ceux qui ne veulent plus utiliser MODULECO.

Dans son environnement chaque utilisateur possède plusieurs sortes d'objets dont une liste exhaustive est donnée ci-dessous :

- des segments qui peuvent être considérés comme des fichiers où sont rangées les données de MODULECO. Par la suite nous en verrons une définition plus précise avec le développement de la solution proposée.

- un ou des annuaires qui sont simplement des catalogues soit de segments, soit d'autres annuaires permettant de structurer l'espace d'adressage de l'utilisateur <8>.

- de liens qui permettent d'avoir accès, depuis une branche de l'arborescence, à des objets situés sur une autre branche. On peut ainsi utiliser un même objet dans des environnements différents.

- des listes d'accès associés aux objets. Un utilisateur ne peut référencer un objet situé hors de son environnement personnel que s'il y a été autorisé. Une liste d'accès peut-être attachée à un annuaire ou à un segment, elle est constituée de la liste des noms d'utilisateurs qui sont autorisés à utiliser l'objet.

- des chaînes de documentation qui sont référencées depuis les annuaires, les segments ou même les séries chronologiques de la base de données.

2.1.2. Les données dans MODULECO

Il faut maintenant expliquer plus en détail l'utilisation qui va être faite des fichiers pour exécuter les différentes commandes. Les sous-systèmes de MODULECO peuvent utiliser des fichiers pour stocker plusieurs sortes de données :

modèles, commandes, séries chronologiques personnelles.

Les besoins sont différents selon le type de données manipulées. Nous allons successivement les recenser.

2.1.2.1. Modèles

Pour gérer les modèles plusieurs sortes de fichiers sont nécessaires :

- Table des symboles utilisés pour la compilation du langage LEEE. Les éléments de la table des symboles sont répartis en classes selon un Hashcode approprié. A l'intérieur de chaque classes les éléments sont rangés par ordre alphabétique.

- Table des équations. Le langage L3E est un langage conversationnel qui peut-être utilisé pour corriger un modèle équation par équation. Il y a donc une sorte de catalogue pour les équations, qui permet de les retrouver soit dans le fichier des arbres d'équations, soit dans le fichier contenant le source de celles-ci.

- Les séries chronologiques sont stockées par noms et leurs valeurs par date. Pour exécuter une simulation il est interessant d'avoir, afin d'amèliorer les temps de calcul, à chaque instant toutes les variables pour une même date. Ce

besoin est résolu par les segments transposés. Ces segments sont répartis en deux fichiers, un qui contient les noms des variables et un deuxième qui contient leurs valeurs.

2.1.2.2. Commandes

Toutes les commandes de MODULECO sont aussi gérées comme des données; un stockage est donc nécessaire :

- Catalogue. Dans ce fichier est enregistré le nom de la commande et son abréviation, ainsi qu'un pointeur qui permet de retrouver directement le descripteur de la commande dans le fichier adhoc. Dans le catalogue les commandes sont réparties en classe selon un Hashcode. A l'intérieur d'une même classe elles sont classées par ordre alphabétique.

- Dicco. Dans ce fichier sont rangés les descripteurs de commande. C'est dans ces descripteurs que l'on trouve des pointeurs qui permettent de retrouver la table des symboles, les options, la valeur des constantes et l'arbre représentant la commande. Toutes ces données se trouvent donc réparties dans des fichiers différents.

2.1.2.3 Séries chronologiques

Les séries chronologiques sont stockées dans trois fichiers :

- Catalogue des séries qui est organisé en classe selon un calcul de Hashcode. Ce catalogue contient des pointeurs qui permettent de retrouver les valeurs de la série.

- Fichiers des valeurs des séries chronologiques. Le catalogue pointe sur la première valeur de la série, les autres sont à lire séquentiellement. Pour le fichier des valeurs, la gestion en langage PASCAL est plus compliquée, car le nombre de valeurs d'une série n'est pas connu d'avance. Il ne sera connu qu'à l'exécution.

- Fichiers des nomenclatures. Ce sont simplement des identificateurs qui vont permettre d'indexer les séries à plusieurs dimensions avec des index explicites. Ces nomenclatures sont stockées dans deux fichiers : un catalogue qui pointe sur les valeurs de la nomenclature qui se trouvent dans le deuxième fichier. Ce catalogue était nécessaire car une nomenclature peut servir à plusieurs séries chronologiques existantes ou même à venir.

2.2 Présentation de la solution proposée

Deux problèmes étaient posés pour la gestion des fichiers dans le projet MODULECO :

- la multiplicité des fichiers
- le fait que le langage choisi pour l'implémentation soit typé.

2.2.1. Multiplicité des fichiers.

Un point important à résoudre était d'éviter la prolifération des fichiers sur le système hôte. Ceux-ci peuvent en effet être très nombreux (voir le paragraphe 2.1), fichiers de séries chronologiques, modèles, commandes, etc...). Cette multitude de fichiers pouvait entrainer la saturation des tables du système machine hôte. Pour pallier à cet inconvénient, il n'y a qu'un seul fichier physique machine hôte par utilisateur.

Afin de permettre à l'utilisateur de ne pas être pénalisé par ce fait, le fichier est partitionné en segments. Ce sont ces segments seuls qui représenteront les fichiers nécessaires aux différents sous-systèmes. L'utilisateur n'a pas à connaitre le fichier physique; il ne référence que des segments qui sont, de son point de vue, autant de fichiers différents.

2.2.2. Problème du langage

Le fait de faire gérer les entrées-sorties par un module distinct permet de s'affranchir des types de PASCAL. En effet sur les compilateurs existants il n'existe aucun contrôle de type dans le cas de compilation séparée. Profitant de ce fait, dans le module d'entrées-sorties nous ne gérons que des fichiers de tableaux d'entiers. Ceci résoud le problème des types mais il reste la longueur des enregistrements. Considérés comme des tableaux d'entiers, les enregistrements sont de longueurs différentes. Deux solutions sont possibles :

- Ecrire l'enregistrement entier par entier à l'aide d'une boucle. Dans ce cas l'utilisateur passe à la primitive d'écriture: un pointeur sur un buffer et le nombre d'entiers à écrire.

- Choisir une longueur d'enregistrement physique différente de celle des enregistrements logiques. Cette longueur doit-être suffisamment grande pour contenir le plus grand enregistrement logique. Les autres enregistrements plus courts sont bloqués.

La première solution obligeait à appeler les primitives d'entrées-sorties PASCAL pour chaque entier ce qui entrainait un coût non négligeable. C'est donc la deuxième solution qui a été retenue.

Comme nous avons pu le voir dans la description des sous-systèmes, beaucoup de fichiers peuvent-être regroupés ensemble pour former une entité logique. Par exemple :

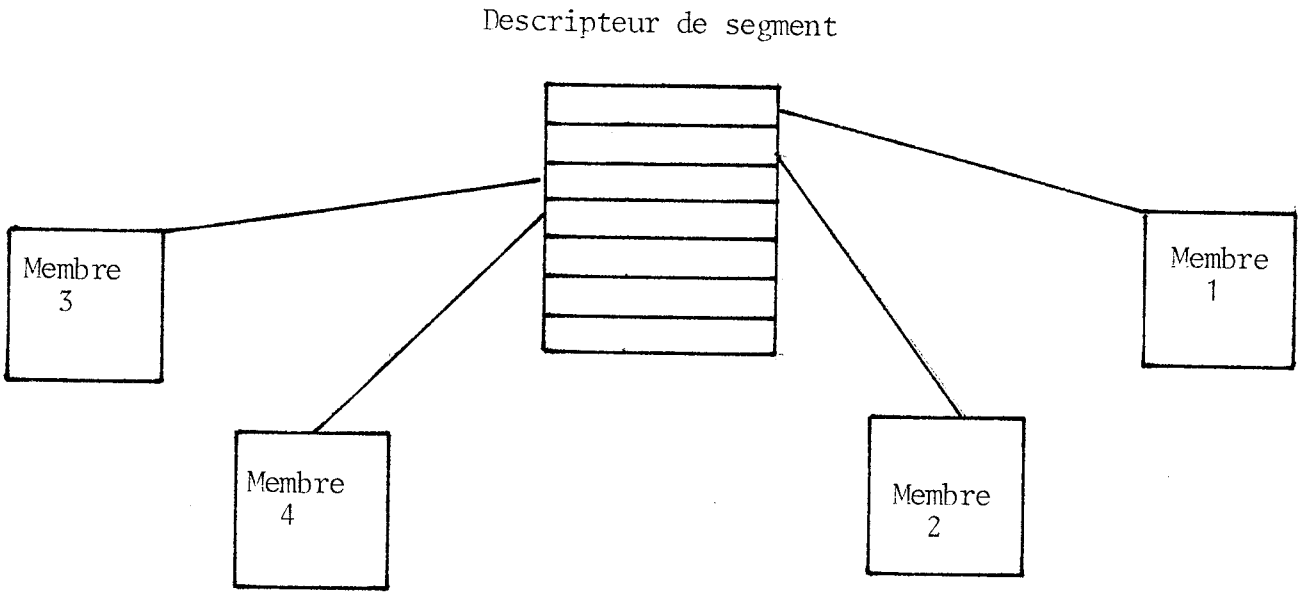
- les modèles, composés de : table de symboles, table des équations, segment transposé

- les commandes, composées de : catalogue, dicco, table des symboles, options.

- les séries chronologiques, composées de : catalogues, valeurs et nomenclatures.

Un utilisateur qui veut par exemple simuler un modèle va se servir de tous les segments correspondants. Ceci nous a conduit à introduire la notion de membre. Un segment peut lui-même être découpé en sept membres au maximum. Ce découpage permet d'écrire des enregistrements de types différents dans un segment, chaque membre pouvant en contenir un. Le segment devient donc un catalogue qui permet de regrouper plusieurs membres ayant un même centre d'intérêt.

Les primitives d'ouverture et de fermeture sont donc placées au niveau du segment ce qui permet de minimiser le nombre de ces opérations, qûi coûtent toûjoûrs assez cher.



En général dans les fichiers en accès direct, il y a beaucoup d'espace disque perdu, car le matériel permet un adressage par secteurs. Ceux-ci sont bien souvent de longueur fixe. Il est très rare que les enregistrements écrits par l'utilisateur soient de même longueur que les secteurs. Plus la longueur des enregistrements est petite devant celle d'un secteur, plus la place perdue est importante. Aussi nous avons décidé, pour une machine hôte particulière, de prendre comme longueur d'enregistrement physique la longueur d'un secteur ou un multiple de celle-ci. Afin de faire la distinction entre les enregistrements logiques et physiques ceux-ci seront appelés cases dans la suite du mémoire.

La longueur de l'enregistrement physique est paramétré à

l'aide d'une constante PASCAL. Ceci permet lorsque l'on change de site, de pouvoir ajuster la longueur aux nouveaux périphériques par une simple recompilation.

2.2.3. Mode d'accès

Au vu des besoins des sous-systèmes décrits en 2.1, nous voyons qu'il est nécessaire de réaliser plusieurs modes d'accès. Nous allons en décrire trois qui nous sont apparus indispensables:

- accès séquentiel
- accès direct bloqué
- accès direct chaîné

Le choix des primitives d'écriture, lecture et invalidation a été fortement inspiré des Système de Gestion de Fichiers de constructeurs <2> <7>.

2.2.3.1. Accès séquentiel

C'est l'accès classique à un fichier que l'on retrouve dans

tous les systèmes de gestion de fichier. Les enregistrements sont rangés consécutivement dans les cases.

A la création d'un membre séquentiel une seule case est réservée. Par la suite au fur et à mesure des écritures, le membre sera allongé, l'utilisateur est ainsi libéré de tous soucis de taille du membre.

2.2.3.1.1. Primitives accès séquentiel

Deux primitives sont proposées pour l'utilisation de cet accès, lecture et écriture :

- Ecriture (ECRSQ), permet d'écrire un enregistrement dans un membre en accès séquentiel. L'utilisateur doit fournir :

. le code délivré pour le segment par la primitive OUVRE

. le nom du membre

. un pointeur sur le buffer contenant les données

. un booléen de contrôle

Dans le cas d'utilisation normale, ce booléen doit toujours être à faux. Mis à vrai il sert à recommencer au début du membre. Il joue pour ainsi dire le même rôle que le rebobinage pour une bande magnétique.

- Lecture (LIRSQ), permet de lire un enregistrement dans un membre en accès séquentiel. L'utilisateur doit fournir :

  . le code délivré pour le segment par la primitive OUVRE

  . le nom du membre

  . un pointeur sur un buffer vide

  . un booléen de contrôle, qui a le même rôle que celui décrit pour l'écriture

Un membre ouvert en écriture peut être écrit ou lu indifféremment, le contraire n'est pas vrai.

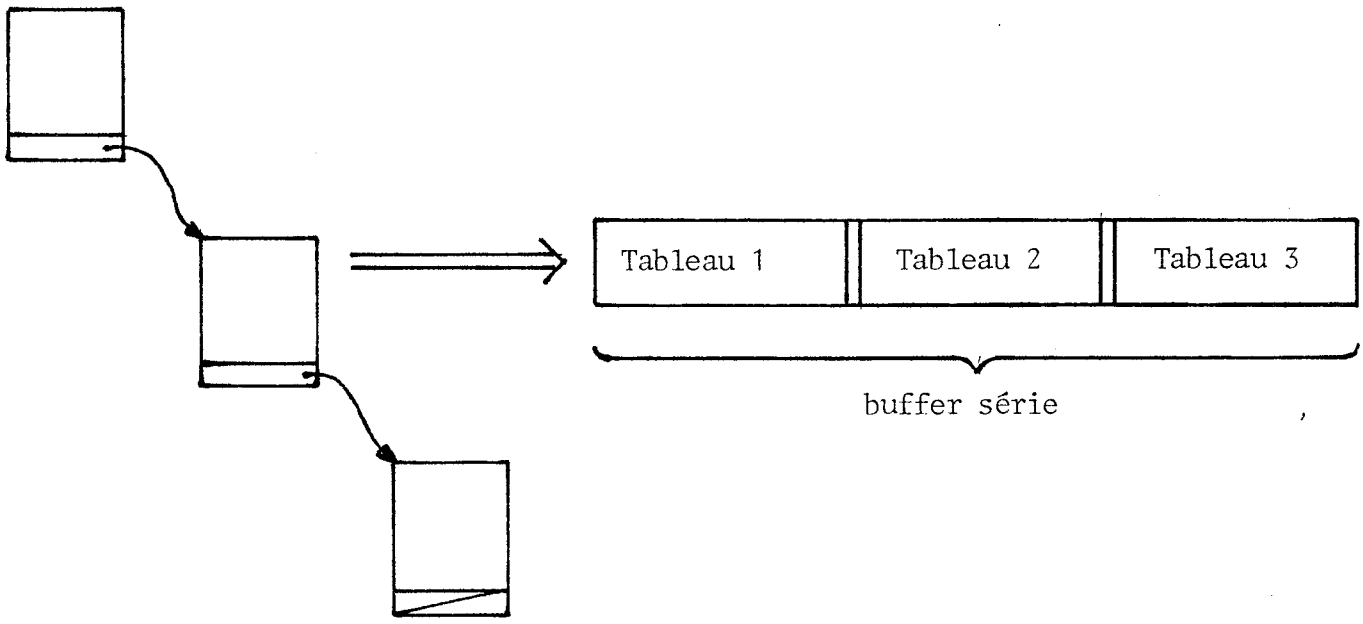
2.2.3.2. Accès direct bloqué

Cet accès est un accès direct, mais les enregistrements logiques sont regroupés à l'intérieur d'une case. Comme pour l'accès séquentiel le membre est allongé au fur et à mesure des besoins et n'est limité que par la taille du fichier. Cet accès a surtout été étudié pour résoudre le problème de stockage des séries chronologiques. Deux nécessités s'imposaient : pouvoir pointer le début d'une série chronologique et la lire entièrement, pouvoir gérer des longueurs de série variables tout en ne disposant pas des tableaux dynamiques en langage PASCAL.

Pour résoudre ces problèmes, les enregistrements logiques sont chaînés entre eux. A la différence des autres systèmes d'entrées-sorties, dans le cas de création d'une nouvelle clé, ce n'est pas l'utilisateur qui fixe le numéro de la clé mais le module d'E/S. Lorsque l'on écrit des chaînes de buffers, il est donc possible de stocker le numéro de la première clé dans un directory. Par la suite, pour relire la chaîne, il suffit de lire la première clé et séquentiellement les suivantes.

Le fait de donner le numéro de clé au module d'E/S, indique à celui-ci qu'il se trouve confronté à une mise à jour sur une clé déjà existante.

Pour les problèmes des longueurs de séries qui ne sont connues qu'à l'exécution, ceci a été résolu en mémoire par l'emploi de plusieurs tableaux successifs chaînés et pour le stockage sur fichier par le fait que l'utilisateur peut passer les enregistrements par "morceau".



Le schéma ci-dessus montre le principe de l'écriture par morceau, à gauche trois tableaux chaînés, qui seront transmis dans le buffer du membre par trois primitives d'écriture.

Sept primitives sont à la disposition de l'utilisateur pour cet accès, trois pour une utilisation normale et quatre pour une utilisation par morceau.

2.2.3.2.1. Primitives accès direct bloqué

Dans le cas d'une utilisation normale nous trouvons l'écriture, la lecture et l'invalidation :

- Ecriture (ECRDB), permet d'écrire un enregistrement dans un membre en accès direct bloqué. L'utilisateur doit fournir :

  . le code délivré pour le segment par la primitive OUVRE

  . le nom du membre

  . un pointeur sur le buffer contenant les données

  . le numéro de la clé à écrire. Deux possibilités :

    clé nulle, c'est un nouvel article en création, le numéro de clé où il est stocké est rendu à l'utilisateur

    clé non nulle, c'est une mise à jour sur une clé déjà écrite

- Lecture (LIRDB), permet de lire un enregistrement dans un membre en accès direct bloqué. L'utilisateur doit fournir:

. le code délivré pour le segment par la primitive OUVRE

. le nom du membre

. un pointeur sur un buffer vide

. le numéro de la clé à lire. Deux possibilités :

clé non nulle, les données enregistrées à cette clé sont rendues à l'utilisateur

clé nulle, c'est l'enregistrement suivant le dernier lu dans le membre qui est rendu.

- Invalidation (INVDB), permet d'invalider le dernier article dans un membre en accès direct bloqué. L'utilisateur doit fournir :

. le code délivré pour le segment par la primitive OUVRE

. le nom du membre

L'enregistrement invalidé est remis dans la chaîne des enregistrements libres.

2.2.3.2.2. Primitives traitement par "morceau"

Dans le cas d'une utilisation par morceau nous avons l'écriture, la lecture et deux primitives d'invalidation :

- Ecriture (ECRCH), permet d'écrire un enregistrement dans un membre en accès direct bloqué. Cette primitive n'introduit pas un nouveau mode d'accès, ce n'est qu'un cas particulier de ECRDB. Elle permet de fournir l'enregistrement logique par morceau.

- Lecture (LIRCH), permet de lire un enregistrement dans un membre en accès direct bloqué. Comme exposé pour la primitive ECRCH ce n'est qu'un cas particulier de LIRDB. Elle permet de récupérer l'enregistrement logique par morceau.

- Invalidation (INVCH), permet d'invalider le dernier enregistrement lu dans un membre en accès direct bloqué. Contrairement à INVDB, l'article est seulement invalidé mais n'est pas remis dans la chaîne des enregistrements libres. Ce qui fait que cette clé ne sera pas utilisée pour une nouvelle écriture. Ceci permet d'invalider des valeurs de séries chronologiques sans mettre à jour le catalogue qui pointe sur les valeurs.

(INVCHB), permet d'invalider une chaîne de N clés, en commençant par la dernière lue. L'utilisateur doit fournir :

. le code délivré pour le segment par la primitive OUVRE

. le nom du membre

. le nombre de clés à invalider

Tous les enregistrements sont remis dans la chaîne des enregistrements libres et seront donc utilisés à la prochaine création de clé (écriture avec clé nulle).

2.2.3.3. Accès direct chaîné

Ce type d'accès <8> est prévu pour résoudre les besoins d'utilisateurs stockant leurs données à l'aide d'un calcul de hashcode. Il permet de résoudre facilement les collisions. Il peut-être employé pour stocker les séries chronologiques de la base de données ou les tables des symboles des modèles ou des commandes.

Un membre en accès direct chaîné est organisé en deux parties:

- la zone principale, qui est de dimension fixe. Son nombre de clés est déterminé par le Hashcode.

- la zone secondaire, qui est de dimension variable, elle est allongée au fur et à mesure des besoins. Cette zone sert de zone débordement pour stocker les clés en collision.

La partie principale d'un tel membre ne peut-être exploitée

qu'en accès direct, alors que la zone débordement est accédée séquentiellement après lecture d'un enregistrement de la partie principale.

2.2.3.3.1. Primitives accès direct chaîné

Quatre primitives sont disponibles pour cet accès :

- Ecriture (ECRDC), permet d'écrire un enregistrement dans un membre en accès direct chaîné. L'utilisateur doit fournir :

  . le code délivré pour le segment par la primitive OUVRE

  . le nom du membre

  . un pointeur sur le buffer contenant les données

  . le numéro de la clé à écrire

  . un booléen, mis à vrai dans le cas de création d'un nouvel article.

Dans le cas de création d'un nouvel article la clé ne doit jamais être nulle. Pour la mise à jour ce renseignement est ignoré, car c'est le dernier article lu qui est mis à jour.

- Lecture (LIRDC), permet de lire un enregistrement dans un membre en accès direct chaîné. L'utilisateur doit fournir :

. le code délivré pour le segment par la primitive OUVRE

. le nom du membre

. un pointeur sur un buffer vide

. le numéro de la clé de l'enregistrement à lire. Ce numéro doit être non nul pour la première lecture d'un enregistrement de clé donnée (lecture en zone principale), puis nul pour lire la suite de la chaîne des enregistrements de même numéro de clé.

. un booléen est rendu par la primitive. Lorsqu'il est à vrai il signale la fin d'une chaîne d'enregistrement de même clé, le buffer rendu est vide. Lorsqu'il est à faux, la chaîne d'enregistrement n'est pas finie, le buffer contient l'enregistrement courant.

- Lecture séquentielle (LIRSQDC), permet de lire séquentiellement le contenu d'un membre en accès direct chaîné. A chaque appel de la primitive un enregistrement est délivré jusqu'à la fin du membre. Le membre est parcouru clé par clé, à l'intérieur de chaque clé, la chaîne d'enregistrements est parcourue entièrement avant de passer à la clé suivante. L'utilisateur doit fournir :

. le code délivré pour le segment par la primitive OUVRE

. le nom du membre

. un pointeur sur un buffer vide

. un booléen de contrôle

Ce booléen joue le même rôle que celui de la lecture séquentielle (LIRSQ). C'est à dire qu'il n'est pas nécessaire de fermer et de rouvrir le membre pour faire commencer la lecture au début. Il suffit de mettre ce booléen à vrai, ce qui a pour effet de forcer la lecture de la clé 1.

- Invalidation (INVDC), permet d'invalider le dernier article lu dans un membre en accès direct chaîné. L'utilisateur doit fournir:

. le code délivré pour le segment par la primitive OUVRE

. le nom du membre

L'enregistrement invalidé est remis dans la chaîne des enregistrements libres.

2.2.4 Prolongation de fichier

Comme nous avons pu le voir les membres en accès séquentiel, en accès direct bloqué et la zone secondaire des membres en accès direct chaîné sont allongés jusqu'à saturation du fichier utilisateur.

Dans ce cas il est prévu une primitive qui permet de recopier

le fichier dans un fichier plus grand. Après cette recopie l'utilisateur peut continuer son travail normalement comme s'il avait toujours eu un fichier de la nouvelle taille.

3. AVANTAGES DE LA SOLUTION PAR RAPPORT A LA PORTABILITE

Comme exposé dans <1> il y a trois sphères d'influence sur la portabilité d'un programme : la machine, le système d'exploitation et le langage de programmation.

Dans le cas du module de gestion de fichiers, nous sommes dépendant du système et bien entendu du langage de programmation <6>.

3.1. Dépendance système

Pour le module d'entrées-sorties la dépendance système provient du fait qu'aucune méthode d'accès n'est incluse dans la norme ISO de PASCAL. Selon les différents compilateurs plusieurs cas peuvent se présenter :

- l'accès direct existe et les primitives d'utilisation sont identiques dans les deux compilateurs origine et cible.
- l'accès direct existe, les primitives d'utilisation sont différentes
- l'accès direct n'existe pas.

Afin de minimiser les effets de ces différences de compilateurs, le module d'E/S est coupé en deux parties :

- une qui gère tous les objets : descripteurs de segments, membres, buffers, mais ne lance aucune entrée-sortie PASCAL. Cette partie est donc entièrement portable du point de vue système.

- l'autre qui ne comporte que les primitives d'appel des entrées-sorties PASCAL. Le fait d'avoir mis ces primitives dans un module séparé, permet à l'équipe chargé du transport de n'avoir à modifier, ou à réécrire dans un autre langage, qu'un module très petit.

Cette décomposition apporte un avantage certain, car il est toujours plus facile de comprendre et modifier des petits programmes que des gros.

3.2. Dépendance langage de programmation

Le fait de se limiter au PASCAL norme ISO a beaucoup limité les différences entre les compilateurs. Les seuls problèmes rencontrés sont liés au fait de faire des compilations séparées. Dans ce cas nous trouvons les différences :

- d'ordre lexical pour déclarer les variables et les procédures externes

- de gestion des variables globales des modules

Pour minimiser le travail de transport les déclarations de procédures externes (DEF) ont été regroupées en déclaration FORWARD en-tête du module. De même les déclarations de procédures référencées sont regroupées.

Toutes les variables globales du module sont mises dans un seul RECORD PASCAL et placées dans un module à part. Dans le cas de différence de traitement dans les divers compilateurs, il n'y a qu'un RECORD par module à vérifier pour le transport.

Toutes ces normes internes sont prises en compte dans tous les modules de MODULECO. Ceci a permis le transport des 70000 lignes de code PASCAL de Multics en PASCAL-VS d'IBM en une période de l'ordre de trois mois. Des procédures semi-automatiques de transformation de code ont pu être réalisées avec les éditeurs de textes classiques.

4. AMELIORATION DU LOGICIEL EN VUE DE SON AUTONOMIE

Le système de gestion de fichiers présenté dans ce mémoire est intégré à MODULECO. De ce fait certaines opérations lui échappent complètement. Il suffit de quelques modifications afin de le rendre autonome et utilisable par d'autres projets.

Nous décrivons ci-dessous, les quelques modifications apportées au module :

- Suppression de toutes les procédures particulières à MODULECO, tel que INITSYS, DEBSYS,etc...

- Simplification de la notion de segment. Les membres disparaissent car ils sont bien particuliers à MODULECO et ne représentent rien pour un système de gestion d'entrées-sorties. De ce fait nous avons maintenant un fichier utilisateur qui est partitioné en segment.

- Possibilité de gestion de quatre fichiers (à l'aide des CUTS), qui n'ont aucune prérogative l'un sur l'autre. Pour le module d'entrées-sorties, ces quatre fichiers peuvent se gérer indépendamment l'un de l'autre.

- Gestion d'un catalogue de segments. Dans MODULECO, cette gestion était prise en charge par le gestionnaire d'objets.

- Modification de l'accès direct bloqué. Cet accès n'était pas un vrai accès direct, puisque l'utilisateur ne pouvait pas décider

du numéro de clé d'écriture. Seules les lectures ou les modifications étaient en accès direct.

- Suppression de la notion d'enregistrements temporaires qui étaient très liés au système MODULECO.

Nous allons examiner la gestion du catalogue de segments et l'accès direct bloqué qui sont les seules modifications représentant un travail important.

4.1. Catalogue de segments

Dans le module d'entrées-sorties en service sur MODULECO, la gestion des descripteurs de segments est sous la responsabilité du gestionnaire d'objets. Du fait de la structuration de l'espace d'adressage avec les annuaires, un nom segment complet peut comporter plusieurs noms d'annuaires avant le nom de segment lui-même (Ex: Annu1.Annu2.Seg1). Le module d'entrées-sorties ne connait qu'un pointeur sur un descripteur de segment qui ne contient que le dernier nom (Seg1), beaucoup de vérifications ne sont donc pas possibles.

Le module d'entrées-sorties autonome gère un catalogue des segments. Ce catalogue est exploité à l'aide de l'accès direct chaîné qui se prête bien à ce genre de stockage. Le catalogue est créé lors du formattage du fichier utilisateur. Il est ouvert par

la primitive INITES qui s'occupe de la connection dynamique et de l'ouverture du fichier utilisateur. Il est ainsi accessible pendant toute la session d'utilisation du fichier.

4.2. Création du segment

Lors de la création d'un segment, son descripteur est rangé dans le catalogue selon un calcul de hashcode approprié. Lors de l'appel de la primitive CRSEG l'utilisateur doit communiquer :

- le nom du segment (16 caractères maxi)
- la longueur des enregistrements logiques
- le mode d'accès

Ce sont ces renseignements ainsi que le numéro de l'enregistrement physique de début du segment qui sont stockés dans le descripteur de segment.

Par la suite toutes les primitives de gestion de segments (DETSEG, OUVRE, COPIE) se font uniquement à l'aide du nom de segment, toutes les autres primitives étant appelées avec le code rendu par la primitive OUVRE.

4.3. Accès direct bloqué

L'accès direct bloqué n'était pas réellememt un accès direct. Tous les enregistrements étaient chaînés entre eux. C'est le module d'entrées-sorties qui décidait du numéro de la clé choisie en création et ensuite le délivrait à l'utilisateur. Seule les lectures et les écritures en mise à jour se faisaient réellement en accès direct. Nous avons rajouté en plus des possibilités actuelles, l'écriture en accès direct de clé en création. Par rapport à l'utilisation antérieure, une restriction a été apportée, lors de création de clé en accès direct. Celle-ci doit exister physiquement, il n'y a pas d'allongement de segment automatique comme pour l'autre cas d'utilisation.

CONCLUSION

Nous avons présenté dans ce mémoire un système de gestion de fichiers écrit en langage PASCAL pour le logiciel MODULECO.

Ce système a permis de simplifier le travail de l'équipe de réalisation et de celle chargée du transport du logiciel sur d'autres ordinateurs.

Nous avons amélioré ce système afin de le rendre autonome et pensons qu'il pourra être très utile à des projets Informatique utilisant le langage PASCAL comme support de programmation.

ANNEXE

ANNEXE 1 PLAN

1. PRESENTATION DES ENTREES-SORTIES.

Le langage PASCAL utilisé pour la réalisation de MODULECO est un langage typé. Un problème est posé: l'écriture des fichiers. Dans le cas d'une utilisation normale du langage PASCAL, il faut des déclarations pour chaque fichier de type différent et par suite des procédures pour l'écriture et la lecture de chacun de ces types de fichier.

Un autre problème posé pour la réalisation de MODULECO est la multiplication des fichiers qui peuvent-être très nombreux. (Fichiers de séries chronologiques de la base de données, modèles, différentes variantes, etc..), ceci peut-être gênant dans certains systèmes d'exploitation.

Dans certains compilateurs PASCAL les Entrées-Sorties en accès direct peuvent ne pas être implémentées, car non prévues dans la norme. Il est donc intéressant de regrouper dans un même module toutes les instructions qui peuvent ne pas être portables et de ce fait susceptibles d'être modifiées.

Le module d'Entrées-Sorties pallie ces différents inconvénients:

- un seul fichier par utilisateur, où sont regroupées toutes ces données;
- le module d'Entrées-Sorties ne gère que des fichiers d'entiers, profitant du fait qu'il n'y a pas de contrôle lors de compilations séparées.

2. OBJETS GERES ET MODE D'ACCES.

Le module d'Entrées-Sorties gère des segments qui sont subdivisés en membres. Un segment est un ensemble de membres où sont regroupées toutes les données ayant un même intérêt. (Par exemple : ensemble des données décrivant un modèle).

Chaque segment est divisé en membres. Ceci permet simplement de pouvoir gérer des enregistrements logiques de longueurs différentes ou des modes d'accès différents.

Trois modes d'accès sont possibles :

- accès séquentiel
- accès direct bloqué
- accès direct chaîné.

2.1. Organisation du fichier utilisateur

Le fichier utilisateur est constitué d'enregistrements chaînés entre eux. Un nouvel utilisateur de MODULECO doit se faire allouer un espace disque (par l'administrateur du site) en même temps qu'il demande une entrée dans le système MODULECO.

Pour créer cet utilisateur, l'administrateur doit lui affecter un fichier de la taille demandée. Lors de l'entrée de cet utilisateur dans MODULECO, la primitive du gestionnaire d'objets CRUTIL, doit appeler une primitive, du module d'Entrées-Sorties qui formatte le fichier selon la longueur d'enregistrement, prévue sur cette machine hôte. Ces enregistrements appelés case dans la suite de cette brochure ont une longueur qui dépend de l'unité physique adressable. Lors du formattage tous les enregistrements sont chaînés, afin de les mettre dans une chaîne d'enregistrements libres. Le nom du fichier est rangé dans le descripteur utilisateur. Chaque case est en fait constituée d'une partie utile de longueur variable en fonction de la machine hôte et d'un entier qui sert au chaînage des enregistrements.

Le fichier est ainsi prêt à être utilisé. Le premier enregistrement est réservé au module E/S afin de stocker les informations nécessaires à la gestion du fichier :

- pointeur sur la première case libre;
- nombre des cases du fichier;
- deux entiers réservés au fichier système, ils servent à régénérer le descripteur de segments des deux membres constituant le fichier système. Ils donnent le début de chacun des deux membres constituant ce fichier;
- flag de validité du fichier
- nombre de cases occupées

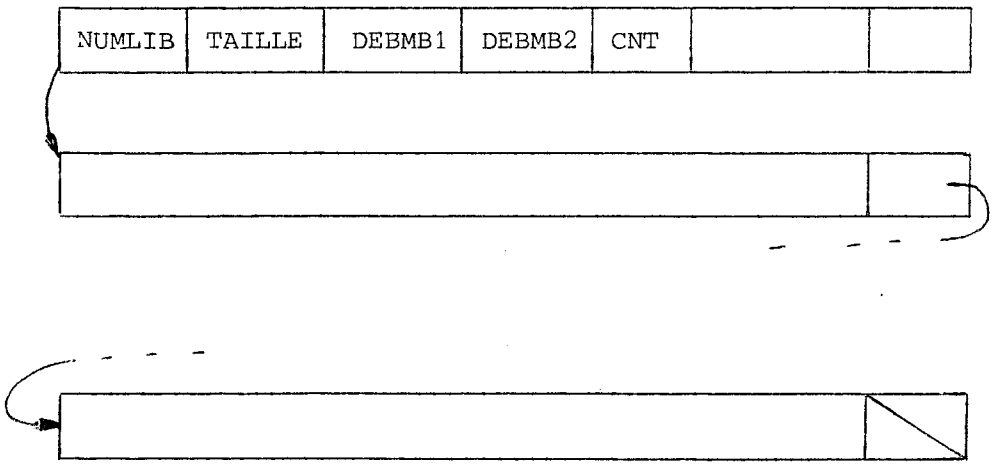


Schéma d'organisation d'un fichier utilisateur après formattage

NUMLIB : pointeur sur le début de la chaîne des libres.

TAILLE : nombre de cases réservées pour le fichier

DEBMB1 : pointeur sur le début du premier membre du fichier système

DEBMB2 : pointeur sur le début du deuxième membre du fichier système.

FLAG : flag de validité du fichier

0. Le fichier est cohérent

1. Le fichier est détruit

NBOCCU : nombre de cases occupées dans le fichier.

2.2. Gestion des segments

Pour se faire allouer un segment, l'utilisateur doit appeler la primitive CRSEG (CRéer SEGment) du gestionnaire d'objets. C'est celui-ci qui appelle la primitive CRMB (CRéer MemBre) du module E-S.

A l'appel de la primitive CRMB, il faut communiquer :

- le nom du membre
- la longueur des enregistrements logiques. Dans la version actuelle la longueur d'un enregistrement logique ne peut pas dépasser la longueur d'une case.
- l'utilisation que l'on veut faire du membre :

  ASEQ : accès séquentiel

  ADBL : accès direct bloqué

  ADCH : accès direct chaîné

- dans les cas ADBL et ADCH, il faut en plus indiquer le nombre prévisible d'enregistrements. En ADBL, le membre s'allonge au fur et à mesure des besoins. Il suffit donc de mettre un nombre d'enregistrements minimum pour réserver une case au départ. En ADCH, il faut donner le nombre de numéros d'enregistrements différents que pourra par exemple délivrer le Hashcode. Seule la zone secondaire qui sert à gérer les collisions est allongée au fur et à mesure des besoins.
- un pointeur sur un descripteur de segments, où le module E-S va ranger les renseignements nécessaires à la gestion du membre

  pointeur sur le début du membre

mode d'accès

longueur d'enregistrement

nombre d'enregistrements maximum dans le membre

Fonctionnement de la primitive CRMB

Dans le cas d'un accès séquentiel, le module E/S réserve une case du fichier utilisateur et stocke son numéro dans le descripteur de segment (mode d'accès = 1).

Dans le cas d'un accès direct bloqué, le module E/S réserve N/Facteur de blocage cases du fichier utilisateur plus une pour la table d'index. Le numéro de la case table index est stocké dans le descripteur de segment. (mode d'accès = 2)

Dans le cas d'un accès direct chaîné, le module E/S réserve N/Facteur de blocage cases du fichier utilisateur plus une pour la table d'index, puis deux cases pour la zone débordement, une pour la table index et une pour les enregistrements. Le numéro de la case index zone débordement est enregistré dans la première case du membre. Le numéro de la case table index de la zone principale est stoké dans le descripteur de segment (mode d'accès = 3).

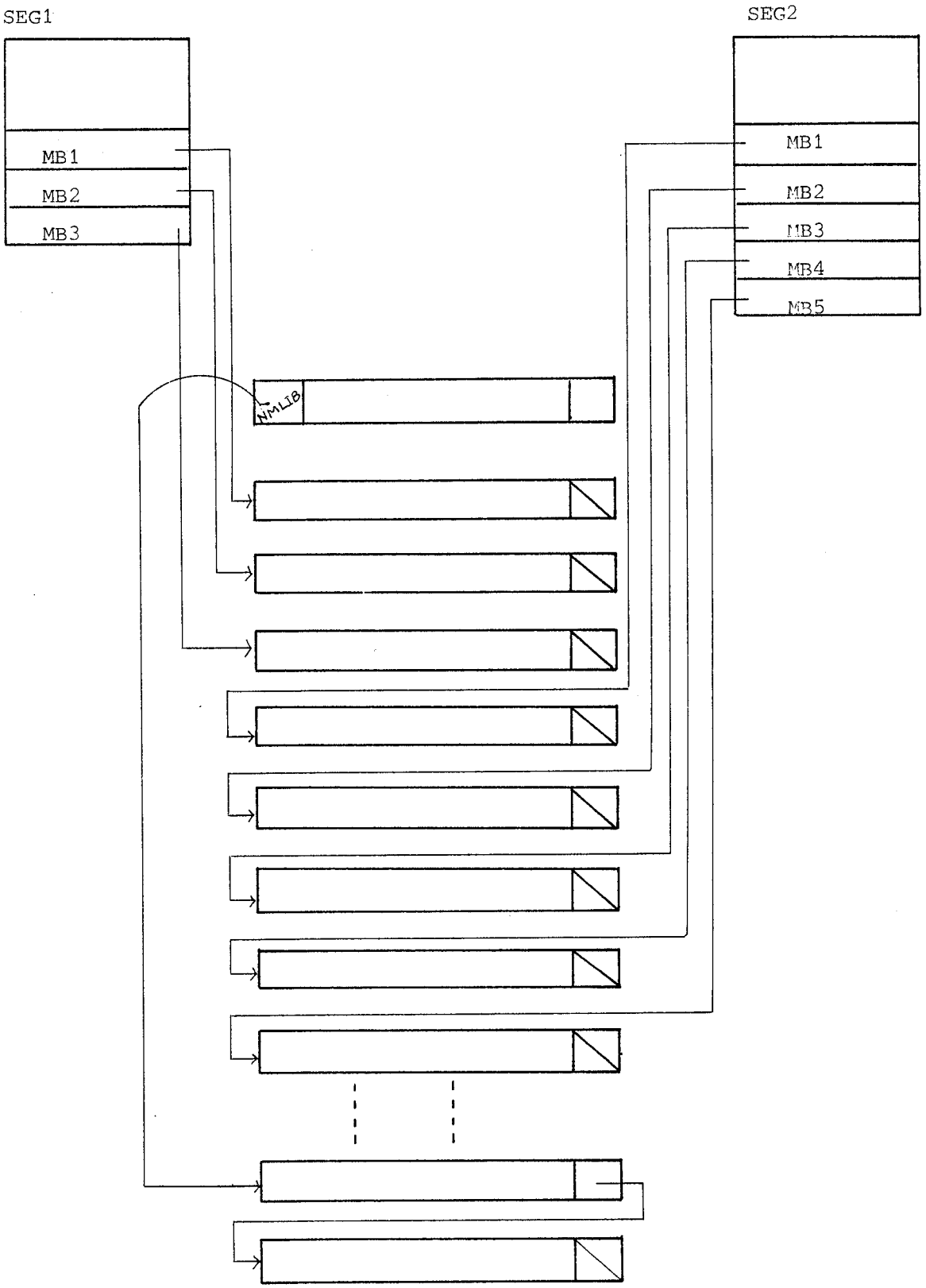
Le numéro de la case du fichier qui représente le début du membre n'est pas communiqué à l'utilisateur. Le fait que le membre soit un morceau d'un fichier n'est pas son problème, pour lui tout se passe comme s'il avait un fichier qui a le nom de son membre.

Fonctionnement de la primitive DETSEG

A l'aide de la primitive DETSEG, l'utilisateur peut détruire un segment devenu inutile. Dans ce cas, tous les membres du segment sont détruits. Toutes les cases encore occupées sont remises dans la chaîne des libres. Les noms de chaque membre sont remis à blanc dans le descripteur de segment.

Fonctionnement de la primitive DETMBR

A l'aide de la primitive DETMBR, l'utilisateur peut détruire un membre d'un segment. Toutes les cases occupées par ce membre sont remises dans la chaîne des libres. Le nom du membre est remis à blanc dans le descripteur de segment.


SEG1
MB1
MB2
MB3
SEG2
MB1
MB2
MB3
MB4
MB5

2.3. Structure des enregistrements et des cases

2.3.1 Accès Séquentiel

Dans le cas d'accès séquentiel, il n'y a aucune difficulté. Les enregistrements sont rangés consécutivement dans les cases. Un simple calcul tenant compte de la longueur d'une case et de la longueur d'un enregistrement logique permet de retrouver l'emplacement de l'enregistrement concerné. Les cases d'un membre en accès séquentiel sont chaînées entre elles, à l'aide d'un pointeur, le pointeur de la dernière case est mis à zéro.

2.3.2. Accès Direct Bloqué

Dans le cas d'accès direct bloqué, les enregistrements logiques sont chaînés entre eux afin de permettre les invalidations. On retrouve deux entiers entre chaque enregistrement qui permettent de faire :

- un chaînage avant
- un chaînage arrière.

Les cases d'un membre sont chaînées entre elles, à l'aide d'un pointeur, celui de la dernière case est mis à zéro. Au début d'un membre en accès direct, une case est réservée afin d'y stocker une table d'index permettant un accès plus rapide aux clés en cours de traitement. Devant la table d'index se trouve 7 mots qui servent à la gestion du membre. Pour un membre en accès direct bloqué, 5 seulement sont utilisés :

- le troisième, qui contient un entier pointant sur le début de la chaîne des enregistrements libres. Pour un membre vide, il est égal à 1.
- le quatrième, qui contient un entier pointant sur le début de la chaîne des enregistrements occupés. Pour un membre vide, il est égal à -1.
- le cinquième, qui contient le numéro de la dernière case réservée
- le sixième, qui contient le numéro de la dernière clé utilisée Pour un membre vide, il n'est pas initialisé.

- le septième, qui contient le numéro de la dernière clé réservée

2.3.3. Accès Direct CHainé

Dans le cas d'accès direct chaîné, un membre est constitué de deux parties:

- une partie principale qui est identique dans sa forme et son utilisation au membre en accès diect bloqué. Dans cette partie principale il est réservé le nombre d'enregistrements prévus par l'utilisateur plus un. Pour un membre en accès direct chaîné les sept mots devant la table d'index sont tous utilisés, on y trouve :

  le premier, numéro de la première case de la zone débordement

  le deuxième, plus grand numéro d'enregistrement logique possible dans la zone débordement actuelle (soit avant sa prochaine prolongation).

  le troisième contient un entier pointant sur la chaîne des enregistrements libres de la zone débordement. Tant que la zone débordement est vide, ce mot contient le nombre d'enregistements déclarés au CRMB plus deux. En l'état actuel de l'implémentation, il est impossible de savoir si la partie principale contient des données.

  le quatrième contient un entier pointant sur le début de la chaîne des enregistrements occupés. Au début il est initialisé à -1.

  le cinquième, numéro de la dernière case de la zone débordement en cours de traitement.

  le sixième, contient le numéro de la dernière clé utilisée . Pour un membre vide il n'est pas initialisé.

  le septième, qui contient le numéro de la dernière clé réservée pour la zone principale.

- une partie zone débordement qui au départ comprend deux cases, la première pour y mettre une table index, la seconde pour y ranger le début des enregistrements en

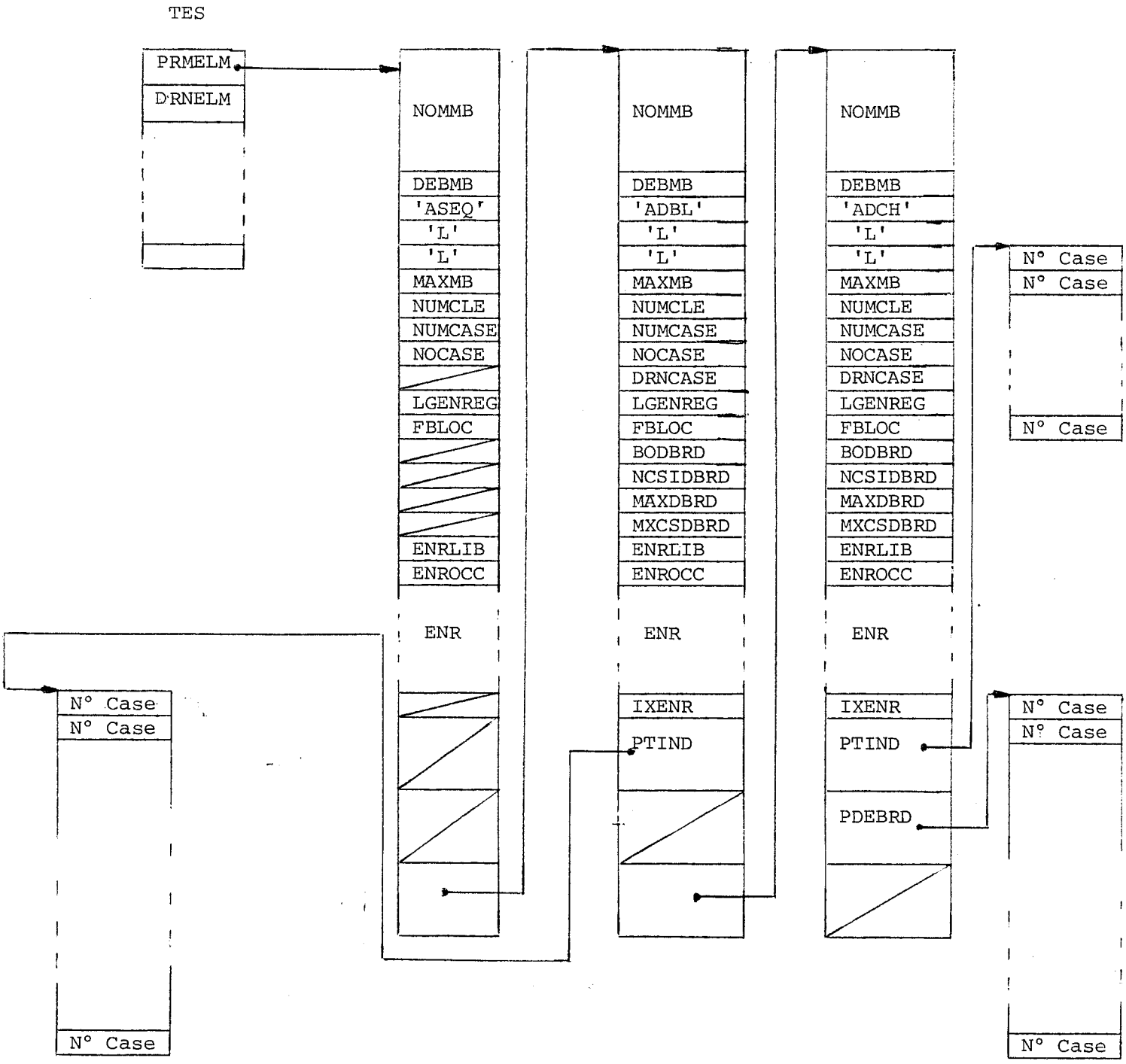
débordement. Par la suite cette zone sera allongée au fur et à mesure des besoins comme dans le cas de l'accès séquentiel ou direct bloqué.

Les cases d'un tel membre sont chaînées entre elles à l'aide du pointeur, en trois chaînes :

- la première pour la zone principale, la dernière case ayant un chaînage nul

- la deuxième pour la zone débordement, la dernière case ayant un chaînage nul.

- la troisième pour la table index de la zone débordement, mais le chaînage se trouve dans l'avant-dernier mot de la case. De la même manière que les autres chaînes la dernière case a un chaînage nul. Seule la pemière case de la table index est inclue dans la chaîne normale de la zone débordement, car elle est la première case de cette chaîne.



Nous trouvons dans ce schéma trois membres :

- un en accès séquentiel
- un en accès direct bloqué
- un en accès direct chaîné

représentés immédiatement après l'ouverture du segment

2.4. Structures table d'index.

La table d'index est une suite de pointeurs sur les différentes cases du membre concerné. Chaque clé de la table index représente donc le premier numéro d'enregistrement logique qui se trouve en-tête de la case pointée.

La première clé représente l'enregistrement numéro un.

La deuxième clé l'enregistrement de numéro un + facteur de blocage, etc...

NOTA En annexe est inclus un exemple simple d'un fichier utilisateur.

3. MODE D'ACCES.

Trois modes d'accès sont possibles avec le module d'E-S :

- accès séquentiel
- accès direct bloqué
- accès direct chaîné

3.1. Accès séquentiel

Comme il a été vu pour le CRMB, une seule case du fichier utilisateur a été réservée. Au fur et à mesure des écritures, le membre sera allongé lorsque ce sera nécessaire. Cet accès fonctionne comme un accès séquentiel classique, mais profitant du fait qu'il est implémenté par dessus un accès sur disque, un booléen permet de reprendre le membre au début. Un membre ouvert en écriture peut-être écrit ou lu indifféremment. Après avoir lu ou écrit le membre, il n'est pas nécessaire de le fermer et de le rouvrir pour changer d'ordre, (lecture ou écriture), il suffit de mettre le booléen à vrai pour recommencer au début. Le booléen à vrai joue en fait le même rôle que le rebobinage pour une bande magnétique.

3.2. Accès direct bloqué

Cet accès est un accès direct, mais les enregistrements logiques sont regroupés à l'intérieur des cases. A la différence des autres systèmes d'E/S, dans le cas de création d'une nouvelle clé, ce n'est pas l'utilisateur qui fixe le numéro de la clé mais le module d'E/S. Lorsque l'on écrit des chaînes de buffer, il est donc possible de stocker le numéro de la première clé dans un directory. Par la suite, pour relire la chaîne, il suffit de lire la première clé et séquentiellement les suivantes.

Dans le cas où l'utilisateur donne le numéro de clé en écriture, le module d'E/S, considère que c'est une mise à jour sur une clé déjà existante. Un membre ouvert en lecture peut seulement se lire, alors qu'un membre ouvert en écriture peut-être écrit ou lu indifféremment.

Les membres ne sont pas limités en nombre d'enregistrements, c'est seulement la taille du fichier utilisateur qui les limitera quand il n'y aura plus de cases disponibles.

Pour l'accès direct bloqué, il est possible de donner les buffers par morceaux. C'est seulement lorsque le buffer est plein qu'il est écrit dans le fichier. Attention c'est seulement une facilité pour découper un buffer en morceaux, mais il n'est pas question de donner des morceaux de buffer et que le module d'E/S puisse mettre les morceaux à cheval sur deux enregistrements logiques.

3.3. Accès direct chaîné

Ce type d'accès est prévu pour résoudre les besoins d'utilisateurs stockant leurs données à l'aide d'un calcul de hashcode. Il permet de résoudre facilement les collisions. Il peut-être employé pour stocker les séries chronologiques de la base de données ou même les tables des symboles des modèles ou des commandes.

Les membres en accès direct chaîné sont organisés en deux parties :

- la première, appelée zone principale, est de dimension fixe. Son nombre de cases est déterminé par le hashcode;
- la deuxième, appelée zone débordement, est de dimension variable, elle est allongée au fur et à mesure des besoins, pour stocker les enregistrements résultant de collisions.

La partie principale d'un tel membre ne peut-être exploitée qu'en accès direct, alors que la zone débordement est accédée séquentiellement après lecture d'un enregistrement de la partie principale.

Un membre ouvert en lecture peut seulement se lire, alors qu'un membre ouvert en écriture peut-être écrit ou lu indifféremment.

3.4. Prolongation des fichiers utilisateurs

Dans le cas de fichier utilisateur plein, qui provient lors de la réservation d'une nouvelle case provoquée par une écriture dans le cas de l'allongement du membre ou par CRMB, le système MODULECO est automatiquement arrêté en signalant l'erreur.

Grâce à l'utilisation des enregistrements temporaires, il n'y a aucun problème de reprise. Il suffit que l'utilisateur demande à l'administrateur du Site d'agrandir son fichier. Dans le cas de travail en sauvegarde automatique, seule la dernière commande est perdue, une fois le fichier agrandi, il suffira de réexecuter celle-ci. Dans le cas de travail en sauvegarde non automatique, tout le travail exécuté depuis le dernier appel de la commande SAUVE est perdu, s'il n'y en a pas eu la session complète est perdue.

L'administrateur du site doit recopier le fichier plein sous un autre nom car il est impératif que le nouveau fichier ait le même nom que l'ancien, ensuite il crée un fichier plus grand que l'ancien et enfin appelle la commande COPYFIC. Cette commande demande le nom et la taille du nouveau fichier, puis le nom de l'ancien. Après formattage du nouveau fichier, le contenu de l'ancien est recopié. Tous les indicateurs contenus dans le premier enregistrement sont mis à jour, afin de pouvoir continuer le travail normalement, comme s'il avait toujours eu un fichier de la nouvelle taille.

4. ORGANISATION DU MODULE D'ENTREES-SORTIES

Le module d'Entrées-Sorties comprend 31 primitives et 2 commandes

- Primitives de Trace
  DEBPROC, FINPROC
- Primitives d'initialisation
  INITES, INITMODES, INITSYS, DEBSYS, INIREPRI
- Primitive de formattage de fichier
  IFIC
- Primitives de gestion de fichier
  TAUXOCC, COPYFIC, FNBCASE, FMAXMB, ETATFICH
- Primitives de gestion des segments
  CRMB, DETSEG, OUVRE, FERME, DETMBR, COPIE, TSTFERME
- Primitives d'écriture
  ECRSQ, ECRDB, ECRCH, ECRDC
- Primitives de lecture
  LIRSQ, LIRDB, LIRCH, LIRDC, LIRSQDC
- Primitives d'invalidation
  INVDB, INVDC, INVCH, INVCHB

Le contrôle des Entrées-Sorties est effectué à l'aide du descripteur de segment pour les créations de membres, destructions de membres, et à l'aide de la table de gestion des Entrées-Sorties pour les écritures, lectures et invalidations.

Les Entrées-Sorties sont écrites sous forme de deux modules MODES et TRTES. Le premier est entièrement portable. Le second qui peut-être écrit en PASCAL devra-être refait entièrement dans le cas où le compilateur ne supporte pas les Entrées-Sorties en accès direct, ou être modifié légèrement pour les cas particuliers de chaque machine hôte, principalement les FCONNECT.

Format PASCAL des diférents types utilisés dans le chapitre 4

```
CONST
   LGENRPHY=1022;    (* LONGUEUR ENREGISTREMENT PHYSIQUE *)
   LGFIC=60;    (* LONGUEUR NOM DE FICHIER UTILISATEUR *)
   LGID=16;    (* LONGUEUR NOM D'IDENTIFICATEUR *)
   LONGBANA=10;    (* LONGUEUR ZONE BANA D'UN SEGMENT *)

TYPE
   DEMIMOT=0..65535;
   TYPDESCR=(SEGR);
   NMSEG=PACKED ARRAY(.1..LGID.) OF CHAR;
   NMMBR=PACKED ARRAY(.1..LGID.) OF CHAR;
   PBUFF= CHMP;
   CHMP=ARRAY(.1..LGENRPHY.)OF INTEGER;    (* PART UTI DE L'ENR *)
   NOMFIC=PACKED ARRAY(.1..LGFIC.)OF CHAR;    (* NOM DE FICH *)
   PFICH=FICH;
   FICH=PACKED RECORD    é DECLARATION ENREGISTREMENT DU FICHIER è
      CHAMP:CHMP;          é UTILISATEUR è
      REPERE:DEMIMOT;
      PTR:DEMIMOT;
      NUM:INTEGER;
   END;
   PDESCRSG= DESCRSEG;
   DESCRSEG=PACKED RECORD
      CHAINAGE:INTEGER;
      MARQUE:TYPDESCR;
      NOMSEG:NMSEG;
      PERE:INTEGER;      é ANNUAIRE PERE è
      LISTACC:INTEGER;    é LISTE D'ACCES è
      DOC:INTEGER;
      COMPT:DEMIMOT;    é COMPTEUR DE LIAISONS è
      NBMEMB:DEMIMOT;    é NOMBRE DE MEMBRES è
      DCR:ALFA;    é DATE DE CREATION è
      DMOD:ALFA;    é DATE DERNIERE MODIFICATION è
      BANA:ARRAY(.1..LONGBANA.) OF INTEGER;
      ZONMBR:ARRAY(.1..NMMBR.) OF PACKED RECORD
         NOMMB:NMMBR;
         MODE:1..4;
         LGENREG:DEMIMOT;
         DEBMB:INTEGER;    é CLE DEBUT DE MEMBRE è
      END;
   END;
```

4.1. Description des variables du module MODES

Dans le module MODES nous trouvons actuellement une variable CUT:ENTIER, qui sert à la gestion des différents fichiers:

1 pour le fichier système,

2 pour le fichier utilisateur,

3 pour le fichier d'un voisin,

4 pour le fichier commande.

Pour les primitives de gestion de segment, l'index est passé explicitement. Dans le cas des primitives d'écriture, lecture ou invalidation, l'index est inclu dans le code rendu à l'ouverture.

Nous trouvons :

a/ Une variable ENREG qui sert de buffer de travail, lors de toutes les manipulations pour les corrections de chaînage.

b/ Le tableau de gestion des Entrées-Sorties qui est la clé de tout le module, c'est dans ce tableau que sont répertoriés tous les segments ouverts au cours de la session. C'est un tableau de doubles pointeurs permettant de chaîner les différents membres de chaque segment.

Dans chaque membre nous trouvons les renseignements suivants :

NOMMB nom du membre

DEBMB numéro case début du membre

MODE mode d'utilisation (1,2 ou 3)

OUV mode d'ouverture lecture ou écriture

MACR dernière macro utilisée L, E ou I

MAXMB en ASEQ, numéro de la dernière clé écrite dans le membre en ADBL ou ADCH, numéro de la plus grande clé possible dans le membre

NUMCLE numéro de la clé en cours de traitement

NUMCASE numéro de case en cours de traitement (case fichier)

NOCASE numéro de case relatif dans le membre

DRNCLE numéro dernière clé utilisée en écriture

MAXCASE numéro dernière case du membre ADBL, dernière case zone débordement en ADCH

LGENREG longueur d'un enregistrement logique.

FBLOC facteur de blocage

BODBRD Vrai, si le buffer contient un enregistrement de la zone débordement donc NUMCASE contient une clé sur la zone de débordement

NCSIDBRD numéro case index de la zone débordement

MAXDBRD numéro plus grande clé possible dans la zone débordement avant la prochaine prolongation

MXCSDBRD plus grand numéro de case de la zone débordement écrit actuellement

ENRLIB numéro du premier enregistrement libre.

ENROCC numéro du premier enregistrement occupé

ENR buffer de lecture pour le membre

IXENR index courant dans le buffer, pour écriture par morceaux

PTIND pointeur vers la table index

PDEBRD pointeur vers la table d'index de la zone débordement.

Voir plus loin le schéma de représentation de segments de différentes formes.

c/ Tableau des numéros début de chaîne des enregistrements libres, une entrée par fichier.

d/ Tableau taille des cases de chaque fichier.

Format Pascal

```
PGES= GES;
   NOMMB:NMMBR;
   DEBMB:INTEGER;    (* NUMERO DE CASE DEBUT DU MEMBRE *)
   MODE:INTEGER;     (*MODE D'UTILISATION *)
   OUV:CHAR;    (* LECTURE OU ECRITURE *)
   MACR:CHAR;     (* DERNIEREMACRO UTILISEE *)
   MAXMB:INTEGER;    (* NUMERO PLUS GRANDE CLE REPERTORIEE *)
   NUMCLE:INTEGER;    (*NUMERO DE CLE EN COURS *)
   NUMCASE:INTEGER;     (*NUMERO DE CASE EN COURS DE TRAITEMENT *)
   NOCASE:INTEGER;    (*NUMERO DE CASE RELATIF EN COURS *)
   DRNCLE:INTEGER;    (* NUMERO DERN CLE UTILISEE EN ECRIT *)
   MAXCASE:INTEGER;    (* NUMERO DERNIERE CAE DUMEMBRE ADBL *)
   LGENREG:INTEGER;    (*LONGUEUR D'ENREGISTREMENTLOGIQUE *)
   FBLOC:INTEGER;    (*FACTEUR DE BLOCAGE DANS LA CASE *)
   BODBRD:BOOLEAN;    (* TRUE, SI ZONE DEBORD DANS LE BUFFER *)
   NCSIDBRD:INTEGER;     (*NUMERO CASE INDEX DEBORDEMENT *)
   MAXDBRD:INTEGER;    (* NUMERO PLUS GRANDE CLE EN ZONE DEBORD *)
   MXCSDBRD:INTEGER;     (* NUMERO DERN CASE ZONE DEBORDEMENT *)
   ENRLIB:INTEGER;    (* NUMERO PREMIERENREGISTREMENT LIBRE *)
   ENROCC:INTEGER;    (* NUMERO PREMIERENREGISTREMENT OCCUPE )
   ENR:PFICH;
   IXENR:INTEGER;    (* INDEX COURANT, POUR CHOUIA *)
   PTIND:PFICH;    (* TABLE INDEX ACCES DIRECT *)
   PDEBRD:PFICH;    (* TABLE INDEX ZONE DEBORDEMENT *)
   PSUIV:PGES;
END;
```

4.2. Description du module TRTES

Pour le module TRTES les variables sont limitées à cinq booléens qui vont permettre de tester si la variable fichier PASCAL a déjà servi pour un OPEN auquel cas, il faudra avant de faire la connexion dynamique, faire un CLOSE sur ce fichier.

Le module comporte aussi deux tables qui vont permettre la gestion des enregistrements temporaires TABCORU et TABCORS. Dans ces tables il doit exister autant d'entrées qu'il y a d'enregistrements possibles dans le fichier. Au départ d'une session ces tables sont mises à zéro. Au fur et à mesure qu'il y a écriture d'enregistrement dans le fichier temporaire, l'entrée de la table correspondante au numéro d'enregistrement est remplie avec la clé de stockage dans le fichier temporaire. Donc lorsqu'une entrée est différente de zéro, ceci veut dire que cet enregistrement est écrit dans le fichier temporaire à la clé indiquée par la valeur de l'entrée.

Ce module est composé de sept procédures indépendantes qui vont permettre d'exécuter les connexions dynamiques de fichier, ouverture ou fermeture, lecture et écriture :

INITRTES, CONNEC, OUVMAJ, LEC, OUVL, ECR, FERM

4.2.1. Procédure INITRTES

Cette procédure appelée par INITMODES au début de chaque session MODULECO sert à initialiser les booléens de contrôle de connexion dynamique à faux. Pas d'ouverture de fichier effectuée.

4.2.2. Procédure CONNEC

Cette procédure permet de connecter dynamiquement une variable de type FILE de PASCAL et un fichier physique. Les variables FILE sont au nombre de cinq. La variable connectée dépend de la valeur du CUT :

1 FICHIERS
2 FICHIERU
3 FICHIERV
4 FICHIERC
5 FICHIERI

4.2.3. Procédures OUVMAJ et OUVL

Ces procédures servent à ouvrir une variable de type FILE en mode mise à jour pour OUVMAJ, en mode lecture (quand cela est possible) pour OUVL. Le booléen correspondant pour la surveillance de connexion dynamique est mis à Vrai.

4.2.4. Procédure ECR

Cette procédure permet d'écrire un enregistrement physique sur le fichier correspondant au CUT. L'enregistrement à écrire se trouve dans un buffer pointé par ADBUF. Pour les fichiers système et utilisateur, soit les CUTS 1 et 2 les écritures ne sont pas effectuées directement sur le fichier permanent mais dans un fichier temporaire. Le passage des données du fichier temporaire dans le fichier permanent est exécuté par la procédure SAUVE du module TRAITEMP. Cette procédure est appelée soit en tant que commande par l'utilisateur soit automatiquement à la fin de chaque commande.

4.2.5. Procédure LEC

Cette procédure sert à lire un enregistrement physique sur le fichier correspondant au CUT. L'enregistrement est chargé dans le buffer pointé par ADBUF. Pour les CUTS 1 et 2 avant de lire dans le fichier permanent, il faut vérifier que cet enregistrement n'est pas stocké dans le fichier temporaire auquel cas c'est dans celui-ci qu'il faut le lire.

4.2.6. Procédure FERM

Cette procédure sert à fermer une variable de type FILE. Le booléen correspondant pour la surveillance de connexion dynamique est mis à Faux.

Format PASCAL

```
PROCEDURE INITRTES;

PROCEDURE CONNEC(NOMFICH:NOMFIC;CUT:INTEGER);
```

```
PROCEDURE OUVMAJ(CUT:INTEGER);

PROCEDURE OUVL(CUT:INTEGER);

PROCEDURE LEC(CLE:INTEGER;ADBUF:PFICH;CUT:INTEGER);

PROCEDURE ECR(CLE:INTEGER;ADBUF:PFICH;CUT:INTEGER);

PROCEDURE FERM(CUT:INTEGER);
```

4.3. Description du module TRAITEMP

Le module TRAITEMP se sert des mêmes variables que TRTES soit les deux tables TABCORU et TABCORS définissant la correspondance entre les enregistrements des fichiers temporaires et permanents (cf. 4.2).

Ce module est composé de trois procédures :
INITEMP, SAUVE, REPRISE

4.3.1. Procédure INITEMP

Cette procédure appelée par INITMODES au début de chaque session de MODULECO sert à initialiser les deux tables de correspondance à zéro.

4.3.2. Procédure SAUVE

Cette procédure est appelée soit automatiquement à la fin de chaque commande, soit directement par l'utilisateur quand il le désire. Elle recopie les enregistrements temporaires dans le fichier permanent. Au début de son exécution elle marque le fichier permanent "non cohérent", ce qui permet s'il y a un plantage système de repartir en réeffectuant la fin de la

sauvegarde. Lorsque toute la sauvegarde est effectuée le fichier permanent est remis "cohérent".

4.3.3. Procédure REPRISE

Cette procédure appellée par le système à chaque début de session MODULECO, vérifie si le fichier permanent est cohérent. Dans l'affirmative elle rend la main sans rien faire, dans la négative, elle restaure le fichier permanent à partir du fichier temporaire. Tout se passe comme si la sauvegarde était reprise au point où elle avait été abandonnée.

4.4. Primitives de Trace

Deux primitives de traces sont mises à la disposition des utilisateurs, DEBPROC et FINPROC. Ces primitives doivent être appelées en début et fin de procédure. Elles admettent un paramètre d'appel, une chaîne de caractères (alfa de PASCAL) qui est imprimée sur le listing de sortie. A chaque appel de DEBPROC, il y a décalage de la marge gauche de 2 caractères vers la droite. A chaque appel de FINPROC, il y a un décalage de 2 caractères vers la gauche. Ceci permet de visualiser l'imbrication des procédures lors de l'exécution.

```
PROCEDURE DEBPROC(NOM:ALFA);

PROCEDURE FINPROC(NOM:ALFA);
```

4.5. Primitives d'initialisation

Il existe cinq primitives d'initialisation :
INITMODES, INITSYS, DEBSYS, INITES, INIREPRI

4.5.1. Primitive INITMODES

Dans tous les cas la première qui doit être appelée est la primitive INITMODES, elle sert à initialiser la table de gestion des Entrées-Sorties afin de la mettre vide et appelle l'initialisation de TRTES pour positionner les booléens de contrôle d'ouverture des fichiers à faux.

4.5.2. Primitive INITSYS

La primitive INITSYS sert à enregistrer dans le premier enregistrement du fichier système les numéros de la première case des deux membres du segment système. Cette primitive est réservée au gestionnaire d'utilisateurs, elle est appelée uniquement lors de la génération du système MODULECO sur une nouvelle machine hôte.

4.5.3. Primitive DEBSYS

La primitive DEBSYS est en somme l'inverse de INITSYS. Cette primitive est réservée au gestionnaire d'objets, elle est appelée à chaque initialisation de session, afin de récupérer les numéros de case de début des deux membres du segment système permettant de reconstituer le descripteur de segment nécessaire à son traitement.

4.5.4. Primitive INITES

La primitive INITES effectue la connexion dynamique du fichier de nom NOMFICH à la variable fichier d'index CUT.

SI CUT = 1 FICHIERS

2 FICHIERSU

3 FICHIERSV

4 FICHIERC

Lecture premier enregistrement du fichier pour récupération de :

- NUMLIB : Numéro première case de la chaîne des cases libres
- TAILLE : Nombre de cases du fichier

Erreurs détectées

13 MODE D'OUVERTURE ERONNE (L OU E)

16 CUT ERRONNE

4.5.5. Primitive INIREPRI

Cette procèdure est appelée par le système MODULECO, pour la reprise après une interruption de l'utilisateur, provoquée par la frappe d'un BREAK sur sa console. Le contrôle est soit rendu en séquence, soit au langage de commande, pour attendre la commande suivante. Dans ce dernier cas il faut restaurer les tables du module d'Entrées-Sorties, c'est le rôle de la procédure INIREPRI. De plus elle appelle l'initialisation de TRTES pour positionner les booléens de contrôle d'ouverture des fichiers à faux. Puis ensuite la procédure REPGEST qui est chargée de repositionner les fichiers environnements utilisateurs et système.

Format PASCAL

```
PROCEDURE INITMODES;

PROCEDURE INITSYS(P:PDESCRSG);

PROCEDURE DEBSYS(VAR NOMFICH:NOMFIC;VAR DBMB1,DBMB2:INTEGER);

PROCEDURE INITES(VAR NOMFICH:NOMFIC;CUT:INTEGER;MOD:CHAR);

PROCEDURE INIREPRI;
```

4.6. Primitive de formattage du fichier

Cette primitive appelée à chaque création d'un nouvel utilisateur effectue le formattage du fichier de nom NOMFICH et chaîne tous les enregistrements entre eux afin de constituer une chaîne de vide. Le numéro de la première case libre, le nombre de cases sont écrits dans le premier enregistrement du fichier.

Format PASCAL

```
PROCEDURE IFIC(VAR NOMFICH:NOMFIC;TAILLE:INTEGER);
```

Erreurs détectées

12 TAILLE DU FICHIER PLUS GRANDE QUE TAILLE AUTORISEE

23 TAILLE DU FICHIER ERRONNE

4.7. Primitives de gestion de fichier

Il existe trois primitives et deux commandes de gestion de fichier :
TAUXOCC, COPYFIC, FNBCASE, FMAXMB, ETATFICH

4.7.1. Commande TAUXOCC

La commande TAUXOCC imprime à la console le nombre de cases du fichier utilisateur et le taux d'occupation de celui-ci.

4.7.2. Primitive FMAXMB

La primitive FMAXMB rend :

- en ASEQ, numéro de la dernière clé écrite,

- en ADBL, numéro de la plus grande clé répertoriée
- en ADCH, numéro de la plus grande clé répertoriée dans la zone principale, soit le nombre d'enregistrements déclarés au CRMB

Cette primitive sert dans le cas de copie de segment, afin de connaître la taille de chaque membre. Le segment concerné doit avoir été préalablement ouvert. L'utilisateur doir fournir le code qui lui a été rendu à l'OUVRE et le nom du membre concerné.

Erreur détectées par FMAXMB

9 MEMBRE NON TROUVE

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

4.7.3. Primitive FNBCASE

La primitive FNBCASE rend, le nombre de cases réservées et le nombre de clés utilisées pour le membre de nom NOMMB et dont le descripteur est pointé par P. Le segment doit-être ouvert avant l'appel de la primitive.

Erreurs détectées par FNBCASE

9 MEMBRE NON TROUVE

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

4.7.4. Primitive ETATFICH

La primitive ETATFICH sert à connaitre l'état d'un fichier désigné par un cut. Pour un fichier deux états sont possibles : bon ou mauvais. Un fichier peut-être mauvais s'il y a eu une erreur pendant la procédure de sauvegarde du fichier temporaire. Dans ce cas il sera remis en état par la procédure de reprise lors du prochain appel de MODULECO par le propriétaire du fichier.

La primitive ETATFICH renvoie un booléen positionne à :

- TRUE le fichier est valide
- FALSE le fichier est dans un état incohérent

4.7.5. Commande COPYFIC

La commande COPYFIC sert à agrandir le fichier utilisateur dans le cas d'obtention du message "FICHIER UTILISATEUR PLEIN". Dans ce cas l'utilisateur doit contacter l'administrateur du site qui lui agrandit son fichier. Pour cela il doit :

- Recopier l'ancien fichier sous un autre nom
- Créer un fichier plus grand avec le nom de l'utilisateur, ce nom est impératif, car c'est ce nom qui est enregistré dans le fichier système de MODULECO et qui est appelé lors de chaque Login.
- Appeler la commande COPYFIC pour recopier l'ancien fichier dans le nouveau.

Seul l'administrateur du site à accès à la commande COPYFIC.

Erreurs détectées par COPYFIC

12 TAILLE DU FICHIER PLUS GRANDE QUE TAILLE AUTORISEE

13 MODE D'OUVERTURE ERRONE (L OU E)

16 CUT ERRONE

23 TAILLE DU FICHIER ERRONNE

41 TAILLE ANCIEN ET NOUVEAU FICHIER INCOMPATIBLE

43 CLE INVALIDE

Format PASCAL

```
PROCEDURE TAUXOCC;

FUNCTION FMAXMB(CODE:INTEGER;NOMMB:NMMBR):INTEGER;

PROCEDURE FNBCASE(COD:INTEGER;NOMMB:NMMBR;VAR NBCASE:INTEGER;
                  VAR NBCLE:INTEGER);
```

```
FUNCTION ETATFICH(CUT:INTEGER):BOOLEAN;

PROCEDURE COPYFIC;
```

4.8. Primitives de gestion des segments

Il existe sept primitives de gestion de segments : CRMB, DETSEG, OUVRE, FERME, DETMBR, COPIE, TSTFERME

4.8.1 Primitive CRMB

La primitive CRMB sert à créer un membre. L'utilisateur doit fournir un pointeur sur un descripteur de segment où seront rangés les renseignements nécessaires au module E-S :

- Nom du membre
- Mode d'utilisation du membre
- Nombre d'enregistrements prévus en accès direct
- Pointeur sur le descripteur de segment
- CUT entier désignant le fichier sur lequel se trouve le segment.

La primitive CRMB se contente :

- de réserver les cases nécessaires aux membres en fonction du mode d'utilisation défini plus haut
- d'initialiser les chaînages et les tables d'index
- de remplir le descripteur de segment

Erreurs détectées

16 CUT ERRONNE

21 DESCRIPTEUR DE SEGMENT PLEIN

22 ERREUR MODE D'ACCES

24 FICHIER UTILISATEUR PLEIN

27 LONGUEUR ENREGISTREMENT ERRONNE

29 NOMBRE ENREGISTREMENTS ERRONNES

31 LONGUEUR ENREGISTREMENT PLUS GRANDE QUE LONGUEUR CASE

42 MEMBRE DEJA EXISTANT

43 CLE INVALIDE

500 ZONE DYNAMIQUE PASCAL SATUREE

4.8.2. Primitive DETSEG

La primitive DETSEG sert à détruire un segment. L'utilisateur doit fournir un pointeur sur le descripteur de segment et le CUT (entier désignant le fichier à prendre en considération). Tous les membres se trouvant dans le segment sont détruits, les cases occupées sont remises dans la chaîne des cases libres. Le nom des membres est remis à blanc dans le descripteur de segment. La primitive DETSEG ne peut être exécutée sur un segment ouvert.

EErreurs detectées

16 CUT ERRONNE

28 MEMBRE ADCH MAL STRUCTURE, VOIR MAINTENANCE

4.8.3. Primitive DETMBR

La primitive DETMBR sert à détruire un seul membre dans un segment. L'utilisateur doit fournir un pointeur sur le descripteur de segment, le nom du membre concerné et le CUT, entier désignant le fichier à prendre en considération. Le membre est détruit, les cases encore occupées sont remises dans la chaîne des cases libres. Le nom du membre est remis à blanc dans le descripteur de segment. La primitive DETMBR ne peut être

exécutée sur un segment ouvert.

Ereurs détectées par DETSEG ou DETMBR

16 CUI ERRONNE

28 MEMBRE ADCH MAL STRUCTURE, VOIR MAINTENANCE

4.8.4. Primitive OUVRE

La primitive OUVRE sert à ouvrir un segment. L'utilisateur doit fournir un pointeur sur le descripteur de segment, le mode d'ouverture (lecture ou écriture), l'indice du fichier utilisé. La primitive :

réserve une case dans la table de gestion des Entrées-Sorties,

y écrit pour chaque membre les renseignements nécessaires à l'exécution des primitives de lecture et écriture.

réserve les buffers nécessaires, soit :
- un buffer pour lire les enregistrements
- un buffer pour la table d'index pour l'accès direct
- un buffer pour la table d'index de la zone débordement dans le cas d'accès direct chaîné.

La primitive OUVRE rend à l'utilisateur un code que celui-ci devra fournir pour toutes les primitives de traitement sur les membres de ce segment. Ce code permet aux modules d'Entrées-Sorties de retrouver l'indice du fichier à traiter et la case de la table de gestion des Entrées-Sorties où est enregistré le segment concerné. Il inclut aussi le CUT du fichier.

Erreurs détectées par OUVRE

7 OUVERTURE SEGMENT INCOMPATIBLE AVEC OUVERTURE FICHIER

13 MODE D'OUVERTURE ERRONNE (L OU E)

14 TABLE DE GESTION DES ENTREES-SORTIES PLEINES

16 CUT ERRONNE

24 FICHIER UTILISATEUR PLEIN

37 DESCRIPTEUR DE SEGMENT VIDE

43 CLE INVALIDE

500 ZONE DYNAMIQUE PASCAL SATUREE

4.8.5. Primitive FERME

La primitive FERME sert à fermer un segment. L'utilisateur doit fournir un pointeur sur le descripteur de segment et le code que lui a rendu l'OUVRE effectué sur ce segment. Dans le cas où la dernière primitive exécutée pour chaque membre était une écriture ou une invalidation, écrit le dernier buffer qui peut encore se trouver en mémoire et corrige les pointeurs de la prmière case du membre dans le cas d'accès direct. La primitive FERME rend la mémoire réservée par OUVRE pour le fichier.

Erreurs détectées

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

43 CLE INVALIDE

4.8.6. Primitive COPIE

La primitive COPIE permet de copier un segment dont le descripteur est pointé par PENT dans un segment de descripteur PSORT. Le segment d'origine peut-être dans le fichier de l'utilisateur ou dans le fichier d'un autre utilisateur. Ce fichier est désigné par la variable FICENT qui donne le CUT du fichier d'origine. Le fichier destinataire quant à lui est toujours celui de l'utilisateur car il est interdit d'écrire dans le fichier d'un autre .

Pour réaliser la copie de segment, la commande COPIER doit:

- créer un nouveau descripteur de segment

- créer les différents membres à l'aide de la primitive CRMB en donnant le nombre d'enregistrements nécessaire pour stocker tout le membre dans son état actuel. Ce nombre peut-être obtenu à l'aide de la primitive FMAXMB.

- copier les cases 2 à n (la dernière comporte un chaînage nul),en utilisant les primitives LEC et ECR directement.

Du fait de l'utilisation des primitives LEC et ECR la copie des segments est bien plus rapide mais les compteurs qui se trouvent dans le premier enregistrement du membre ne sont pas mis à jour. Cette mise à jour est faite en deux temps :

- au CRMB pour les pointeurs sur les cases et clés réservées

- dans la primitive COPIE elle-même pour les pointeurs :

  début de chaîne des enregistrements occupés

  début de chaîne des enregistrements libres

  numéro dernière clé utilisée

La primitive COPIE de plus pour les membres en accès direct chaîné s'occupe d'ajuster la longueur de la zone secondaire. Après le CRMB, le membre est créé avec une partie secondaire comportant une seule case. Il faut donc l'allonger en fonction de la plus grande clé réservée dans le membre du segment à copier.

Erreurs détectées par COPIE

7 OUVERTURE SEGMENT INCOMPATIBLE AVEC OUVERTURE FICHIER

13 MODE D'OUVERTURE ERRONE (L OU E)

14 TABLE DE GESTION DES ENTREES-SORTIES PLEINES

16 CUT ERRONE

24 FICHIER UTILISATEUR PLEIN

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

37 DESCRIPTEUR DE SEGMENT VIDE

43 CLE INVALIDE

500 ZONE DYNAMIQUE PASCAL SATUREE

4.8.7. Primitive TSTFERME

La primitive TSTFERME est appelée par la commande SAUVE qui est exécutée à la fin de chaque commande dans le cas de sauvegarde automatique ou sur appel de l'utilisateur dans le cas contraire.

Cette primitive vérifie dans la table de gestion des Entrées-Sorties qu'il n'y a plus de segment ouvert. Elle écrit sur le fichier SORT la liste des membres de chaque segment encore ouvert.

Cette primitive rend un booléen :

TRUE plus de segments ouverts, la sauvegarde est possible

FALSE il reste des segments ouverts, la sauvegarde est abandonnée

La dernière commande exécutée comporte une erreur de programmation.

Format Pascal

```
PROCEDURE CRMB(UNOMMB:NMMBR;UMODE:MOD1;ULGENREG:INTEGER;
               NBENREG:INTEGER;PTR:PDESCRSG;CUTP:INTEGER);

PROCEDURE DETSEG(PTR:PDESCRSG;CUTP:INTEGER);

PROCEDURE DETMBR(NOMMB:NMMBR;P:PDESCRSG;CUTP:INTEGER);

PROCEDURE OUVRE(PTR:PDESCRSG;MODOUV:CHAR;VAR CODE:INTEGER;
                CUT:INTEGER);

PROCEDURE FERME(PTR:PDESCRSG;CODE:INTEGER);

PROCEDURE COPIE(PENT,PSORT:PDESCRSG;FICENT:INTEGER);

FUNCTION TSTFERME:BOOLEAN;
```

4.9. Primitives d'écriture

Il existe quatre primitives d'écriture : ECRSQ, ECRDB, ECRCH, ECRDC

4.9.1. Primitive ECRSQ

La primitive ECRSQ permet d'écrire un enregistrement dans un membre en accès séquentiel. L'utilisateur doit fournir :

- le code délivré pour le segment par la primitive OUVRE,
- le nom du membre où doit se faire l'écriture,
- un pointeur sur le buffer contenant les données à écrire. Le buffer doit avoir la longueur déclarée au CRMB. Si le buffer est plus long, des données seront perdues, si le buffer est plus court, les renseignements se trouvant derrière seront écrits dans le membre. Bien entendu il est toujours possible de donner un buffer plus court ou plus long, à condition de bien savoir ce que l'on fait, ce qui n'est pas le cas pour la lecture (voir primitive LIRSQ).
- un booléen qui doit toujours être à faux dans un cas d'utilisation normal. Mis à vrai il sert à recommencer au début du membre. Si un utilisateur, met toujours le booléen à vrai, il écrit toujours dans le premier enregistrement du membre. Il n'est pas obligatoire de le mettre à vrai après un OUVRE, de toute façon la première écriture ou lecture s'effectue au début du membre.

Lors de l'appel de la primitive ECRSQ, il y a un simple transfert du buffer utilisateur dans le buffer de la case du fichier, c'est seulement lors du changement de case qu'il y a une écriture sur le disque.

Erreurs détectées par la primitive ECRSQ

8 OUVERTURE INCOMPATIBLE AVEC UTILISATION

9 MEMBRE NON TROUVE

24 FICHIER UTILISATEUR PLEIN

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

43 CLE INVALIDE

44 ERREUR MODE DE TRAIT. ASEQ, ADBL, ADCH

46 MELANGE DE PRIMITIVE ECRSQ ET LIRSQ INTERDIT

500 ZONE DYNAMIQUE PASCAL SATUREE

4.9.2. Primitive ECRDB

La primitive ECRDB permet d'écrire un enregistrement dans un membre en accès direct bloqué. L'utilisateur doit fournir :

- le code délivré pour le segment par la primitive OUVRE
- le nom du membre où doit se faire l'écriture
- un pointeur sur le buffer contenant les données à écrire. Voir explication primitive ECRSQ
- le numéro de la clé où doit se faire l'écriture. Pour cet accès, l'utilisateur peut soit fournir un numéro de clé, soir donner une clé nulle. Mais ceci a un sens pour la primitive :

  une clé nulle signifie nouvelle clé à écrire, la primitive prend la clé suivante dans la chaîne des libres et en rend le numéro à l'utilisateur pour que celui-ci puisse la conserver si besoin est;

  une clé non nulle signifie, mise à jour d'une clé déjà existante.

ATTENTION : pour l'écriture de nouvelles clés, rien n'assure que la primitive fasse clé plus un pour prendre la suivante, mais comme dit plus haut il y a réservation d'un enregistrement dans la chaîne des libres. Du fait de la récupération des trous lors des invalidations, il y aura donc des mélanges de clés. Pour lire ces clés séquentiellement, il faudra donc les lire avec clés nulles, du fait qu'elles sont chaînées, elles seront rendues dans l'ordre où elles

ont été écrites.

Erreurs détectées par la primitive ECRDB

8 OUVERTURE INCOMPATIBLE AVEC UTILISATION

9 MEMBRE NON TROUVE

15 ECRITURE CLE EN DEHORS DU MEMBRE EN ACCES DIRECT

24 FICHIER UTILISATEUR PLEIN

29 NOMBRE ENREGISTREMENTS ERRONES

30 LE CODE NE CORESPOND PAS A UN SEGMENT OUVERT

34 ECRITURE EN CREATION, CLE DEJA OCCUPEE

35 MELANGE DE PRIMITIVE DB ET CH INTERDIT

40 MISE A JOUR SUR UNE CLE INVALIDE

43 CLE INVALIDE

44 ERREUR MODE DE TRAIT. ASEQ, ADBL, ADCH

500 ZONE DYNAMIQUE PASCAL SATUREE

4.9.3. Primitive ECRCH

La primitive ECRCH permet d'écrire un enregistrement dans un membre en accès direct bloqué. Cette primitive n'introduit pas un nouveau type d'enregistrement, ce n'est qu'un cas particulier de ECRDB. Elle permet de fournir l'enregistrement logique par morceau et uniquement cela. C'est à dire que la somme des morceaux doit correspondre rigoureusement à la longueur d'un enregistrement. Il n'est absolument pas question d'écrire des morceaux de buffer redécoupés à cheval sur deux enregistrements, ECRCH se contente de recoller les morceaux et dès qu'une longueur exacte de buffer est atteinte, se comporte comme ECRDB et cela aussi bien en création qu'en mise à jour.

Erreurs détectées par la primitive ECRCH

8 OUVERTURE INCOMPATIBLE AVEC UTILISATION

9 MEMBRE NON TROUVE

15 ECRITURE CLE EN DEHORS DU MEMBRE EN ACCES DIRECT

20 LECTURE OU ECRITURE PAR MORCEAU, ON SORT DU BUFFER

24 FICHIER UTILISATEUR PLEIN

27 LONGUEUR ENREGISTREMENT ERRONEE

29 NOMBRE ENREGISTREMENTS ERRONES

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

34 ECRITURE EN CREATION, CLE DEJA OCCUPEE

36 MAJ, ON CHANGE DE CASE ET LE BUFF N'EST PAS PLEIN

40 MISE A JOUR SUR UNE CLE INVALIDE

43 CLE INVALIDE

44 ERREUR MODE DE TRAIT. ASEQ, ADBL, ADCH

500 ZONE DYNAMIQUE PASCAL SATUREE

4.9.4. Primitive ECRDC

La primitive ECRDC permet d'écrire un enregistrement dans un membre en accès direct chaîné. L'utilisateur doit fournir :

- le code délivré pour le segment par la primitive OUVRE
- le nom du membre où doit se faire l'écriture
- un pointeur sur le buffer contenant les données à écrire. Voir explication primitive ECRSQ
- le numéro de la clé de l'enregistrement à écrire. Dans le cas d'une création, ce numéro ne doit jamais être nul. Dans le cas d'une mise à jour, ce numéro n'a pas besoin d'être rempli, car il faut avoir lu l'enregistrement pour le mettre à jour, c'est le dernier enregistrement lu dans ce membre qui est mis à jour

- un booléen qui précise si l'on se trouve en création de nouvelles clés ou en mise à jour. Si le booléen est vrai, c'est un nouvel enregistrement à créer, dans ce cas, si le numéro de clé est différent de zéro, il est créé en zone principale, si le numéro de clé est égal à zéro, il est chaîné en zone secondaire. Si le booléen est à faux, c'est la mise à jour d'un enregistrement existant, il doit avoir été lu auparavant, c'est le dernier enregistrement lu qui est mis à jour.

Erreurs détectées par la primitive ECRDC

8 OUVERTURE INCOMPATIBLE AVEC UTILISATION

9 MEMBRE NON TROUVE

15 ECRITURE CLE EN DEHORS DU MEMBRE EN ACCES DIRECT

24 FICHIER UTILISATEUR PLEIN

29 NOMBRE ENREGISTREMENT ERRONES

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

32 CLE NE DOIT PAS ETRE NULLE EN ADCH POUR ECRIRE

40 MISE A JOUR SUR UNE CLE INVALIDE

43 CLE INVALIDE

44 ERREUR MODE DE TRAIT. ASEQ, ADBL,ADCH

45 IL FAUT LIRE L'ENREGISTREMENT POUR LE METTRE A JOUR

500 ZONE DYNAMIQUE PASCAL SATUREE

Format Pascal

```
PROCEDURE ECRSQ(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                BDEB:BOOLEAN);

PROCEDURE ECRDB(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                VAR CLE:INTEGER);

PROCEDURE ECRCH(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
               VAR CLE:INTEGER;LGBUF:INTEGER);

PROCEDURE ECRDC(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;CLE:INTEGER;
               BNCLE:BOOLEAN);
```

4.10. Primitives de lecture

Il existe cinq primitives de lecture : LIRSQ, LIRDB, LIRCH, LIRDC, LIRSQDC

4.10.1. Primitive LIRSQ

La primitive LIRSQ permet de lire un enregistrement dans un membre en accès séquentiel. L'utilisateur doit fournir :

- le code délivré pour le segment par la primitive OUVRE
- le nom du membre où doit se faire la lecture
- un pointeur sur le buffer où seront rangées les données lues.
  Attention : le buffer doit avoir au moins la longueur déclarée au CRMB. Si le buffer est plus long, ce n'est pas grave, seule la fin du buffer ne sera pas remplie, mais si le buffer est plus court, les zones se trouvant derrière seront détruites par le module d'Entrées-Sorties sur la longueur déclarée au CRMB.
- un booléen qui doit toujours être faux dans un cas d'utilisation normale. Mis à vrai, il sert à recommencer au début du membre. Après avoir écrit un membre il est donc inutile de faire un FERME et un OUVRE pour le relire, il suffit de mettre le booléen à vrai pour la première lecture. Il ne faut pas oublier de le remettre à faux pour les appels suivants, sinon c'est toujours le premier enregistrement qui est délivré. La lecture sur disque n'est pas faite à chaque appel de la primitive LIRSQ, mais uniquement lorsqu'il y a changement de case.

Erreurs détectées par la primitive LIRSQ

9 MEMBRE NON TROUVE

10 FIN DE MEMBRE

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

43 CLE INVALIDE

44 ERREUR MODE DE TRAIT. ASEQ, ADBL, ADCH

46 MELANGE DE PRIMITIVE ECRSQ ET LIRSQ INTERDIT

4.10.2. Primitive LIRDB

La primitive LIRDB permet de lire un enregistrement dans un membre en accès direct bloqué. L'utilisateur doit fournir :

- le code délivré pour le segment par la primitive OUVRE
- le nom du membre où doit se faire la lecture
- un pointeur sur le buffer où seront rangées les données lues. Voir explication primitive LIRSQ
- le numéro de la clé qui doit-être lue. Pour cet accès, l'utilisateur peut soit fournir un numéro de clé, soit donner une clé nulle.
  une clé non nulle signifie lire l'enregistrement ayant ce numéro de clé

  une clé nulle signifie lire l'enregistrement suivant.

  Attention : Pour les chaînes d'enregistrements, il faut donner la clé de la première et lire la suite avec clé nulle. Comme expliqué lors de ECRDB rien n'assure que la clé qui suit la clé 15 est bien la clé 16. Toutes les clés sont chaînées entre elles, seul le module d'E/S connaît le numéro de la suivante.

4.10.3. Primitive LIRCH

La primitive LIRCH permet de lire un enregistrement dans un membre en accès direct bloqué. Comme exposé lors de la primitive ECRCH ce n'est qu'un cas particulier de LIRDB. Elle permet de récupérer l'enregistrement logique par morceaux. Pour la primitive LIRCH, l'utilisateur doit fournir les mêmes paramètres que LIRDB avec en plus la longueur du buffer fourni qui est donc la longueur du morceau d'enregistrement rendu. La somme des

longueurs de morceaux doit être égale à la longueur d'enregistrement déclarée au CRMB.

Erreurs détectées par les primitives LIRDB et LIRCH

9 MEMBRE NON TROUVE

10 FIN DU MEMBRE

17 LECTURE D'UNE CLE EN DEHORS DU MEMBRE EN ACC DIRECT

18 LECTURE D'UNE CLE INEXISTANTE

20 LECTURE OU ECRITURE PAR MORCEAU, ON SORT DU BUFFER

25 LEC APRES ECRI PAR MORCEAU ET L'ENR N'EST PAS FINI

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

43 CLE INVALIDE

44 ERREUR MODE DE TRAIT. ASEQ, ADBL, ADCH

4.10.4. Primitive LIRDC

La primitive LIRDC permet de lire un enregistrement dans un membre en accès direct chaîné. L'utilisateur doit fournir :

- le code délivré pour le segment par la primitive OUVRE
- le nom du membre où doit se faire la lecture
- un pointeur sur le buffer où seront rangées les données lues. Voir explication primitive LIRSQ
- le numéro de la clé de l'enregistrement à lire. Ce numéro doit être non nul pour la première lecture d'un enregistrement de clé donné (lecture en zone principale) puis nul pour lire la suite de la chaîne des enregistrements de même numéro de clé.
- un booléen est rendu par la primitive pour signaler la fin d'une chaîne d'enregistrements. Lorsque le booléen est vrai la chaîne est terminée, le buffer rendu est vide, lorsqu'il est faux, la chaîne rendue n'est pas finie, le buffer contient l'enregistrement courant.

Erreurs détectées par la primitive LIRDC

9 MEMBRE NON TROUVE

17 LECTURE D'UNE CLE EN DEHORS DU MEMBRE EN ACC DIRECT

18 LECTURE D'UNE CLE INEXISTANTE

19 LECTURE ZONE SECONDAIRE SUR CLE INEXISTANTE

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

33 EN ADCH, IL FAUT LIRE EN ACCES DIRECT AVANT SEQUENTIEL

43 CLE INVALIDE

44 ERREUR MODE DE TRAIT. ASEQ, ADBL, ADCH

4.10.5 Primitive LIRSQDC

La primitive LIRSQDC permet de lire un membre d'accès direct chaîné en mode séquentiel. A chaque appel de la primitive un enregistrement est délivré jusqu'à la fin du membre.

Le membre est parcouru clé par clé, à l'intérieur de chaque clé, la chaîne d'enregistrements est parcourue entièrement avant de passer à la clé suivante.

Erreurs détectées par LIRSQDC

10 FIN DU MEMBRE

Format Pascal

```
FUNCTION LIRSQ(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;BDEB:BOOLEAN)
                :INTEGER;

FUNCTION  LIRDB(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;CLE:INTEGER)
                :INTEGER;

FUNCTION  LIRCH(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;CLE:INTEGER;
                LGBUF:INTEGER):INTEGER;

FUNCTION  LIRDC(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;CLE:INTEGER;
                VAR FCHN:BOOLEAN):INTEGER;
```

```
FUNCTION LIRSQDC(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                 BDEB:BOOLEAN):INTEGER;
```

4.11. Primitives d'invalidation

Quatre primitives d'invalidation sont disponibles pour l'utilisateur, qui servent uniquement sur les organisations en accès direct. Trois servent pour l'accès direct bloqué : INVDB, INVCH, INVCHB. Une pour l'accès direct chaîné INVDC.

4.11.1. Primitives INVDB, INVDC

Les primitives INVDB et INVDC servent à invalider le dernier article lu dans le membre désigné. L'utilisateur doit fournir :

- le code délivré pour le segment par la primitive OUVRE
- le nom du membre où doit se faire l'invalidation Le dernier enregistrement lu sur ce membre est déchaîné de la chaîne des enregistrements occupés et remis en tête de la chaîne des libres.

Erreurs détectées par les primitives INVDB et INVDC

8 OUVERTURE INCOMPATIBLE AVEC UTILISATION

22 ERREUR MODE D'ACCES

26 IL FAUT LIRE L'ENREGISTREMENT POUR L'INVALIDER

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

43 CLE INVALIDE

4.11.2. Primitive INVCH

La primitive INVCH permet d'invalider le dernier article lu dans le membre désigné. L'utilisateur doit fournir :

- le code délivré pour le segment par la primitive
- le nom du membre où doit se faire l'invalidation

Le dernier enregistrement lu sur ce membre est seulement invalidé, mais n'est pas déchaîné de la chaîne des enregistrements occupés.

Erreurs détectées par la primitive INVCH

8 OUVERTURE INCOMPATIBLE AVEC UTILISATION

22 ERREUR MODE D'ACCES

26 IL FAUT LIRE L'ENREGISTREMENT POUR L'INVALIDER

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

38 ENREGISTREMENT DEJA INVALIDE

4.11.3. Primitive INVCHB

La primitive INVCHB permet d'invalider une chaîne de N articles, commençant par le dernier article lu dans le membre. Cette primitive sert pour le traitement par morceaux afin de faciliter l'invalidation des séries de la base de données. L'utilisateur doit fournir :

- le code délivré pour le segment par la primitive OUVRE
- le nom du membre où doit se faire l'invalidation
- le nombre d'articles à invalider

En commençant par le dernier enregistrement lu sur ce membre, N enregistrements sont déchaînés de la chaîne des enregistrements occupés et remis en tête de la chaîne des libres.

Erreurs détectées par la primitive INVCHB

8 OUVERTURE INCOMPATIBLE AVEC UTILISATION

22 ERREUR MODE D'ACCES

26 IL FAUT LIRE L'ENREGISTREMENT POUR L'INVALIDER

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

43 CLE INVALIDE

Format Pascal

```
PROCEDURE INVDB(CODE:INTEGER;NOMMB:NMMBR);

PROCEDURE INVDC(CODE:INTEGER;NOMMB:NMMBR);

PROCEDURE INVCH(CODE:INTEGER;NOMMB:NMMBR);

PROCEDURE INVCHB(CODE:INTEGER;NOMMB:NMMBR;NBENR:INTEGER);
```

5. ERREURS DETECTEES DANS LE MODULE MODES.

Toutes les erreurs recensées dans le module MODES sont répertoriées ci-dessous :

7 OUVERTURE SEGMENT INCOMPATIBLE AVEC OUVERTURE FICHIER
L'utilisateur désire ouvrir le segment en mode écriture, hors l'INITES sur le fichier qui le contient a été fait en mode lecture, il y a donc incompatibilité.

8 OUVERTURE INCOMPATIBLE AVEC UTILISATION
Sur un segment ouvert en lecture les primitives ECRSQ, ECRDB, ECRCH, ECRDC, INVDB, INVDC, INVCH, INVCHB ne sont pas permises.

9 MEMBRE NON TROUVE
Le membre de nom NOMMB ne se trouve pas dans le segment désigné par CODE dans la table de gestion des Entrées-Sorties.

10 ° FIN DE MEMBRE §
Signale à l'utilisateur que la fin du membre est atteinte. Ce code est rendu par les primitives permettant une lecture séquentielle soit :
LIRSQ, LIRDB, LIRCH, LIRSQDC

12 TAILLE DU FICHIER PLUS GRANDE QUE TAILLE AUTORISEE
La taille demandée par l'utilisateur est plus grande que celle permise actuellement dans MODULECO. La taille maximum autorisée est de 1020 cases. Pour réserver des fichiers de plus de 1020 cases il faut recompiler les modules MODES, TRTES ou PLIES et TRAITEMP après avoir modifié la constante NBTABU qui détermine la dimension des tables pour les fichiers temporaires.

13 MODE D'OUVERTURE ERRONE (L OU E)
Le mode d'ouverture d'un segment doit-être écriture ou lecture

14 TABLE DE GESTION DES ENTREES-SORTIES PLEINES
L'ouverture d'un segment est demandée par l'utilisateur alors que la table de gestion des Entrées-Sorties est saturée. Dans le module actuel, vingt ouvertures de segment simultanées sont possibles. Si l'on veut augmenter ce nombre il faut changer la constante NBENTES et recompiler le module MODES.

15 ECRITURE CLE EN DEHORS DU MEMBRE EN ACCES DIRECT
En accès direct bloqué, écriture d'une clé en mise à jour

qui est en dehors des limites du membre actuel.
En accès direct chaîné, la clé fournie par l'utilisateur est plus que celle donnée au CRMB.

16 CUT ERRONE
L'utilisateur a fourni un CUT en dehors de la plage de valeur 1..4 connu dans MODES

17 LECTURE D'UNE CLE EN DEHORS DU MEMBR EN ACC DIRECT
En accès direct bloqué la clé fournie par l'utilisateur est négative.
En accès direct chaîné la clé fournie par l'utilisateur est soie négative, soit plus grande que le nombre de clés prévu au CRMB.

18 ° CLE INEXISTANTE §
En accès direct bloqué la clé fournie par l'utilisateur est plus grande que la plus grande clé répertoriée dans le membre.
En accès direct chaîné la clé fournie par l'utilisateur est bien dans les limites du membre, mais elle n'a jamais été écrite.

19 LECTURE ZONE SECONDAIRE SUR CLE INEXISTANTE
Lors de l'accès à la zone principale il a été rendu erreur 18, soit clé inexistante et l'utilisateur tente quand même de lire la suite de la chaîne en zone secondaire.

20 LECTURE OU ECRITURE PAR MORCEAU, ON SORT DU BUFFER
La somme de la longueur des différents morceaux lus ou écrits, ne fait pas la longueur exacte déclarée au CRMB

21 DESCRIPTEUR DE SEGMENT PLEIN
L'utilisateur demande la création d'un nouveau membre alors que le descripteur fournie est déjà plein. Il n'est possible de créer que 7 membres par segment.

22 ERREUR MODE D'ACCES
Au CRMB les seuls modes d'accès permis sont ASEQ, ADBL, ADCH
En accès direct bloqué, l'utilisateur appelle la primitive INVDC, se qui est interdit.
En accès direct chaîné, l'utilisateur appelle la primitive INVDB, ce qui est interdit.

23 TAILLE DU FICHIER ERRONE
La taille du fichier communiquée par l'utilisateur est négative ou nulle.

24 FICHIER UTILISATEUR PLEIN
Le fichier de l'utilisateur est saturé. Il doit le faire agrandir par l'administrateur du site.

25 LEC APRES ECRI PAR MORCEAU ET L'ENR NEST PAS FINI

L'utilisateur a demandé une lecture d'un morceau d'enregistrement direct bloqué à l'aide de la primitive LIRCH. La primitive utilisée précédemment sur ce membre était ECRCH et le buffer n'étant pas plein l'opération d'écriture n'est pas terminée. Avant de lire il faut terminer l'écriture en cours.

26 IL FAUT LIRE LENREGISTREMENT POUR L'INVALIDER

L'invalidation d'un enregistrement s'effectue sur le dernier lu pour le membre, hors la dernière primitive utilisée n'était pas une primitive de lecture.

27 LONGUEUR ENREGISTREMENT ERRONE

L'utilisateur a fourni une longueur d'enregistrement négative ou nulle pour un CRMB ou ECRCH.

Cette erreur peut se produire aussi dans le cas de la primitive ECRCH si la longueur du morceau de buffer est supérieure à la longueur d'enregistrement déeclarée au CRMB.

28 MEMBRE ADCH MAL STRUCTURE, VOIR MAINTENANCE

Ce membre en accès direct chaîné ne comporte pas de zone débordement. Voir la maintenance, car il doit y avoir une erreur dans le module MODES.

29 NOMBRE ENREGISTREMENTS ERRONES

Cette erreur peut se produire dans deux cas :

- Pour CRMB, le nombre d'enregistrements prévus dans le membre est négatif ou nul
- Pour INVCHB, le nombre d'enregistrements à invalider est négatif ou nul

30 LE CODE NE CORRESPOND PAS A UN SEGMENT OUVERT

Cette erreur peut se produire dans les primitives : FNBCASE, FMAXMB, FERME, ECRSQ, ECRDB, ECRCH, ECRDC, LIRSQ, LIRDB, LIRCH, LIRDC, LIRSQDC, INVDB, INVDC, INVCH, INVCHB

Le code fourni par l'utilisateur est soit négatif ou nul, soit désigne une entrée de la table de gestion des Entrées-Sorties non utilisées. Ce code rendu par la primitive OUVRE a du être détruit.

31 LONGUEUR ENREGISTREMENT PLUS GRANDE QUE LONGUEUR CASE

En accès séquentiel la longueur d'un enregistrement ne doit pas être plus grande que la longueur d'une case.

En accès direct chaîné ou bloqué la longueur d'un enregistrement est au maximum égale à la longueur d'une case moins deux mots.

32 CLE NE DOIT PAS ETRE NULLE EN ADCH POUR ECRIRE

En accès direct chaîné, en écriture la clé donnée par l'utilisateur ne doit jamais être nulle.

33 EN ADCH, IL FAUT LIRE EN ACC DIRECT AVANT SEQUENTIELLE
L'utilisateur a donné une clé nulle à la primitive LIRDC pour lire la suite d'une chaîne d'enregistrements en secondaire, hors la dernière primitive exécutée sur ce membre n'est pas une lecture.

34 ECRITURE EN CREATION, CLE DEJA OCCUPEE
Cette erreur arrive lors de l'appel des primitives ECRDB, ECRCH, ECRDC pour écrire une nouvelle clé, hors cette clé est déjà occupée.

35 MELANGE PRIMITIVE DB ET CH INTERDIT
Cette erreur arrive lors de l''appel de la primitive ECRDB et que la dernière primitive appelée pour ce membre était un ECRCH

36 MAJ, ON CHANGE DE CASE ET LE BUFF N'EST PAS PLEIN
Cette erreur arrive lors de l'appel de la primitive ECRCH pour mettre à jour un enregistrement et le buffer traité avec les primitives précédentes n'est pas plein donc le buffer n'a pas été sauvegardé.

37 DESCRIPTEUR DE SEGMENT VIDE
Cette erreur arrive lors de l'appel de la primitive OUVRE, le descripteur de segment passé par l'utilisateur ne comporte aucun membre.

38 ENREGISTREMENT DEJA INVALIDE
Cette erreur arrive lors de l'appel de la primitive INVCH. L'utilisateur demande l'invalidation d'une clé déjà invalidée.

39 MAXDBRD ERRONE
Le plus grand numéro d'enregistrement possible dans la zone débordement est négatif, nul ou plus petit que le nombre d'enregistrements déclaré au CRMB. Voir la maintenance, il doit y avoir une erreur dans le module MODES.

40 MISE A JOUR SUR UNE CLE INVALIDE
L'utilisateur demande une écriture en mise à jour avec les primitives ECRDB ou ECRCH ou ECRDC et la clé n'existe pas encore.

41 TAILLE ANCIEN ET NOUVEAU FICHIER INCOMPATIBLE
Lors de l'exécution de la primitive COPYFIC la taille donnée au nouveau fichier est inférieure à la taille de l'ancien fichier

42 MEMBRE DEJA EXISTANT
Cette erreur arrive lors d'un CRMB. Le membre que l'utilisateur veut créer existe déjà dans le decripteur de

segment fourni pour le ranger.

43 CLE INVALIDE

La clé calculée par le module MODES pour écrire dans le fichier est négative nulle ou plus grande que la taille du fichier. Voir la maintenance.

44 ERREUR MODE DE TRAIT. ASEQ, ADBL, ADCH

L'utilisateur appelle une primitive lecture ou écriture pour un membre qui n'est pas dans le mode d'accès prévu pour cette primitive. Par exemple dans le cas d'accès séquentiel les primitives sont ECRSQ et LIRSQ les autres primitives si elles sont utilisées provoqueront l'erreur 44

45 IL FAUT LIRE L'ENREGISTREMENT POUR LE METTRE A JOUR

Cette erreur arrive dans le cas d'utilisation de la primitive ECRDC en mise à jour, la dernière primitive exécutée sur ce membre n'est pas une lecture. Hors la mise à jour se fait sur la dernière clé que l'on vient de lire.

46 MELANGE DE PRIMITIVE ECRSQ ET LIRSQ INTERDIT

L'utilisateur tente de lire ou d'écrire un membre en organisation séquentielle alors que les primitives n'étaient pas de la même catégorie. Après avoir écrit, un membre séquentiel pour le relire il ne faut oublier de mettre le booléen BDEB à TRUE.

500 ZONE DYNAMIQUE PASCAL SATUREE

Lors d'un NEW pour demander de la mémoire en dynamique, on s'aperçoit que la zone dynamique du compilateur est saturée. Voir l'administrateur du site pour résoudre le problème.

6. SPECIFICATIONS DE REALISATION DES PROCEDURES

6.1. Procédures de Services

```
PROCEDURE CNSTCHID(NUM:INTEGER);
```

Cette procédure appelée par CRMB lors de la réservation d'une case :

- Réserve un enregistrement de type PINDX
- Y stocke le numéro de case NUM
- Chaîne l'enregistrement en queue de la chaîne pointée par TINDX

A la fin de la réservation des cases par CRMB, nous aurons une chaîne d'enregistrements, contenant chacun un numéro de case, dans l'ordre de la réservation. Cette chaîne permettra de constituer la table index du membre en cours de création.

```
PROCEDURE MAJTIND(CLE,NCASE:INTEGER;P:PGES);
```

Cette procédure permet de mettre à jour la table index lors de l'allongement d'un membre en accès direct. Elle range le numéro (NCASE) dans la case qui vient d'être réservée en fin de la table index existante.

```
Recherche dernière case table index
Recherche emplacement libre dans dernière case
Si dernière case pleine alors
   Réservation d'une nouvelle case
   Chainage de celle-ci en fin table index
Stockage numéro (NCASE) dan premier emplacement libre
```

```
PROCEDURE CALCCAS(P:PGES;VAR NCASE:INTEGER;VAR DEPL:INTEGER;
                  CLE:INTEGER);
```

Cette procédure détermine l'emplacement de la clé dans le membre, pour cela elle rend deux valeurs :

- NCASE, numéro de case relatif dans le membre
- DEPL, déplacement de la clé dans la case

Dans le cas d'une case en zone débordement, NCASE = NCASE + MAXMB

```
PROCEDURE CALNOCAS(P:PGES;DBRD:BOOLEAN);
```

Cette procédure calcule le numéro de case réelle dans le fichier utilisateur, à l'aide du numéro de case relatif dans le

membre et de la table index de celui-ci. DBRD true indique que l'on travaille sur la zone débordement d'un membre en accès direct chaîné.

```
Calcul index élément table index
Ajustement index si zone débordement
Si index hors case alors
   Lecture des cases suivantes
Récupération numéro case Réelle
Repositionnement sur début table index
```

. PROCEDURE DECHNENR(VAR NENR:INTEGER;P:PGES);

Cette procédure rend le numéro du premier enregistrement libre ou bien l'enregistrement de numéro NENR si l'utilisateur a choisi sa clé et le dechaîne de la chaîne des enregistrements libres pour le mettre dans celle des occupés. Dans le cas d'accès direct chaîné, la procédure est toujours appelée avec NENR égal à zéro

```
Si clé non fixée par l'utilisateur alors
   NENR = première clé chaîne libre
   Si chaîne libre vide alors
      Prolongation du membre
      Mise à jour table index
Correction "chaînage avant" enregistrement précédent
Lecture case contenant nouvel enregistrement si nécessaire
NO1 = numéro enregistrement suivant
"chaînage avant" = fin chaîne (0 mode 2, NBENREG+1 mode 3 )
Mise à jour "chaînage arrière"
Mise à jour pointeur début chaîne libre  Pà.ENRLIB
Si écriture dans un membre vide alors
   Mise à jour pointeur début chaîne occupée  Pà.ENROCC
Corretion "chaînage arrière" premier enregistrement libre
Rechargement pointeur courant si nécessaire
```

PROCEDURE CHNENR(P:PGES);

Cette procédure déchaîne l'enregistrement pointé par Pà.NUMCLE de la chaîne des enregistrements occupés et le rechaîne en-tête de la chaîne des enregistrements libres.

```
Mise à jour pointeur début chaîne libre  Pà.ENRLIB
NO1 = numéro enregistrement suivant
NO2 = numéro enregistrement précédent
Correction "chaînage avant" enregistrement précédent
Correction "chaînage arrière" enregistrement suivant
Si début de chaîne alors
   Correction pointeur début chaîne des enrtegistrements occupés
    Pà.ENROCC
```

Correction "chaînage avant" et "chaînage arrière" enregistrement libéré

```
PROCEDURE DECHNCAS(VAR NCASE:INTEGER);
```

Cette procédure rend le numéro de la première case de la chaîne des enregistrements libres et la déchaîne. Le premier enregistrement du fichier est mis à jour.

- pointeur début de la chaîne des libres
- nombre de cases occupées

```
PROCEDURE CHNCAS(CLE:INTEGER);
```

Cette procédure rechaîne la case de numéro CLE en-tête de la chaîne des enregistrements libres. Le premier enregistrement du fichier est mis à jour.

- pointeur début chaîne libres
- nombre de cases occupées

```
PROCEDURE TRANSFE(IDEB:INTEGER;ADBUF:PBUFF;P:PGES;
                  LGBUF:INTEGER);
```

Cette procédure transfère le buffer pointé par ADBUF dans le buffer courant du fichier PASCAL.

Si accès direct alors
  Saut des deux mots de chaînage
Si écriture par morceau alors
  Mise à jour de l'index de parcours du buffer IXENR
Transfert des données dans buffer courant du fichier

```
PROCEDURE TRANSFL(IDEB:INTEGER;ADBUF:PBUFF;P:PGES;
                  LBUF:INTEGER);
```

Cette procédure transfère les données se trouvant dans le buffer courant Pà.ENR du fichier PASCAL dans le buffer utilisateur pointé par ADBUF

Si accès direct alors
  Saut des deux mots de chaînage
Si écriture par morceau alors
  Mise à jour de l'index de parcours du buffer IXENR
Transfert des données dans le buffer utilisateur

```
PROCEDURE IINVART(FBLOC,ULGENREG;INTEGER;VAR NUM:INTEGER;
                  VAR NO:INTEGER;B:BOOLEAN);
```

Cette procédure initialise les chaînages dans une case réservée pour un membre en accès direct. Tous les chaînage avants sont mis négatifs, afin de marquer les enregistrements comme libres.

6.2. Primitives d'initialisation

```
PROCEDURE DEF INITES(VAR NOMFICH:NOMFIC;CUT;INTEGER;
                     MODE:CHAR);
```

Cette procédure effectue la connexion dynamique du fichier physique de nom NOMFICH à la variable fichier PASCAL d'index CUT. Quatre fichiers sont prédéfinis dans MODULECO, selon la valeur du CUT:

1 FICHIERS
2 FICHIERU
3 FICHIERV
4 FICHCOM

```
Connexion dynamique du fichier utilisateur
Si mode lecture alors
   Ouverture en lecture
sinon
   Ouverture en Mise à jour
Lecture du premier enregistrement du fichier
Récupération pointeur début chaîne des enregistrements libres
Récupération taille du fichier
```

```
PROCEDURE DEF INITMODES;
```

Cette procédure appelée par le programme principal lors du lancement d'une session MODULECO, s'occupe de l'initialisation de toutes les tables nécessaires aux Entrées-Sorties.

```
Réservation d'un buffer de travail
Mise à Nil de la table de gestion des Entrées-Sorties
Initialisation des booléens de contrôle connexion
Initialisation table de correspondance fichiers permanents et
  temporaires
```

```
PROCEDURE DEF INIREPRI;
```

Cette procédure s'occupe de la réinitialisation des tables d'Entrées-Sorties après une interruption Utilisateur par la

frappe d'un Break sur sa console.

Mise à nil de la table de gestion des Entrées-Sorties
Initialisation des booléens de contrôle des connexions
Mise à jour de l'annuaire de base REPGEST
Réinitialisation de la table des segments ouverts

```
PROCEDURE DEF INITSYS(P:PDESCRSG);
```

Cette procédure enregistre les numéros d'enregistrements début des deux membres du descripteur P, unique segment du fichier SYSFIC.
Cette procédure est appelée uniquement par le système lors d'une génération de MODULECO sur un nouveau site.

```
PROCEDURE DEF DEBSYS(VAR NOMFICH:NOMFIC;
                     VAR DBMB1,DBMB2:INTEGER);
```

Cette procédure récupère les numéros d'enregistrements début des deux membres du segment contenu dans le fichier SYSFIC.
Ces deux valeurs sont rendues dans les variables DBMB1 et DBMB2. Cette procédure est appelée lors de chaque début de session par le gestionnaire des objets MODULECO

6.3. Primitive de formattage de fichier

```
PROCEDURE DEF IFIC(VAR NOMFICH:NOMFIC;TAILLE:INTEGER);
```

Cette procédure effectue le formattage du fichier utilisateur, en chaînant tous les enregistrements dans une chaîne des enregistrements vides.

Mise à zéro du buffer de travail ES.ENREG
Connexion dynamique du fichier à formatter
Ouverture en Mise à jour
Ecriture de tous les enregistrements en les chaînant
Stockage dans premier enregistrement
  Début chaîne enregistrement libre
  Taille du fichier
  Nombre de cases occupées
Fermeture du fichier

6.4. Primitives de gestion de fichier

```
FUNCTION DEF FMAXMB(CODE:INTEGER;NOMMB:NMMBR):INTEGER;
```

Cette fonction rend le nombre d'enregistrements qu'il est possible d'utiliser actuellement dans le membre de nom NOMMB. Pour l'accès direct chaîné, c'est le nombre d'enregistrements dans la zone principale.

```
PROCEDURE DEF FNBCASE(CODE:INTEGER;NOMMB:NMMBR;
                      VAR NBCASE:INTEGER,VAR NBCLE:INTEGER);
```

Cette procédure rend le nombre de cases réservées pour le membre de nom NOMMB.

```
Recherche du membre concerné
Calcul nombre de cases occupées par les données NBCASE
Si mode séquentiel alors
   Ajustement du nombre de cases
Si mode accès direct alors
   Ajustement du nombre de cases
   Calcul et ajout nombre case table index
Si mode accès direct chaîné alors
   Calcul nombre de cases zone débordement
   Calcul nombre de cases table index débordement
```

```
PROCEDURE DEF TAUXOCC;
```

Cette procédure imprime sur la console le nombre de cases du fichier de l'utilisateur et le taux d'occupation de celui-ci.

```
FUNCTION DEF ETATFICH(CUT:INTEGER):BOOLEAN;
```

Cette fonction teste l'état du fichier permanent désigné par le cut

```
ETATFICH = TRUE, le fichier permanent est bon
           FALSE, le fichier permanent est mauvais
```

```
PROCEDURE DEF COPYFIC;
```

Cette procédure sert dans le cas de fichier utilisateur plein. Elle formatte le nouveau fichier, puis recopie l'ancien dans le nouveau.

```
Demande nom du nouveau fichier
Demande la taille du fichier
```

```
Formattage du nouveau fichier IFIC
Demande nom de l'ancien fichier
Connexion dynamique de l'ancien fichier en lecture
Connexion dynamique du nouveau fichier en écriture
Correction lère case
   Taille du fichier
Recherche dernière case chaîne des libres
Copie des cases 2 à N de l'ancien fichier dans le nouveau
Correction chaînage dernière case libre
```

6.5. Primitives de gestion des segments

PROCEDURE DEF COPIE(PENT,PSORT:PDESCRSG;FICENT:INTEGER);

Cette procédure permet de copier le segment pointé par le descripteur PENT dans le segment pointé par PSORT. Tous les membres du segment PENT sont recopiés l'un après l'autre. FICENT est le CUT du fichier d'origine, car un utilisateur peut copier le segment d'un autre utilisateur. Le CUT du fichier de sortie est inutile, c'est celui du fichier utilisateur. Par définition dans MODULECO, il est enterdit d'écrire chez le voisin.

```
Ouverture du segment origine en lecture
Ouverture du segment destinataire en écriture
Pour tous les membres faire
   Accès séquentiel
      Copie des données du membre
   Accès direct bloqué
      Mise à jour des pointeurs dans le premier enregistrement
       du membre
         Début chaîne des enregistrements libres
         Début chaîne des enregistrements occupés
         Numéro dernière clé utilisée
      Copie des données du membre
   Accès direct chaîné
      Ajustement de la zone débordement ALONGEDC
      Mise à jour des pointeurs dans le premier enregistrement
       du membre
         Début chaîne des enregistements libres
         Début chaîne des enregistrements occupés
         Numéro dernière clé utilisée
      Copie des données de la zone principale du membre
      Copie des données de la zone débordement du membre
Fermeture des segments
```

```
PROCEDURE DEF CRMB(UNOMMB:NMMBR;UMODE:MOD1;ULGENREG:INTEGER;
                   NBENREG:INTEGER;PTR:PDESCRSG;CUTP:INTEGER);
```

Cette procédure permet de réserver un membre :

- accès séquentiel. La procédure réserve une case du fichier utilisateur.
- accès direct bloqué. La procédure réserve N/Facteur de blocage + 1 cases du fichier utilisateur.
- accès direct chaîné. La procédure réserve N/Facteur de blocage + 1 cases du fichier utilisateur pour la zone principale et 2 cases pour la zone débordement.

```
Recherche d'un membre libre dans le descripteur PTR
Stockage longueur enregistrement
Si accès séquentiel alors
   Réservation d'une case DECHNCAS
   Stockage numéro case début membre
   Mise mode d'accès à 1
Si accès direct bloqué ou chaîné alors
   Si accès direct bloqué alors
      Mode d'accès = 2
   sinon
      Mode d'accès = 3
      Ajout de 1 au nombre d'enregistrements
   Calcul nombre de cases à réserver
   Réservation des cases
   Initialisation chaînage avant et arrière des  enregistrements
   Stockage numéro case début membre
   Constitution de la table index CNSTCHID
   Initialisation des pointeurs dans premier enregistrement
   Si accès direct chaîné alors
      Réservation case article zone débordement
      Initialisation chaînage avant et arrière
      Réservation case table index
      Initialisation table index
```

```
PROCEDURE DETMB(P:PDESCRSG;I:INTEGER);
```

Cette procédure détruit le membre de numéro I pointé par le descripteur de segment P.

```
Mise à blanc nom du membre
Pour toute la chaîne de cases faîre
   Rendre case CHNCAS
Si accès direct chaîné alors
   Pour toute la chaîne des cases zone débordement faîre
      Rendre case CHNCAS
```

```
PROCEDURE DEF DETSEG(NOMSEG:NMSEG;CUTP:INTEGER);
```

Cette procédure détruit tous les membres d'un segment. C'est à dire, rechaîne à la liste des cases libres du fichier utilisateur toutes celles occupées par les différents membres du segment. Elle remet à blanc le nom de chaque membre.

```
Pour tous les membres du segment faire
   Destruction du membre DETMB
```

```
PROCEDURE DEF DETMBR(NOMMB:NMMBR;P:PDESCRSG;CUTP:INTEGER);
```

Cette procédure détruit le membre de nom NOMMB du segment pointé par P, à l'aide de la porcédure DETMB.

```
PROCEDURE DEF OUVRE(NOMSEG:NMSEG;MODOUV:CHAR;
                    VAR CODE:INTEGER;CUTP:INTEGER);
```

Cette procédure initialise une entrée de la table de gestion des Entrées-Sorties, afin de permettre la lecture ou l'écriture des différents membres du segment à l'aide des primitives d'écriture ou de lecture.

```
Recherche d'une case libre dans la table de gestion des
  Entrées-Sorties
Pour tous les membres faire
   Réservation d'un buffer d'Entrées-Sorties
   Mise à zéro du buffer
   Récupération des renseignements du descripteur
      Numéro case début du membre
      Mode d'accès
      Longueur enregistrement
   Dernière macro utilisé = 0
   Si accès direct bloqué ou chaîné alors
      Récupération des pointeurs premier enregistrement
         Dernière clé réservée MAXMB
         Début zone débordement FDEBORD
         Début chaîne des libres ENRLIB
         Début chaîne des occupées ENROCC
         Numéro dernière clé utilisée DRNCLE
         Stockage table index PTIND
   Si accès direct chaîné alors
      Stockage table index débordement PDEBRD
Calcul du CODE rendu à l'utilisateur
```

```
PROCEDURE DEF FERME(PTR:PDESCRSG;CODE:INTEGER);
```

Cette procédure permet de fermer un segment, c'est à dire, d'écrire le buffer courant, si nécessaire et nettoyer la table de gestion des Entrées-Sorties de tous les renseignements concernant les membres.

```
Pour tous les membres faire
   Si dernière utilisation en écriture alors
      Ecrire dernier buffer Pà.ENR
   Si membre ouvert enécriture alors
      Si accès direct chaîné alors
         Sauvegarde table index zone débordement
      Si accès direct bloqué ou chaîné alors
         Sauvegarde des pointeurs dans premier enregistrement
            Numéro dernière clé réservée
            Tête chaîne des enregistrements libres
            Tête chaîne des enregistements occupés
            Numéro dernière clé utilisée
            Si accès direct chaîné alors
               Pointeur zone débordement
Si accès direct chaîné alors
   Rendre buffer table index zone débordement
Si accès direct chaîné ou bloqué alors
   Rendre table index
Rendre buffer
```

```
FUNCTION DEF TSTFERME:BOOLEAN;
```

Cette fonction teste s'il y a encore des segments ouverts dans la table (TES) de gestion des Entrées-Sorties. Cette procédure est appelée par la procédure de sauvegarde, car il faut que tous les segments soient fermées, lors de la recopie du fichier temporaire dans le fichier permanent.

```
TSTFERME = TRUE plus de segment ouvert
           FALSE il reste des segments ouverts
```

6.6. Primitives d'Ecriture

```
PROCEDURE DEF ECRSQ(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                    BDEB:BOOLEAN);
```

Cette procédure permet d'écrire un enregistrement dans un membre en accès séquentiel.

```
Si BDEB true ou début traitement membre alors
   Initialisation des pointeurs
sinon
   Si changement de case alors
      Réservation d'une nouvelle case DECHNCAS
      Ecriture case pleine
   Mise à jour dernier numéro clé utilisée MAXMB
Transfert des données dans buffer case TRANSFE
Dernière macro = E
```

```
PROCEDURE DEF ECRDB(CODE;INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                    VAR CLE:INTEGER);
```

Cette procédure permet d'écrire un enregistrement dans un membre en accès direct bloqué.

```
Réservation d'un enregistrement DECHNENR
Transfert des données dans buffer case TRANSFE
Dernière macro = E
```

```
PROCEDURE DEF ECRCH(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                    VAR CLE:INTEGER;LGBUF:INTEGER);
```

Cette procédure permet d'écrire un enregistrement dans un membre en accès direct bloqué par morceau.

```
Si création d'un enregistrement alors
   Si nouvel enregistrement alors
      Réservation d'un enregistrement DECHNENR
sinon
   Si nouvel enregistrement alors
      Réservation d'un enregistrement DECHNENR
Transfert des données dans buffer case
Dernière macro = E
```

```
PROCEDURE DEF ECRDC(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                    CLE:INTEGER;BNCLE:BOOLEAN);
```

Cette procédure permet d'écrire un enregistrement dans un membre en accès direct chaîné.

```
Si Mise à jour alors
   Transfert des données dans le buffer de la case
   Dernière macro = E
sinon
   Si changement de case alors
      Ecriture de l'ancienne case
      Lecture de la nouvelle case
   Si cle déjà occupée alors
      Si premier débordement pour la clé alors
         Réservation enregistrement DECHNENR
      sinon
         Recherche fin de chaîne zone débordement pour cette clé
         Réservation enregistrement DECHNENR
   Transfert des données dans le buffer de la case
   Dernière macro = E
```

6.7. Primitives de lecture

```
FUNCTION DEF LIRSQ(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                   BDEB:BOOLEAN):INTEGER;
```

Cette fonction permet de lire un enregistrement dans un membre en accès séquentiel.

```
Si dernière macro £ O et £ L alors
   Si BDEB alors
      Ecrire le dernier buffer utilisé
Si BDEB ou début du membre alors
   Initialisation pointeur
   Lecture case
   Si fin de membre
      Erreur = 10
sinon
   Progression numéro de clé
   Si changement case alors
      Lecture case suivante
Transfert des données dans le buffer utilisateur
Dernière macro = L
```

```
FUNCTION LECTDIR(CODE:INTEGER;P:PGES;CLE:INTEGER;ADBUF:PBUFF;
                 LGBUF:INTEGER):INTEGER;
```

```
Si clé en dehors du membre alors
   Erreur = 18
Si lecture séquentielle alors
   Si début membre alors
      Si membre vide alors
         Erreur = 10
         Exit
      Initialisation des pointeurs
   sinon
      Si fin de buffer pour morceau ou lecture normale alors
         Si dernière macro £ I alors
            Numéro clé suivante dans chaînage
            Si clé nulle alors
               Erreur = 10
               Exit
Si dernière macro = E alors
   Ecriture dernière case utilisée
Si changement de case alors
   Lecture nouvelle case
Si enregistrement non occupé alors
   Erreur = 18
   Exit
Transfert des données dans buffer utilisateur
```

Dernière macro = L

```
FUNCTION DEF LIRDB(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                   CLE:INTEGER):INTEGER;
```

Cette fonction permet de lire un enregistrement dans un membre en accès direct bloqué.

Appel LECIDIR pour réaliser la lecture

```
FUNCTION DEF LIRCH(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                   CLE:INTEGER;LGBUF:INTEGER):INTEGER;
```

Cette fonction permet de lire un enregistrement dans un membre en accès direct boqué par morceau.

Appel LECTDIR pour réaliser la lecture

```
FUNCTION DEF LIRDC(CODE:INTEGER;NOMMB:NMMBR;ADBUF:PBUFF;
                   CLE:INTEGER;VAR FCHN:BOOLEAN):INTEGER;
```

Cette fonction permet de lire un enregistrement dans un membre en accès direct chaîné.

```
Si lecture séquentielle alors
   Si fin de chaîne alors
      FCHN = true
   sinon
      Clé suivante dans chaîne des enregistrements
      Si changement de case alors
         Ecriture dernière case utilisée
      Lecture case concernée
      Transfert des données dans buffer utilisateur
sinon
   Si dernière utilisation était en écriture alors
      Ecriture dernière clé utilisée
   Si changement de case alors
      Lecture de la case concernée
   Si clé invalide alors
      Erreur = 18
      Exit
   Transfert des données dans le buffer utilisateur
Dernière macro = L
```

6.8. Primitives d'invalidation

```
PROCEDURE INVART(CODE,MODE:INTEGER;NOMMB:NMMBR);

 Si accès direct bloqué ou clé en zone secondaire en accès direct
   chaîné alors
    Remise enregistrement chaîne libre CHNENR
 sinon
    Si suite en zone secondaire alors
       Appel CHNENR pour mise à jour pointeurs
    sinon
       Mise clé libre
 Dernière macro = I
```

```
PROCEDURE DEF INVDB(CODE:INTEGER;NOMMB:NMMBR);
```

Cette procédure permet d'invalider le dernier article lu dans le cas d'accès direct bloqué.

```
 Appel procédure INVART pour invalidation
```

```
PROCEDURE DEF INVDC(CODE:INTEGER;NOMMB:NMMBR);
```

Cette procédure permet d'invalider le dernier article lu dans le cas d'accès direct chaîné.

```
 Appel procédure INVART pour invalidation
```

```
PROCEDURE DEF INVCH(CODE:INTEGER;NOMMB:INTEGER);
```

Cette procédure permet d'invalider le dernier article lu dans le cas d'accès direct bloqué, avec traitement par morceau. Il y a invalidation de l'enregistrement, mais il reste dans la chaîne des occupés.

```
PROCEDURE INVCHB(CODE:INTEGER;NOMMB:NMMBR;NBENR:INTEGER);
```

Cette procédure permet d'invalider NBENR articles à partir du dernier article lu dans le cas d'accès direct bloqué, traitement par morceau.

```
 Tant que NBENR > 0 faire
    On enlève invalidation
    Remise enregistrement dans chaîne libre CHNENR
    NBENR = NBENR - 1
```

BIBLIOGRAPHIE

<1> D.W.BARRON
Pascal. The Language and its Implementation
JOHN WILEY & SONS. 1982

<2> CII-HONEYWELL-BULL
Système de gestion de fichiers
SGF SIRIS8 version C10
Manuel d'utilisation et d'opérations 56F2 7166 REV 1

<3> CROCUS
Système d'exploitation des Ordinateurs
Dunod, Paris (1975)

<4> A.DRUD
A Survey of Model Representation and Simulation Algorithms in Some Existing Modeling systems, Journal of Economic Dynamics and Control
(à paraitre)

<5> R.FORTIER
Conception descendante de machine informatique
Thèse 3 ème cycle, Université de Grenoble, Octobre 1975

<6> J.GUERRIER
Transport du système MENTOR
Mémoire CNAM, Octobre 1980

<7> IBM
OS/VS2 MVS
Data Management
Services Guide GC26-3 875-1

<8> S.KRAKOWIAK
Cours Systèmes d'exploitation des Ordinateurs
Université de Grenoble. 1982

<9> B.OUDET et G.RUDERMAN
Etude comparative des Logiciels disponibles dans
"Méthodes Mathématiques de la modélisation macroéconomique"
Malgrange P. Editeur
Collection Les Synthèses du Sesori. Mai 1979

<10> BRUCE W.RAVENEL
Toward a Pascal Standard
Language Resources,Inc, April 1979

<11> G.MICHAEL SCHNEIDER
Pascal : an overwiew
University of minnesota, April 1979

<12> B.SEFSAF
Conception et réalisation du système de gestion de séries chronologiques du logiciel MODULECO
Thèse 3 ème cycle Université de Grenoble, Novembre 1981

Notes internes MODULECO

<13> R.GARDIEN et P.LESPINASSE
Edition, Création et Appel de Commande
MODULECO-IMAG, réf B12, Janvier 1981

<14> R.GARDIEN et F.RECHENMANN
Manuel de référence du langage Moduléco
MODULECO-IMAG, réf B11, Février 1981

<15> R.GARDIEN et N.VITRY
Spécifications techniques des utilitaires systèmes
MODULECO-IMAG, réf B14, Aôut 1981

<16> R.GARDIEN
Spécifications de Réalisation du langage MODULECO
MODULECO-CSL, réf B15, Aôut 1982

<17> H.JAYET
Manuel de référence Simulation
MODULECO-INSEE, réf B05, Juin 1980

<18> P.LESPINASSE
Gestionnaire des objets MODULECO
MODULECO-IMAG, réf B02, Juin 1981

<19> P.NEPOMIASTCHY
Langage Externe d'Ecriture des Equations
MODULECO-INRIA, réf C01, Juin 1979

<20> P.NEPOMIASTCHY
Manuel de référence du système d'optimisation
MODULECO-INRIA, réf B10, Février 1981

<21> F.RECHENMANN
Le sous-système d'estimation
MODULECO-IMAG, réf A11, Novembre 1979

<22> F.RECHENMANN
Système de modélisation
1ère partie: Organisation des objets rattachés aux modèles

2ème partie: Structures de données
MODULECO-IMAG, réf B04, Mai 1980

<23> F.RECHENMANN
Manuel de référence, édition et manipulation de modèles
MODULECO-IMAG, réf B07, Juillet 1980

<24> B.SEFSAF
Spécifications techniques de la base de données MODULECO
et des commandes associées
MODULECO-IMAG, réf B03, Mai 1980

<25> B.SEFSAF
Manuel de référence base de données
MODULECO-IMAG, réf B06, Juillet 1980