Gestion mémoire : JEVEUX

Résumé :

Ce document présente les différentes fonctions d'une bibliothèque logicielle appelée JEVEUX permettant de créer, utiliser, décharger sur disque et détruire des objets nommés.

Cette bibliothèque résout en particulier le problème de l'allocation dynamique (interdite en Fortran 77) ainsi que celui de la persistance des objets sur disque en vue d'une « reprise » du calcul.

Cette bibliothèque est à la base de la structuration des données dans Code_Aster.

Signalons que JEVEUX n'est pas le seul moyen de s'allouer de la mémoire, mais tout autre allocation dynamique doit être effectuée avec précaution et de préférence avec les deux macros mises à disposition as_allocate et as_deallocate.

Table des matières

1 Présentation générale

1.1 Introduction

La gestion des zones de mémoire affectées à chaque structure de données constitue une part importante de l'organisation d'une application de calcul scientifique rédigée en langage Fortran.

La norme Fortran 77 propose un nombre très réduit de possibilités d'implantation de ces structures de données :

- variable simple,
- tableau de dimension fixe,
- zone commune de dimension fixe partagée entre plusieurs unités de programme.

Ces possibilités sont insuffisantes pour réaliser une gestion dynamique des ressources.

Par ailleurs, pour toute application conduisant à des quantités de données d'un volume supérieur à celui de la mémoire disponible sur une machine donnée, le rédacteur de l'application doit prendre en charge les débordements mémoire et programmer tous les échanges entre la mémoire centrale et la mémoire auxiliaire (fichiers).

Le progiciel JEVEUX permet de prendre en charge, directement dans le texte source Fortran plusieurs niveaux d'opérations :

- la normalisation des types Fortran utilisables sur des machines différentes (entier, réel, complexe, logique, caractère). Les critères de portabilité sont ceux définis avec le progiciel ENVIMA (Manuel de Développement - Fascicule [D6.01.01] : le descripteur d'environnement machine),
- la création de structures de données élémentaires (objets JEVEUX) de taille définie à l'exécution, partageables entre plusieurs unités de programmes et auto documentées,
- la prise en charge de tous les transferts entre mémoire centrale et mémoire auxiliaire.

Remarques :

R1 : Ce document présente l'ensemble des fonctions du progiciel Jeveux *. Un certain nombre de ces fonctions n'a pas à être connues du programmeur "lambda" (celles qui ne doivent être appelées que par le superviseur, cf. § 4.21 Les routines d'initialisation utilisées par le superviseur.). Il faut également remarquer que la grande majorité des utilisations de* Jeveux *dans Aster concerne la gestion d'objets "simples" (des vecteurs). Le lecteur pressé (ou effrayé par l'ensemble de ce document) pourra se convaincre que la gestion d'un vecteur par* Jeveux *est très simple en regardant l'exemple du §5.1 Création et réutilisation d'un vecteur de réel.*
R2 : Le développeur du Code_Aster qui utilise Jeveux *doit (en plus de ce document avoir pris connaissance d'un autre document : "Usage de Jeveux"* [D2.06.01].

1.2 Les objets JEVEUX : objet simple et collection

1.2.1 Les concepts de base

Objet JEVEUX :

Le progiciel JEVEUX permet de gérer deux types de structures de données accessibles par nom au sein de l'application. L'usage a voulu que l'on appelle ces structures de données objets JEVEUX. Par définition, un objet JEVEUX est :

- soit un objet simple,
- soit une collection d'objets.

Un objet JEVEUX est l'ensemble constitué par un descripteur et un ou plusieurs segment(s) de valeurs.

Descripteur :

Un descripteur est un ensemble d'informations constitué d'un nom et de différents attributs. Le descripteur permet d'accéder, à partir du nom, au(x) segment(s) de valeurs de l'objet JEVEUX. Les attributs décrivent la structure de l'objet JEVEUX.

Nom :

Un nom est constitué d'une suite de caractères alphanumériques limitée à 24 caractères (au nom est associé, par une table et une fonction de codage, un entier permettant d'accéder à l'objet par adressage associatif).

Attribut :

Les attributs sont les paramètres (en nombre fixe pour les objets simples) permettant de définir et de décrire la structure de données (variable, tableau). Par exemple le type des valeurs et la longueur du tableau font partie de l'ensemble des attributs.

Segment de valeurs :

Un segment de valeurs est une suite continue de mots ou d'octets (suivant l'unité d'adressage de la machine) utilisée pour stocker des valeurs en mémoire centrale ou en mémoire auxiliaire (fichier).

Remarque :

Sauf exception (les répertoires de noms), un segment de valeurs est une suite de valeurs de même type.

1.2.2 L'objet simple

L'objet simple est la structure de données de base du progiciel. Un objet simple est constitué du descripteur et d'un seul segment de valeurs. Le nombre d'attributs d'un descripteur d'objet simple est fixé pour une version du progiciel.

Remarque :

Le nombre d'attributs est identique pour tous les objets simples.

```
Objet simple = Nom + Attributs + 1 Segment de valeurs
```

Le nom d'un objet simple est constitué de 24 caractères pris parmi les suivants :

- lettres majuscules de A à Z, les lettres minuscules de a à z et chiffres de 0 à 9 ;
- quatre caractères spéciaux :

le blanc	« »
le point	« . »
le blanc souligné	« _ »
le et commercial	« & »

- le caractère spécial $ est licite mais non accessible à l'utilisateur.

Les principaux attributs sont les suivants :

- le genre : décrit la structure de l'objet simple

E	élément simple (variable)
V	vecteur (tableau à un indice)
N	répertoire de noms (ce genre sera défini dans le paragraphe suivant)

Le genre permet d'associer le segment de valeurs aux concepts classiques de variable, de tableau Fortran ou de définir une structure de données plus complexe.

- le type : définit le type Fortran des variables scalaires de l'objet

I	entier	(type Fortran INTEGER)
S	entier	(type Fortran INTEGER*4)
R	réel	(type Fortran REAL*8)
C	complexe	(type Fortran COMPLEX*16)
L	logique	(type Fortran LOGICAL)
K8, K16, K24, K32, ou K80	caractères	(type Fortran CHARACTER)

- la longueur : définit la longueur du segment de valeurs associé (le nombre de scalaires).

1.2.3 La collection

Collection :

Une collection est une structure de données permettant de mettre en commun certains attributs de l'ensemble des objets la composant. Elle autorise indifféremment l'accès par nom ou par numéro aux objets de collection et de gérer des objets de longueur variable.

Deux types de collection sont possibles :

- l'une comportant un segment de valeurs par objet de collection : la collection dispersée,
- l'autre un seul segment de valeurs pour tous les objets de la collection : la collection contiguë.

Remarque :

Une collection ne peut être construite a posteriori par regroupement d'objets simples.

Une collection est réalisée à partir du descripteur de collection et d'un ou plusieurs segments de valeurs.

Descripteur de collection :

Le descripteur de collection est l'ensemble des informations définissant la collection : le nom et les attributs.

Les attributs communs à tous les objets de la collection sont notamment : le genre, le type, la longueur si elle est constante, etc....

Les attributs spécifiques à chaque objet de collection sont gérés séparément. Du point de vue de l'utilisateur, un objet de collection possède tous les attributs d'un objet simple, il pourra être utilisé de la même manière.

```
Collection = Nom + Attributs + 1 ou plusieurs Segments de valeurs
```

Le nom d'une collection est constitué de manière identique à celui d'un objet simple.

Les attributs spécifiques d'une collection sont :

- l'accès :

qui définit le mode d'accès aux objets de la collection : l'accès peut être réalisé par nom (la collection est alors dite nommée), par numéro (la collection est dite numérotée),

- le stockage :

qui décrit l'organisation en mémoire des valeurs associées aux objets de la collection, les valeurs peuvent être contiguës (un seul segment de valeurs), ou bien dispersées (un segment de valeurs par objet de collection). Tous les objets d'une collection contiguë se suivent dans le segment de valeurs associé,

- la longueur :

elle est constante si tous les objets de la collection partagent la même longueur, elle est variable si les objets sont de longueurs différentes,

- le nombre maximum d'objets de la collection.

Deux attributs de collection peuvent être partagés entre plusieurs collections. Dans le cas d'une collection d'objets de longueur variable, le vecteur des longueurs peut être réutilisé pour une autre collection contenant le même nombre d'objets. De même, deux collections peuvent partager les noms des objets de collection. Ces objets sont appelés pointeurs externes.

Types de collection :

La collection numérotée est composée d'objets dont la clef d'accès est un entier variant de 1 au nombre maximum d'objets de la collection. Dans une collection numérotée tous les objets préexistent.
La collection nommée utilise un objet simple particulier de genre répertoire de noms qui contient la liste des noms d'objets de la collection gérée par adressage associatif. Ce répertoire de noms peut être défini par l'utilisateur ou créé automatiquement. Les noms des objets sont insérés dans l'ordre chronologique de création des noms. On peut accéder à un objet de collection nommée par son nom et/ou par le numéro d'ordre d'insertion.

2 Gestion des objets

2.1 Les attributs

2.1.1 Généralités

Les attributs auxquels l'utilisateur a accès sont, à deux exceptions près, non modifiables directement.

Ils sont affectés lors de la création du descripteur (par exemple le type) puis figés jusqu'à destruction de ce descripteur. Le système gère en plus des attributs qui évoluent au cours de l'application (par exemple l'adresse mémoire du segment de valeurs).

On distingue trois catégories d'attributs pour les objets JEVEUX :

- les attributs génériques : classe, genre, type, ...
- les attributs de collection.
- les attributs des objets de collection.

Tous les attributs sont consultables à tout moment par l'utilisateur. Dans la suite du document, pour chaque attribut, on indique :

- le nom symbolique par lequel on peut le consulter,
- le type Fortran de chaque attribut avec les conventions suivantes :

Code	Type	Déclaration Fortran
I	entier	INTEGER
S	entier	INTEGER*4
R	réel	REAL*8
C	complexe	COMPLEX*16
L	logique	LOGICAL
Ki	caractère	CHARACTER*i
K*	chaîne	CHARACTER*(*)

2.1.2 Les attributs génériques

Attributs affectés par l'utilisateur (non modifiables après création du descripteur) :

CLAS	K1	classe de rattachement de l'objet à une base de données
GENR	K1	genre de l'objet : - E variable simple - V vecteur, - N répertoire de noms de type K8, K16 ou K24.
TYPE	K1	type Fortran de l'objet : I,R,C,L ou K
LTYP	I	longueur du type : géré automatiquement pour les types I, R, C et L, - normalisé pour les caractères aux valeurs 8, 16, 24, 32 et 80
longueur de l'objet : LONMAX NOMMAX	I I	nombre de composantes de l'objet de genre V nombre de noms maximum de l'objet de genre N

Attributs modifiables à tout moment par l'utilisateur :

Ces attributs ne sont pas indispensables au fonctionnement du progiciel. Ils permettent à l'utilisateur de compléter la description des objets JEVEUX qu'il manipule.

LONUTI	I	nombre de composantes de l'objet de genre V utilisées
DOCU	K4	quatre caractères libres laissés à l'utilisateur

Attributs affectés et gérés par le système et consultables par l'utilisateur :

Ces attributs sont décrits à titre d'information, il peuvent être consultés lors de la mise au point d'une application (recherche d'erreur, ...).

NOMUTI	I	nombre de noms de l'objet de genre N effectivement utilisés
DATE	I	entier comportant le mois, le jour, l'année, l'heure et les minutes de l'exécution du travail ayant effectué la dernière modification du segment de valeurs (dernier déchargement sur la base de données)

IADM	I	adresse mémoire : position relative dans la mémoire JEVEUX du segment de valeurs et adresse du pointeur vers la zone mémoire allouée dynamiquement (stockés dans 2 entiers pour chaque objet)
IADD	I	adresse disque : contient le numéro de l'enregistrement du segment de valeurs dans la base de données et la position relative dans l'enregistrement (stockés dans 2 entiers pour chaque objet)
LONO	I	longueur en unité d'adressage de l'objet (place occupée par le segment de valeurs mesurée en longueur du type)
USAGE	K16	usage d'un objet JEVEUX : contient à la fois une information sur l'usage en écriture ou en lecture, l'état (déchargeable, amovible, ...) du segment de valeurs lorsqu'il est présent en mémoire et les marques déposées lors de la première requête en lecture et en écriture

Ces derniers ne sont utilisés que de façon interne à JEVEUX.

2.1.3 Les attributs de collection

Attributs communs affectés par l'utilisateur ou gérés par le système :

Les attributs communs (CLAS, GENR, TYPE, LTYP, DOCU, etc.) aux objets de collection sont accessibles de la même façon que les attributs d'un objet simple et répondent aux mêmes règles.

Attribut affecté et géré par le système et consultable par l'utilisateur :

NUTIOC	I	nombre d'objets effectivement créés dans la collection

Remarque :

Cet attribut est actualisé uniquement lors de l'appel de la routine de création d'objet de collection JECROC .

Attributs affectés par l'utilisateur et non modifiables :

ACCES	K*	type d'accès : NOMME ou NUMEROTE (voir l'affectation de cet attribut par JECREC)
STOCKAGE	K*	Mode de stockage des valeurs : - CONTIG : tous les objets sont contigus dans le segment de valeurs - DISPERSE : les objets sont archivés au fur et à mesure des besoins
MODELONG	K*	Mode de définition de la longueur des objets de collection : - CONSTANT : tous les objets sont de même longueur (attribut de longueur de l'objet de référence) - VARIABLE : chaque objet peut avoir une longueur différente
LONT	I	longueur totale d'une collection contiguë
NMAXOC	I	nombre maximum d'objets de la collection

2.1.4 Les attributs des objets de collection

Les règles de nom d'attribut des objets de collection sont les mêmes que pour les objets simples.

2.2 Les bases de données associées

A chaque classe il est possible d'associer une base de données sur disque. Cette possibilité n'est nullement obligatoire : on peut travailler sans écriture et lecture sur disque (ce qui peut conduire à saturer la mémoire disponible et à l'arrêt de l'application).

Le nombre de classes sur lequel on peut travailler simultanément est limité à 5. Il est possible d'ouvrir ou de fermer une classe à tout moment au cours de l'application.

L'ouverture d'une base de données préexistante permet de récupérer l'ensemble des informations stockées sur cette dernière. En fin de travail, il est possible de détruire une classe entière.

Une base de données est un fichier d'accès direct ; elle est définie par :

- un nom de classe (`K1`),
- un nom local du fichier (`K8`),
- une longueur de bloc,
- un nombre de blocs.

Remarque :

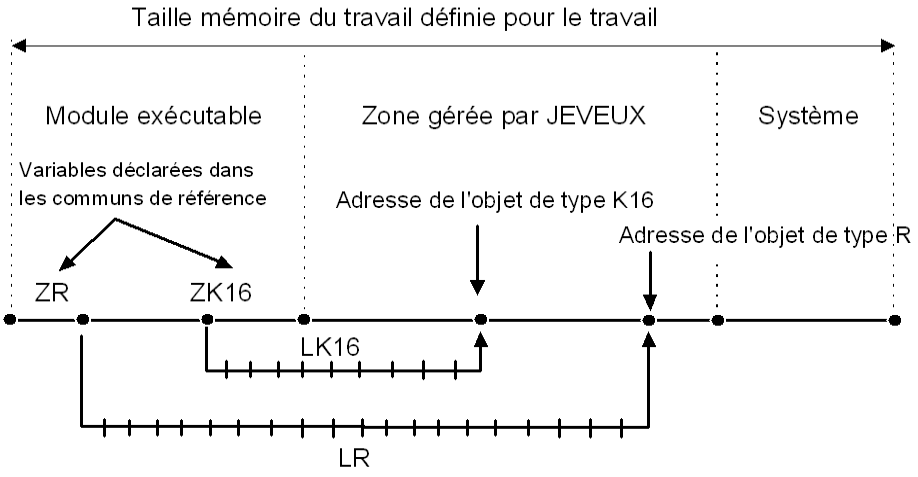
Dans le Code_Aster, toutes les classes sont associées à des bases cf § 3.3 L'environnement du Code_Aster.

2.3 L'accès aux valeurs

L'utilisateur accède aux valeurs d'un objet `JEVEUX` par une adresse relative dans un tableau défini par une des variables Fortran de référence du type convenable. Ces variables de référence (`ZI`, `ZR`, ...) sont placées dans des communs normalisés confer [D2.06.01] : Usage `Jeveux` "Libération des segments de valeurs et notion de marque".

Remarque :

L'adresse renvoyée reste valide tant que l'utilisateur n'a pas déclaré qu'il n'utilisait plus l'objet.



- `ZR (LR)` 1 ère composante des valeurs de l'objet de type réel
- `ZK16(LK16 - 1 + I)` i ème composante des valeurs de l'objet `K16`

2.4 Notions de gestion de la mémoire

L'accès aux objets `JEVEUX` s'effectue par nom. Ces noms sont stockés dans un objet simple de genre répertoire de noms accessible uniquement par le progiciel et géré par une méthode d'adressage associatif.

Pour ordonner une application, l'utilisateur doit regrouper les objets JEVEUX dans une ou plusieurs classes ouvertes au préalable. A chaque classe est associé un catalogue : c'est l'ensemble constitué du répertoire de noms et des attributs des objets JEVEUX de la classe. La classe est l'un des attributs défini à la création du descripteur d'un objet JEVEUX. La recherche du nom s'effectue parmi toutes les classes ouvertes à cet instant

Remarque :

Un nom ne peut donc figurer plus d'une fois parmi l'ensemble des classes ouvertes.

A chaque classe, il est possible d'associer un fichier d'accès direct (ou base de données), qui contiendra en fin d'application, les descripteurs et les segments de valeurs de tous les objets JEVEUX de la classe. Ce type de fichier permet d'accéder rapidement aux différents enregistrements, un index décrivant la position de chacun d'entre eux.

Les échanges entre la mémoire centrale et les bases de données associées aux classes sont entièrement pris en charge par le progiciel. Quand la mémoire centrale est saturée (qu'il n'est plus possible de s'allouer une zone dynamiquement), celui-ci décharge ou détruits les segments de valeurs déclarés inutilisés. En fin d'application, tous les objets JEVEUX présents en mémoire sont déchargés dans la base de données associée, ainsi que les catalogues (objets simples « système »), ce qui permet les réutilisations ultérieures.

Le fichier d'accès direct ne sera valide que si l'index a été actualisé lors de la fermeture, il est donc indispensable d'arrêter proprement l'application en passant par la routine JEFINI. Tout segment de valeurs en mémoire est encadré par huit entiers (quatre devant, quatre derrière) permettant de gérer le segment de valeurs, d'indiquer son usage (libre, utilisé en lecture, déchargeable, ...), de stocker l'identificateur associé, et d'assurer une protection partielle aux débordements (le progiciel contrôle l'intégrité des valeurs contenues par ces entiers lors de toute requête).

2.5 Les requêtes d'utilisation des objets

Le progiciel JEVEUX fournit différents sous-programmes et fonctions permettant de gérer tous les objets. On distingue les différentes requêtes suivantes :

- la création du descripteur d'un objet JEVEUX ou d'un objet de collection,
- l'allocation de l'objet : recherche de place en mémoire pour le segment de valeurs associé, initialisation des valeurs ou relecture sur la base de données, enfin fourniture à l'unité de programme appelante d'une adresse relative du segment de valeurs par rapport à une variable de référence Z* . Cette allocation peut être formulée en lecture seulement ou en lecture/écriture ;
- la consultation du descripteur qui permet de récupérer dynamiquement la valeur d'un attribut ;
- les impressions du descripteur, du segment de valeurs, du catalogue ou de l'état de la mémoire gérée par JEVEUX ;
- la libération qui met fin à l'allocation de l'objet et rend le ou les segment de valeurs inutilisés ;
- la destruction du descripteur et du segment de valeurs d'un objet JEVEUX ou d'un objet de collection.

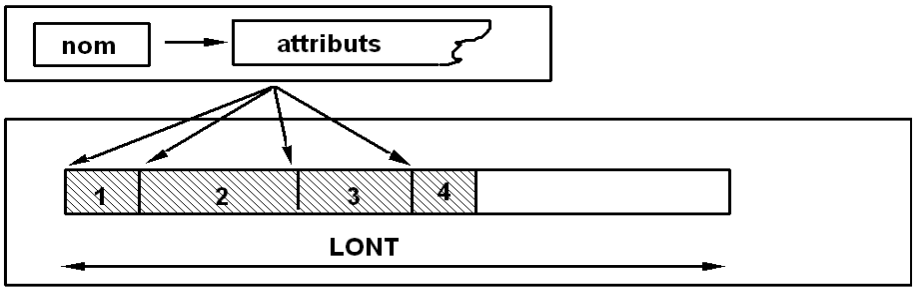
2.6 Les particularités des collections

Le dimensionnement d'une collection dépend de son type. Pour chaque type, on décrit ci dessous le mode de définition du ou des segment(s) de valeurs associés.

2.6.1 Collection contiguë de longueur variable

On définit :

- soit la longueur totale du segment de valeurs par l'attribut LONT (longueur totale) de la collection. On peut dans ce cas créer la collection et définir (sans ordre particulier) ultérieurement la longueur de chaque objet avant de l'utiliser ;
- soit la longueur de tout ou partie des objets de la collection en donnant le vecteur des longueurs géré par l'utilisateur, ou en actualisant l'attribut de longueur pour chaque objet, la longueur totale du segment de valeurs sera calculée par le progiciel.



collection contiguë de longueur variable

Remarques :

Il est conseillé de définir la longueur de tous les objets d'une collection contiguë avant la première requête d'allocation en mémoire,
La longueur du segment de valeurs LONT est figée lors de la première requête d'allocation en mémoire,
Une collection contiguë de longueur variable ne peut donc être agrandie après le premier accès, tous les objets créés dans une collection contiguë sont gérés ensemble,
Le segment de valeurs associé peut être utilisé comme un vecteur de valeurs en ignorant le découpage en objets de collection.

2.6.2 Collection contiguë de longueur constante

On définit la longueur commune à tous les objets de collection ; la longueur du segment de valeurs sera égale au produit de la longueur constante par le nombre maximum d'objets de la collection.

Attention : lorsque l'on fait le JEECRA du LONMAX de la collection, il faut indiquer la longueur d'un seul élément de la collection et non pas la longueur totale.

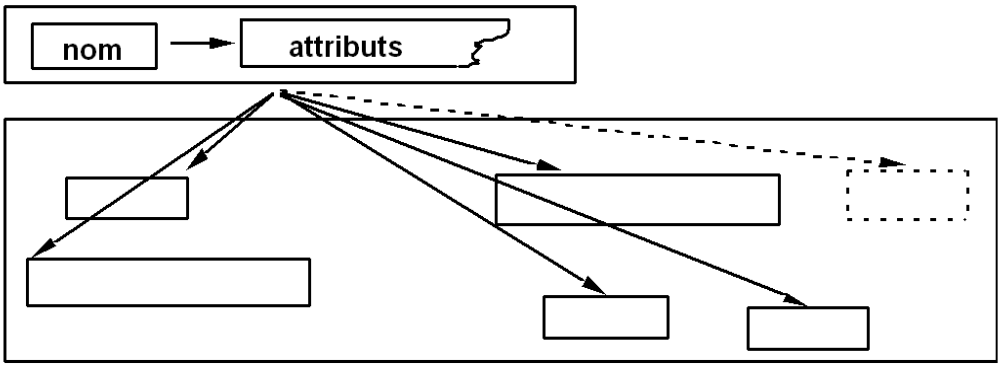
2.6.3 Collection dispersée de longueur variable

On définit :

- le majorant du nombre d'objets,
- la longueur de chaque objet, au fur et à mesure des besoins et sans ordre particulier, avant l'allocation de l'objet.

Remarque :

L'encombrement effectif de ce type de collection est limité à la longueur cumulée des objets effectivement créés.



collection dispersée de longueur variable

Remarque :

Pour une collection dispersée, on peut utiliser un objet (dont la longueur a été définie) avant d'avoir fini de définir l'ensemble des objets.

2.6.4 Collection dispersée de longueur constante

On définit :

- le majorant du nombre d'objets,
- la longueur commune à tous les objets.

2.7 La libération des segments de valeurs

Une requête d'accès en lecture ou en écriture sur un segment de valeurs associé à un objet JEVEUX provoque un chargement en mémoire du contenu du segment de valeurs associé. Lorsque l'on gère un espace mémoire fini, il arrive un moment où il n'est plus possible de trouver de place pour charger un nouvel objet. Il faut alors provoquer des déchargements sur disque ou bien éliminer des segments de valeurs devenus inutiles. Ce mécanisme ne peut être totalement pris en charge par JEVEUX : le programmeur doit avoir au préalable indiqué les objets concernés. Mais certaines précautions doivent être observées : plusieurs unités de programme peuvent utiliser simultanément une même adresse mémoire associée à un objet. La mise en mémoire d'un segment de valeurs s'accompagne de l'affectation d'une marque entière qui mesure le niveau de profondeur dans les appels de sous-programmes pour chaque nouvelle requête. La libération ne peut s'effectuer que si le niveau d'appel à cet instant est identique à la marque associée au segment de valeurs. On appelle marque courante le niveau d'appel en cours. La mise en œuvre de ce mécanisme impose une règle d'usage de JEVEUX très stricte : toute routine qui appelle JEVEUO ou WKVECT doit effectuer un appel à JEMARQ comme première instruction exécutable et un appel à JEDEMA comme dernière instruction exécutable. La routine JEMARQ permet d'actualiser la valeur de la marque courante, en l'incrémentant de 1, affectée à tous les segments de valeurs chargés ultérieurement. La routine JEDEMA libère tous les segments de valeurs associés à la marque courante puis décrémente cette dernière.
Certaines configurations tolèrent des exceptions à cette règle : il s'agit par exemple de boucles sur des blocs de matrices. Il est alors nécessaire de libérer au fur et à mesure les objets, c'est la routine JELIBE qui est utilisée dans ce cas.

Il est parfois nécessaire de disposer en permanence ou tout du long d'une commande du Code_Aster de certains objets (pour le Superviseur d'exécution, le matériau codé), des requêtes spécifiques sont utilisées, qui affectent une marque spéciale.

La valeur de la marque courante peut être consultée à tout instant à l'aide de la routine JEVEMA.

3 Environnement d'application

3.1 Les fichiers d'impression

L'impression des descripteurs, des segments de valeurs, du catalogue, de l'état de la mémoire gérée par JEVEUX et des messages d'erreur est dirigée vers des unités logiques définies en début d'application (cf. §3.3 L'environnement du Code_Aster).

3.2 L'aide à la mise au point

Les erreurs rencontrées et détectées par le progiciel provoquent l'impression d'un message d'erreur, et dans certains cas ferment les bases de données ouvertes par l'application. Le progiciel imprime alors l'état de la mémoire qu'il gère et les catalogues de toutes les classes ouvertes.

Remarque :

JEVEUX utilise ses propres routines d'impression de messages différentes de celles du Code_Aster pour éviter des appels dynamiquement récursifs.

Tous les objets JEVEUX, sauvegardés au préalable, sont récupérables dans une exécution ultérieure.

3.3 L'environnement de Code_Aster

Deux bases de données sont utilisables au sein de l'application de Code_Aster :

base 'GLOBALE'	associée à la classe 'G'
base 'VOLATILE'	associée à la classe 'V'

Remarque :

Une troisième base de nom 'BASEELEM' est utilisée pour stocker et relire le catalogue compilé des éléments.

Les impressions de messages d'erreur sont réalisées par les utilitaires d'impression de message avec les conventions suivantes :

- erreur de programmation (mauvaise utilisation de JEVEUX) : message de classe 'S' avec arrêt immédiat ;
- erreur d'exploitation (accès à un objet inexistant, …) : message de classe 'F' avec tentative de clôture des bases de données ouvertes.

Les utilitaires d'impression de messages d'erreur de Code_Aster peuvent parfois être employés afin de communiquer avec le Superviseur et d'arrêter proprement le code en validant les concepts créés. Les noms d'unités d'impression indiqués lors de l'appel des routines d'impression (JEIMPO, JEIMPR, …) sont les suivants :

- RESULTAT : résultats du calcul ,
- MESSAGE : messages de conduite et messages d'erreur.

4 Sous-programmes fournis et utilisation

4.1 Préliminaires

Dans ce chapitre, on désigne par :

```
nom_os    un nom d'objet simple
nom_co    un nom de collection
nom_oc    un nom d'objet de collection
num_oc    un numéro d'objet de collection
```

Lors de la description des arguments des routines, on précise le type de chaque argument et s'il doit être fourni en entrée (in) ou récupéré en sortie (out). Les arguments modifiés sont signalés par (var).

4.2 Les fonctions d'accès pour les objets de collection

L'accès aux objets JEVEUX et aux objets de collection s'effectue par nom. Dans le cas d'une collection nommée, il est nécessaire de passer, comme argument, un des deux groupes d'informations :

- le nom de la collection et le nom de l'objet de collection,
- le nom de collection et le numéro d'ordre de l'objet de collection dans le répertoire.

Pour limiter le nombre de routines utilisateur et uniformiser les arguments d'appel, on a introduit des fonctions de type Fortran CHARACTER *32. Celles-ci renvoient une chaîne de caractères composée du nom de collection et d'un suffixe interprétable par le progiciel. Elles affectent un commun qui assure le transfert vers la routine appelée du nom d'objet de collection, du numéro d'ordre dans le répertoire. Afin de bien synchroniser l'appel avec le nom de collection et l'affectation du commun, il est obligatoire d'appeler ces fonctions dans les arguments des routines concernées.

Remarque :
Ces fonctions ne doivent être utilisées que lors de l'appel d'une routine de requête sur un objet.

Fonctions de type CHARACTER*32 (à déclarer dans tout sous-programme d'appel) :

Les trois fonctions suivantes doivent être appelées uniquement comme argument des routines agissant sur des objets de collection

CHARACTER*32 FUNCTION JEXNOM (nom,nom_oc)

in	nom	K2	nom de répertoire ou de collection
in	nom_oc	K*	nom d'objet de collection.

CHARACTER*32 FUNCTION JEXNUM (nom,num_oc)

in	nom	K24	nom de répertoire ou de collection
in	num_ocv	I	numéro d'objet de collection.

Exemple d'utilisation :

```
CALL JEVEUO(JEXNUM(NOMCO,NUMO),'L',vx=TABL)
```

Dans le cas d'une collection contiguë, une fonction de type Fortran CHARACTER*32 est utilisée pour accéder au vecteur des longueurs cumulées.
Ce vecteur d'entiers contient $n+1$ valeurs pour une collection de n objets : la composante V_i de ce vecteur fournit l'adresse relative de l'objet i dans le segment de valeurs de la collection ; la longueur de cet objet est obtenue par la différence $V_{i+1}-V_i$.

CHARACTER*32 FUNCTION JEXATR (nom_co,arg)

in	nom_co	K24	nom de collection
in	arg	K8	'LONCUM'

			'LONUTI', 'LONMAX'

De la même façon, l'argument LONUTI, respectivement LONMAX, permet d'accéder au vecteur des longueurs utilisées, respectivement maximum, par une simple requête du type :
CALL JEVEUO(JEXATR(nom_co,'LONUTI'),'L',vi=LUTI).
Notation : on désignera, à partir de maintenant, un nom d'objet JEVEUX, ou un appel à l'une de ces fonctions au sein des arguments d'une routine par nom_o.

4.3 La création des descripteurs

Le descripteur d'un objet simple ou d'une collection est créé en général par plusieurs appels. Le premier appel permet de créer le descripteur en définissant le nom de l'objet JEVEUX, et en indiquant les valeurs des attributs obligatoires (non modifiables ultérieurement) :

- la classe de rattachement,
- le genre,
- le type.

Dans le cas d'une collection, on doit définir en plus :

- l'accès,
- le stockage,
- le mode de définition de la longueur,
- le nombre maximum d'objets.

4.3.1 Création du descripteur d'objet simple

SUBROUTINE JECREO (nom_os,lis_at)

in	nom_os	K24	nom d'objet simple.
in	lis_at	K*	texte définissant CLAS GENR TYPE et si besoin LTYP, lorsqu'elle est fournie, la longueur du type doit être collée au type.

Par exemple : 'G V K16'. Le premier caractère doit être non blanc, au moins un blanc sépare chaque valeur d'attribut ; les majuscules sont obligatoires.

4.3.2 Création du descripteur de collection

SUBROUTINE JECREC (nom_co,lis_at,acces,stock,modlon,nmaxoc)

in	nom_co	K24	nom de collection
in	lis_at	K*	texte définissant CLAS GENR TYPE et si besoin LTYP, lorsqu'elle est fournie, la longueur du type doit être collée au type.
in	acces	K*	'NO' : collection nommée avec répertoire interne (noms de 8 caractères au maximum), objets de collection créés par nom et accessibles ensuite par nom ou par numéro. 'NU' : collection numérotée séquentiellement, les objets de collection sont accessibles uniquement par numéro.
in	stock	K*	mode de stockage de la collection : 'CONTIG' objets de collection contigus dans le segment de valeurs, 'DISPERSE' objets de collection indépendants les uns des autres.
in	modlon	K*	mode de définition de la longueur des objets de collection

			'CONSTANT' : tous les objets de la collection sont de longueur identique ; 'VARIABLE' : chaque objet de la collection peut avoir une longueur différente.
in	nmaxoc	I	nombre maximum d'objets de la collection.

Remarques :

Il n'est plus autorisé de créer des collections s'appuyant sur un répertoire de noms créé par ailleurs et pouvant être partagé.
Il n'est plus autorisé de créer des collections s'appuyant sur un vecteur de longueur créé par ailleurs et pouvant être partagé.

4.3.3 Affectation d'un attribut (après création du descripteur)

Il est souvent nécessaire de compléter l'affectation des attributs du descripteur (définition de la longueur d'un vecteur, de la longueur totale d'une collection , ...).

SUBROUTINE JEECRA (nom_o,nom_at,ival=ival,cval=cval)

in	nom_o	K24	nom d'objet JEVEUX
in	nom_at	K*	nom d'attribut (cf. annexe 2 : liste des attributs).
in	ival	I	valeur entière pour un attribut de type entier.
in	cval	K*	texte pour un attribut de type CHARACTER.

Remarque :

Les deux arguments ival et cval sont facultatifs, cval doit être différent de la chaîne vide. Les deux arguments peuvent être fournis lors de l'appel à la routine avec ou sans l'identificateur respectif « ival= » et «cval= » .

```
CALL JEECRA (nom_o, 'DOCU', cval='JPJP')
CALL JEECRA (nom_o, 'LONMAX', ival=500)
```

4.4 Insertion d'un nom dans un répertoire ou création d'un objet de collection

C'est la même routine qui est utilisée pour créer un nom d'objet de collection et pour insérer un nom dans un objet simple de genre répertoire.

SUBROUTINE JECROC (JEXNOM(nom_o,nom))

in	nom_o	K24	nom d'objet simple de genre répertoire, ou nom de collection nommée
in	nom	K*	nom d'objet à insérer dans le répertoire, ou nom d'objet de collection

SUBROUTINE JECROC (JEXNUM(nom_co,num_oc))

in	nom_co	K24	nom de collection
in	num_oc	I	numéro d'objet de collection

Remarque :

Pour une collection numérotée, cet appel actualise l'attribut NUTIOC .

4.5 La requête d'allocation

C'est par l'intermédiaire de la routine JEVEUO que l'utilisateur récupère un pointeur sur le segment de valeurs. Alternativement, la routine JEVEUO permet également de récupérer une adresse relative par rapport à une variable Z* mesurée dans la longueur du type de l'objet JEVEUX.
Cela lui permet ensuite d'utiliser le segment de valeurs associé. Lors de cet appel si le segment de valeurs n'est pas présent en mémoire, le progiciel effectue une allocation dynamique. Si l'objet JEVEUX n'a pas d'image sur disque, le segment de valeurs est initialisé suivant le type de l'objet (zéro ou blanc). Dans le cas contraire on récupère sur disque les valeurs précédentes. On appelle usage la condition d'accès ('E' ou 'L') au segment de valeurs. Le segment de valeurs reste alloué avec l'usage affecté tant que l'utilisateur n'a pas effectué de nouvelle requête et tant que l'appel à JEDEMA au bon niveau n'a pas été effectué. Une requête en écriture sur un objet alloué en lecture en modifiera l'usage. Une requête en lecture sur un objet alloué en écriture n'affectera pas l'usage.

SUBROUTINE JEVEUO (nom_o,cel,jtab,vi,vr,…,vk80)

in	nom_o	K24	accès au segment de valeurs défini par : nom_os : objet simple, nom_co : collection contiguë, JEXNOM (nom_co, nom_oc) : objet de collection nommée, JEXNUM (nom_co, num_oc) : objet de collection numérotée, JEXATR (nom_co, 'LONCUM') vecteur des longueurs cumulées de collection contiguë.
in	cel	K*1	condition d'accès ou usage du segment de valeurs, 'E' en lecture/écriture (permet de modifier le segment de valeurs), 'L' en lecture (pas de modification du segment de valeurs).
out	jtab	I	adresse de la première valeur de l'objet dans le tableau Fortran associé à la variable Z* correspondant au type de l'objet
out	vi	I	Tableau d'entiers
out	vr	R	Tableau de réels
...			
out	vk80	K80	Tableau de character*80

Les arguments de sortie sont tous des arguments facultatifs et ils s'excluent mutuellement.
On préconise d'utiliser préférentiellement les arguments : vl, vi, vi4, vr, vc, vk8, vk16, vk32 et vk80. Chaque argument est adapté au type de l'objet : LOGICAL, INTEGER, ...
L'argument jtab rend une « adresse » dans un tableau mis en COMMON : ZI, ZR, ... Son usage rend la programmation en général moins lisible.

Remarque :
La requête JEVEUO est interdite sur les objets de genre répertoire.

4.5.1 Utilisation des arguments vi, vr, …, vk80

Exemple :

```
character(len=24) :: obj_coor
real(kind=8), pointer :: coordo(:) => null()

call jeveuo(obj_coor, 'L', vr=coordo)
! kième valeur du vecteur coordo :
x=coordo(k)
```

4.5.2 Utilisation de l'argument `jtab`

Exemple :

```
character(len=24) :: obj_coor
integer :: jcoor

call jeveuo(obj_coor, 'L', jcoor)
! kième valeur du vecteur coordo :
x=zr(jcoor-1+k)
```

COMMON de référence

La requête d'allocation JEVEUO avec l'argument de sortie jtab retourne l'"adresse" de l'objet dans un tableau mis en COMMON : ZI, ZR, ...

La correspondance entre le type fortran de l'objet et le COMMON de référence qui lui correspond est donnée par le tableau suivant :

type fortran	COMMON	nom de la variable
I	IVARJE	ZI
S	I4VAJE	ZI4
R	RVARJE	ZR
C	CVARJE	ZC
L	LVARJE	ZL
K8	KVARJE	ZK8
K16	KVARJE	ZK16
K24	KVARJE	ZK24
K32	KVARJE	ZK32
K80	KVARJE	ZK80

Le bloc de déclarations ci-dessous doit donc être inséré dans toute routine voulant lire ou écrire dans un objet jeveux :

```
C --------- DEBUT DECLARATIONS NORMALISEES  JEVEUX --------------------
      INTEGER            ZI
      COMMON  / IVARJE / ZI(1)
      INTEGER*4          ZI4
      COMMON  / I4VAJE / ZI4(1)
      REAL*8 ZR
      COMMON / RVARJE / ZR(1)
      COMPLEX*16 ZC
      COMMON / CVARJE / ZC(1)
      LOGICAL ZL
      COMMON / LVARJE / ZL(1)
      CHARACTER*8        ZK8
      CHARACTER*16               ZK16
      CHARACTER*24                        ZK24
      CHARACTER*32                                 ZK32
      CHARACTER*80                                          ZK80
      COMMON  / KVARJE / ZK8(1) , ZK16(1) , ZK24(1) , ZK32(1) , ZK80(1)
C --------- FIN  DECLARATIONS  NORMALISEES  JEVEUX --------------------
```

4.6 La requête d'allocation permanente au cours d'une commande

JEVEUT permet d'allouer des objets JEVEUX en leur affectant une marque différente de la marque courante valant -1. Les objets ne seront pas affectés par les appels à JEDEMA mais devront être libérés explicitement par un appel à JELIBZ.

La syntaxe d'appel est identique à celle de JEVEUO.

4.7 La requête d'allocation permanente au cours de l'exécution

Cette requête est exclusivement réservée au Superviseur. JEVEUS permet d'allouer des objets JEVEUX en leur affectant une marque différente de la marque courante valant -3. Ils ne seront libérés qu'en fin d'exécution.

La syntaxe d'appel est identique à celle de JEVEUO.

4.8 Surcouche de création et allocation d'un objet de genre vecteur

WKVECT est une surcouche JEVEUX permettant d'allouer un vecteur (création et présence en mémoire), elle permet « l'économie » de l'appel à trois sous-programmes JEVEUX.

SUBROUTINE WKVECT (nom_os,lis_at,long,jtab,vi,vr,…,vk80)

in	nom_os	K24	nom d'objet simple
in	lis_at	K*	texte définissant CLAS GENR TYPE LTYP, GENR vaut ici obligatoirement V.
in	long	I	valeur entière associée à l'attribut LONMAX, longueur du vecteur.
out	jtab	I	adresse de la première valeur de l'objet dans le tableau Fortran associé à la variable Z* correspondant au type du vecteur.
out	vi	I	Tableau d'entiers
out	vr	R	Tableau de réels
...			
out	vk80	K80	Tableau de character*80

L'allocation de l'objet est effectuée par défaut en tant que premier écrivain (cel='E').

Remarques :

Les arguments de sortie : jtab, vi, vr, …, vk80 s'excluent mutuellement.
Aucun objet de nom nom_os ne doit exister dans les bases (sous peine d'arrêt du programme).

Exemple d'utilisation :

Allocation sur la base VOLATILE (V) d'un vecteur de travail (V) de REAL*8 (R) de longueur 100 ; le nom du tableau est &&OP000.TAMPON.

```
real(kind=8), pointer :: VTRAV => null()

CALL WKVECT ('&&OP000.TAMPON', 'V V R', 100, vr=VTRAV)
```

Lorsqu'il n'est pas nécessaire de disposer d'un accès nommé, pour allouer un vecteur de travail utilisé au sein d'une routine, il est possible de faire appel aux macros **as_allocate** et **as_deallocate**.

4.9 Alternative à l'allocation d'objets de travail

Bien qu'il soit possible d'allouer des objets de travail à l'aide de JEVEUX au sein d'une routine, cet usage n'est pas toujours pertinent. Si les zones allouées ne sont utilisée que temporairement, ou bien de taille ne nécessitant pas de déchargement sur disque, il est plus avantageux de faire appel aux fonction **as_allocate** et **as_deallocate** . Ces dernières enrobent les fonctions standard et permettent d'appeler les mécanismes de déchargement des objets JEVEUX afin de libérer de la mémoire. Elles communiquent à tout instant le volume de mémoire allouée ou libérée, toutes les allocations dynamiques dans le Fortran doivent impérativement passer par ce mécanisme. Ces utilitaires (macro commande) permettent d'allouer et de désallouer un vecteur des différents types gérés par JEVEUX. Ces deux macros utilisent des arguments optionnels.

AS_ALLOCATE (size=size, vl=vl, vi=vi, vi4=vi4, vr=vr, vc=vc, vk8=vk8, vk16=v16, vk24=vk24, vk32=vk32, vk80=vk80)

in	size	I	longueur du vecteur à allouer
in/out	vx	L,I,I4,C,Kx	vecteur dans le type souhaité

Cette macro permet d'allouer un vecteur, qui doit être déclaré sous le modèle ci-dessous. Les différents arguments sont tous optionnels, il est bien sûr nécessaire de fournir la longueur du vecteur.

```
integer, pointer :: tab_para(:) => null()
AS_ALLOCATE(vi=tab_para, size=1958)
```

AS_DEALLOCATE (vl=vl, vi=vi, vi4=vi4, vr=vr, vc=vc, vk8=vk8, vk16=v16, vk24=vk24, vk32=vk32, vk80=vk80)

in	vx	L,I,I4,C,Kx	vecteur dans le type souhaité

Cette macro permet de libérer le vecteur dont le nom est passé en argument et de restituer la mémoire en informant le gestionnaire JEVEUX.

4.10 Surcouche d'agrandissement d'un objet de genre vecteur

SUBROUTINE JUVECA (nom_os,long)

in	nom_os	K24	nom d'objet simple
in	long	I	nouvelle valeur associée à l'attribut LONMAX, longueur du vecteur.

Cette routine permet de modifier la taille d'un objet simple de genre vecteur. Le nouveau vecteur est affecté à la même classe que l'ancien.

Remarques :

L'objet doit être en mémoire (appel à JEVEUO préalable),
L'attribut LONUTI est affecté à la même valeur que LONMAX ,
Les valeurs sont recopiées,

4.11 Les recopies d'objets JEVEUX

4.11.1 Recopie d'un objet JEVEUX (objet simple, collection ou objet de collection) :

SUBROUTINE JEDUPO (nom_in,nom_clo,nom_ou,dup_co)

in	nom_in	K24	nom de l'objet JEVEUX à recopier
in	nom_clo	K1	nom de la classe de l'objet réceptacle (≠ ' ')
in	nom_ou	K24	
in	dup_co	L	utilisé uniquement pour les collections si les classes sont différentes, si =.TRUE. : on recopie les pointeurs externes à une collection, si =.FALSE. : on conserve les pointeurs externes

Remarque :
L'objet réceptacle est détruit s'il existe auparavant.

4.11.2 Recopie d'un ensemble d'objets JEVEUX :

Au lieu de travailler sur un seul objet, il est possible de fournir à certaines routines une chaîne de caractères servant à sélectionner les objets JEVEUX dont le nom contient cette chaîne.

SUBROUTINE JEDUPC (nom_cli,souchi,ipos,nom_clo,soucho,dup_co)

in	nom_cli	K1	nom de la classe des objets à recopier
in	souchi	K*	chaîne de caractères à identifier dans les noms d'objets
in	ipos	I	position dans les noms de la chaîne à identifier
in	nom_clo	K1	nom de la classe des objets réceptacles
in	soucho	K*	chaîne à substituer dans noms origines pour obtenir le nom de l'objet JEVEUX réceptacle
in	dup_co	L	utilisé uniquement pour les collections si les classes sont différentes, si =.TRUE. : on recopie les pointeurs externes à une collection, si =.FALSE. : on conserve les pointeurs externes

4.12 Les requêtes de libération

Ces requêtes conduisent à une écriture sur disque (différée ou non) suivant la condition d'accès choisie lors de la requête d'allocation et les règles de libération [§2.3] :

On appelle libération l'arrêt de la requête en cours sur un objet JEVEUX :

- fin d'une écriture, avec écriture différée sur disque,
- fin de lecture, qui n'implique aucune écriture sur disque.

Les libérations sont effectuées lors de l'appel à JEDEMA. Les appels à la routine JELIBE sont en règle générale prohibés.

Remarque :
L'accès en lecture permet d'éviter les écritures sur disque en fin d'usage (gain de temps).

Gestion des accès :

Les autorisations d'accès affectées lors du premier appel à JEVEUO en lecture ou en écriture sont gérées à l'aide de la routine suivante :

SUBROUTINE JEMARQ ()

JEMARQ incrémente la valeur de la marque courante. Son appel est obligatoire dans toute unité de programme chargeant en mémoire des objets JEVEUX.

Libération implicite de tous les objets chargés dans une unité de programmation

SUBROUTINE JEDEMA ()

JEDEMA libère tous les objets affectés de la marque courante et décrémente la valeur de la marque courante. Son appel est obligatoire dans toute unité de programme chargeant en mémoire des objets JEVEUX.

Libération d'un objet JEVEUX ou d'un objet de collection dispersée :

SUBROUTINE JELIBE (nom_o)

in	nom_o	K24	nom d'objet JEVEUX (objet simple, collection ou objet de collection dispersée)

Son appel est tolérée dans certaines conditions.

Libération de l'ensemble d'objets JEVEUX :

SUBROUTINE JELIBZ ()

L'appel à cette routine provoque la libération de l'ensemble des objets JEVEUX qui ont été chargés en mémoire par un appel à JEVEUT (ils sont affectés d'une marque valant -1).

4.13 Les requêtes d'existence

Les requêtes d'existence permettent de vérifier l'existence du descripteur d'un objet JEVEUX ou d'un objet de collection, mais aussi la présence d'un nom dans un répertoire. Elles permettent aussi de récupérer le numéro d'ordre d'un nom dans un répertoire de noms.

Existence d'un objet JEVEUX ou d'un objet de collection :

SUBROUTINE JEEXIN (nom_o,iret)

in	nom_o	K24	nom d'objet JEVEUX
out	iret	I	code retour de la routine $iret=0$ le descripteur de l'objet n'existe pas, $iret\neq 0$ le descripteur existe

4.14 Le passage du numéro d'ordre au nom et vice versa

Obtention du numéro d'ordre à partir du nom :

S**UBROUTINE JENONU (JEXNOM(nom_o,nom),num)**

in	nom_o	K24	nom de collection ou d'objet simple de genre répertoire
in	nom	K*	nom d'objet de collection ou nom
out	num	I	num = numéro de l'objet correspondant au nom nom

Remarques :

Si le nom cherché ne figure pas dans le répertoire, le numéro renvoyé est 0.
Cet appel est inutile dans le cas d'une collection numérotée.

Obtention du nom à partir du numéro d'ordre :

SUBROUTINE JENUNO (JEXNUM(nom_o,num),nom)

in	nom_o	K24	nom de collection ou d'objet simple de genre répertoire
in	num	I	numéro d'ordre d'insertion
out	nom	K	nom = nom de l'objet correspondant au numéro num

Remarques :

Si le numéro cherché ne figure pas dans le répertoire, le nom renvoyé est blanc.
Cet appel est impossible dans le cas d'une collection numérotée.
Si le numéro cherché est supérieur au nombre d'objets numérotés stocké, la routine s'arrête en erreur.

4.15 La destruction des descripteurs

Il est possible de détruire le descripteur d'un objet JEVEUX et par extension le nom et les attributs d'un objet de collection. Cette destruction s'accompagne de la destruction du ou des segments de valeurs associés présents en mémoire et rend inaccessibles ceux présents sur disque.

SUBROUTINE JEDETR (nom_o)

in	nom_o	K24	nom d'objet JEVEUX (objet simple, collection ou objet de collection nommée)

La place libérée dans le catalogue ou le répertoire de noms devient immédiatement réutilisable pour un autre descripteur. La place, éventuellement utilisée sur disque, ne sera récupérable que par une opération de « retassage » du fichier.

Remarques :

Il n'est pas possible de détruire un objet de collection numérotée,
pour une collection nommée, la destruction d'un objet actualise l'attribut NUTIOC .
Dans le cas d'un objet simple de type répertoire de noms, il n'est pas possible de détruire un point d'entrée dans le répertoire, l'appel à cette fonction détruit complètement l'objet JEVEUX.

On peut aussi utiliser la routine suivante pour détruire un ensemble de descripteurs.

SUBROUTINE JEDETC (nom_cl,souch,ipos)

in	nom_cl	K1	nom de la classe ou ' ' pour traiter toutes les classes ouvertes.
in	souch	K*	chaîne de caractères à identifier dans l'ensemble des noms contenus dans le répertoire d'une ou plusieurs classes.
in	ipos	I	position dans le nom de la chaîne à identifier.

On recherche tous les descripteurs dont le nom contient la sous chaîne souch à la position ipos dans les classes définies par le paramètre nom_cl et on détruit les descripteurs ainsi repérés.

Il est préférable d'utiliser la routine JEDETR lorsque l'on connaît explicitement le nom des objets à détruire, l'appel dans une boucle ou dans une routine de bas niveau peut se révéler très coûteux.

4.16 La récupération de la taille des zones de mémoire disponible

SUBROUTINE JEDISP (nbp,lplace)

in	nbp	I	nombre de positions cherchées.
in	lplace	V_I	taille en unité d'adressage des différentes zones disponibles.

JEDISP renvoie, par ordre décroissant dans le vecteur d'entiers $lplace$, la taille des nbp plus grands segments de valeurs disponibles pour une allocation (JEVEUO), à l'instant de l'appel. Les valeurs obtenues restent valides sous la condition du seul usage des requêtes d'allocation concernant des segments de valeurs de longueurs inférieures ou égales.

Ainsi $lplace(1)$ renvoie la taille du plus gros objet que l'on pourrait allouer, $lplace(2)$ celle du plus gros objet que l'on pourrait allouer après avoir utilisé la zone correspondante à $lplace(1)$.

Remarque :

L'allocation d'une collection peut entraîner le chargement en mémoire des objets attributs et rendre ainsi caduques les valeurs obtenues par JEDISP.

4.17 Les consultations

Lecture d'un attribut d'un objet JEVEUX ou d'un objet de collection

SUBROUTINE JELIRA (nom_o,nom_at,ival=ival,cval=cval)

in	nom_o	K24	nom d'objet JEVEUX
in	nom_at	K*	nom d'attribut (cf. annexe 2 : liste des attributs accessibles)
out	ival	I	valeur entière pour un attribut de type entier.
out	cval	K*	texte pour un attribut de type caractère.

Remarques :

Outre les attributs décrits au [§2], il est possible de récupérer avec la valeur XOUS le type d'objet JEVEUX associé au descripteur, « S » objet simple, « X » collection.
Les deux arguments ival et cval sont facultatifs. Les deux arguments peuvent être fournis lors de l'appel à la routine avec ou sans l'identificateur respectif « ival= » et «cval= » .

Lecture de la valeur de la marque courante

SUBROUTINE JEVEMA (marque)

out	marque	I	valeur de la marque courante

Recherche de la liste des noms de descripteurs présents dans une classe

SUBROUTINE JELSTC (nom_cl,souch,ipos,nbmax,l_nom,nbval)

in	nom_cl	K1	nom de la classe ou ' ' pour traiter toutes les classes ouvertes.
in	souch	K*	chaîne de caractères à identifier dans l'ensemble des noms contenus dans le répertoire d'une classe.
in	ipos	I	position dans le nom de la chaîne à identifier.
in	nbmax	I	dimension du vecteur de K24 fourni ci-dessous

var	l_nom	V_K24	vecteur contenant la liste des identificateurs répondant au critère de recherche
out	nbval	I	nombre maximum d'identificateurs répondant au critère de recherche $nbval = -nbval$ si $nbval > nbmax$

4.18 Les impressions

Plusieurs routines permettent d'imprimer le contenu d'un objet JEVEUX ou d'un objet de collection, d'imprimer les attributs, d'imprimer l'état de la mémoire gérée par JEVEUX (objets présents, position, taille, etc.), d'imprimer le catalogue d'une classe, ou de consulter l'état d'une base de données.

SUBROUTINE JEIMPO (unit,nom_o,param,cmess)

Impression du contenu d'un objet JEVEUX ou d'un objet de collection

in	unit	I	Numéro d'unité logique associée au fichier d'impression, au sein de Code_Aster on utilisera 6 ('MESSAGE') ou 8 ('RESULTAT')
in	nom_o	K24	nom d'objet JEVEUX
in	param	K*	non utilisé actuellement (blanc ' ' obligatoire)
in	cmess	K*	texte apparaissant en commentaire à l'impression

SUBROUTINE JEIMPA (unit,nom_o,cmess)

Impression de tous les attributs d'un objet JEVEUX ou d'un objet de collection

in	unit	I	Numéro d'unité logique associée au fichier d'impression, au sein de Code_Aster on utilisera 6 ('MESSAGE') ou 8 ('RESULTAT')
in	nom_o	K24	nom d'objet JEVEUX
in	cmess	K*	texte apparaissant en commentaire à l'impression

SUBROUTINE JEIMPR (unit,nom_cl,cmess)

Impression du catalogue

in	unit	I	Numéro d'unité logique associée au fichier d'impression, au sein de Code_Aster on utilisera 6 ('MESSAGE') ou 8 ('RESULTAT')
in	nom_cl	K1	nom de classe ou ' ' pour traiter toutes les classes ouvertes
in	cmess	K*	texte apparaissant en commentaire à l'impression

SUBROUTINE JEIMPM (unit)

Impression de l'état de la zone mémoire gérée par JEVEUX

in	unit	I	Numéro d'unité logique associée au fichier d'impression, au sein de Code_Aster on utilisera 6 ('MESSAGE') ou 8 ('RESULTAT')

La routine JEIMPM permet d'éditer le contenu de zone mémoire gérée au moment de l'appel. Voici dans l'ordre la signification des valeurs apparaissant à l'impression :

- nom en abrégé de la classe associé à l'objet,

- G base GLOBALE ,
- V base VOLATILE ,

- identificateur de collection, si ce dernier vaut 0 , c'est un objet simple,
- identificateur d'objet simple ou bien numéro d'objet de collection,
- niveau d'appel dans la pile **JEMARQ/JEDEMA**
- adresse mémoire du segment de valeur,
- usage du segment de valeur, vaut U ou X ,
- longueur en unité d'adressage (entier 8 octets en 64 bits) du segment de valeurs,
- statut du segment de valeur, vaut D ou A ,
- nom de l'objet, suivi éventuellement du numéro d'objet de collection.

Les combinaisons possibles du statut et de l'usage d'un segment de valeurs, ainsi que leur signification sont les suivantes :

- U D : segment de valeurs utilisé, en accès en écriture et éventuellement en lecture. Il devra être déchargé sur disque.
- U A : segment de valeurs utilisé, en accès en lecture. Il ne sera pas déchargé sur disque.
- X D : segment de valeurs inutilisé, déchargeable : il devra être déchargé sur disque. Une requête en écriture ou en lecture sur l'objet associé renvoie directement sa position sans mouvement mémoire et sans accès disque. Sa position peut être récupérée à tout moment pour placer un nouveau segment de valeurs au prix d'un accès disque (origine : U D).
- X A : segment de valeurs inutilisé, amovible. Une requête en écriture ou en lecture sur l'objet associé renvoie directement sa position sans mouvement mémoire et sans accès disque. Sa position peut être récupérée à tout moment pour placer un nouveau segment de valeurs sans accès disque (origine : U A).

Exemple d'impression obtenue :

```
 CL-  --NUM-- -MA-  ---------IADY--------- -U- - LON UA -  -S- ------------- NOM --------------

|G|   0|       1|  -2|           118073808|U|         11| D| ________GLOBALE ________$$CARA
|G|   0|       2|  -2|           120752752|U|       4000| D| ________GLOBALE ________$$IADD
|G|   0|       3|  -2|           118553440|U|        251| D| ________GLOBALE ________$$GENR
|G|   0|       4|  -2|           118533472|U|        251| D| ________GLOBALE ________$$TYPE
|G|   0|       5|  -2|           118559216|U|       1001| D| ________GLOBALE ________$$DOCU
|G|   0|       6|  -2|           120784832|U|       2000| D| ________GLOBALE ________$$ORIG
|G|   0|       7|  -2|           120800912|U|       8004| D| ________GLOBALE ________$$RNOM
|G|   0|       8|  -2|           120865024|U|       2000| D| ________GLOBALE ________$$LTYP
|G|   0|       9|  -2|           120881104|U|       2000| D| ________GLOBALE ________$$LONG
|G|   0|      10|  -2|           120897184|U|       2000| D| ________GLOBALE ________$$LONO
|G|   0|      11|  -2|           120913264|U|       2000| D| ________GLOBALE ________$$DATE
|G|   0|      12|  -2|           120929344|U|       2000| D| ________GLOBALE ________$$LUTI
|G|   0|      13|  -2|           120945424|U|       3203| D| ________GLOBALE ________$$HCOD
|G|   0|      14|  -2|      46912496128016|U|     188742| D| ________GLOBALE ________$$USADI
|G|   0|      15|  -2|           118578720|U|      62914| D| ________GLOBALE ________$$ACCE
|G|   0|      16|  -2|           118002208|U|       4000| D| ________GLOBALE ________$$MARQ
|G|   0|      17|  -2|           118034288|U|       2000| D| ________GLOBALE ________$$INDI
|G|   0|      18|  -2|           119082112|U|     102400| D| ________GLOBALE ________$$TLEC
|G|   0|      19|  -2|           119901392|U|     102400| D| ________GLOBALE ________$$TECR
|G|   0|      20|  -2|           120720672|U|       4000| D| ________GLOBALE ________$$IADM
|G|   0|      21|   0|            93362960|X|         21| D| &FOZERO             .PROL
|G|   0|      22|   0|            98181600|X|          2| D| &FOZERO             .VALE
|G|   0|      23|   0|           108074656|X|          1| D| &&_NUM_CONCEPT_UNIQUE
|G|   0|      24|   0|           123411680|X|       5010| D| &&SYS.KRESU
|G|   0|      25|   0|            88929024|X|         11| D| &CATA.ACOUSTIQUE
…
```

SUBROUTINE JEIMPD (unit,nom_cl,cmess)

Impression de la liste des objets JEVEUX présents dans une base de données :

in	unit	I	Numéro d'unité logique associée au fichier d'impression, au sein de Code_Aster on utilisera 6 ('MESSAGE') ou 8 ('RESULTAT')
in	nom_cl	K1	nom de classe ou ' ' pour traiter toutes les classes ouvertes
in	cmess	K*	texte apparaissant en commentaire à l'impression

SUBROUTINE JEPRAT (unit, nom, nom_at, param, cmess)

Cette routine édite le catalogue des objets JEVEUX présents sur disque.

in	unit	I	Numéro d'unité logique associée au fichier d'impression, au sein de Code_Aster on utilisera 6 ('MESSAGE') ou 8 ('RESULTAT')
in	nom	K24	nom d'objet JEVEUX ou nom de classe précédée du caractère $
in	nom_at	K8	nom de l'attribut ou suffixe de l'objet système à imprimer. nom_at prend ses valeurs parmi la liste suivante : pour une collection $$DESO, $$IADD, $$IADM, $$PEPL, $$NOM, $$REEL, $$LONG, $$LONO, $$LUTI, $$NUM. pour toute une classe $$CARA, $$IADD, $$GENR, $$TYPE, $$DOCU, $$ORIG, $$RNOM, $$LTYP, $$LONG, $$LONO, $$DATE, $$LUTI, $$HCOD, $$USADI, $$ACCE, $$MARQ, $$TLEC, $$TECR, $$IADM
in	param	K*	non utilisé actuellement (blanc ' ' obligatoire)
in	cmess	K*	texte apparaissant en commentaire à l'impression

Remarque :

Cette routine est un utilitaire réservé à l'assistance.

4.19 Les utilitaires

Les utilitaires suivants ont été écrits pour copier ou pour réinitialiser des parties d'objets simples ou des objets de collection.

Remise à zéro ou à blanc suivant le type de l'objet JEVEUX.

SUBROUTINE JERAZO (nom_o, n , i1)

in	nom_o	K24	nom d'objet JEVEUX
in	n	I	nombre de valeurs à réinitialiser
in	I1	I	indice dans le vecteur de la première valeur à réinitialiser

4.20 Un utilitaire de déverminage

Un mauvais usage de l'adresse renvoyée par la routine JEVEUO peut conduire à un écrasement mémoire, et notamment à la destruction du chaînage des segments de valeurs. Le programmeur peut dans ce cas essayer de localiser la routine provocant l'écrasement en instrumentant son code source avec des appels à JXVERI. Cette routine contrôle les entiers présents de part et d'autre de chaque segment de valeurs : lors d'un débordement ces valeurs sont en général écrasées. L'écrasement amont correspond à l'entier situé juste devant la première valeur du segment de valeurs. L'écrasement aval correspond à l'entier situé juste derrière la dernière valeur du segment de valeurs.

SUBROUTINE JXVERI (cfic,cmess)

in	cfic	K*	texte définissant le nom local du fichier d'impression, au sein de Code_Aster on utilisera 'MESSAGE' ou 'RESULTAT'
in	cmess	K*	texte apparaissant en commentaire à l'impression

Cette routine émet un message indiquant l'intégrité ou non du chaînage lors qu'il existe, ou lors de l'écrasement des identificateurs situés de part et d'autre des valeurs.

4.21 Les routines d'initialisation utilisées par le superviseur

L'utilisation d'objets JEVEUX n'est possible qu'après l'initialisation qui nécessite de définir :

- le nombre maximum de bases de données, que l'on pourra gérer simultanément,
- la taille de la zone mémoire gérée par JEVEUX, qui sera allouée dynamiquement.

Cette initialisation est obligatoirement réalisée par le superviseur au sein des commandes DEBUT ou POURSUITE à l'aide de la routine JEDEBU, qui crée, automatiquement, tous les objets système nécessaires :

SUBROUTINE JEDEBU (nbases,lzone,cmess,cvig,idebug)

in	nbases	I	nombres maximum de bases de données simultanées (≤5)
in	lzone	I	taille de la mémoire allouée en unité d'adressage
in	cmess	K*	nom local du fichier d'impression des messages d'erreur
in	cvig	K*	non utilisé
in	idebug	I	utilisé pour un fonctionnement en mode debug $idebug=0$ mode de fonctionnement normal $idebug=1$ debug JEVEUX enclenché

Lorsque l'application JEVEUX est initialisée, il importe d'ouvrir les classes d'objets sur lesquelles on souhaite travailler. Pour ouvrir une classe, il est nécessaire de préciser ses caractéristiques : présence ou non d'un fichier associé, nom de la classe, nom de la base de données associée, caractéristiques du fichier, etc … En mode DEBUG le fonctionnement du gestionnaire de mémoire est modifié, les déchargements sur disque ne sont plus différés et la place mémoire occupée par un segment de valeurs est positionné à une valeur indéfinie. Ce mode de fonctionnement est utilisé pour déverminer du code : l'usage d'une adresse correspondant à un segment de valeurs libéré provoque un arrêt brutal.

SUBROUTINE JEINIF (stin,stout,nom_bas,nom_cl,nmax,nbloc,lbloc)

in	stin	K*	texte définissant le statut en début de travail : 'DUMMY' pas de fichier associé 'DEBUT' initialisation ou remise à zéro d'une classe existante 'POURSUIT' récupération du contenu d'une classe existante
in	stout	K*	texte définissant le statut en fin de travail : 'SAUVE' sauvegarde sur fichier en fin d'application 'DETRUIT' destruction de la classe en fin d'application
in	nom_bas	K*	nom local de la base de données (exemple : 'GLOBALE', 'VOLATILE')
in	nom_cl	K1	nom de la classe associée (exemple : 'G', 'V')
in	nmax	I	nombre maximum de noms d'objets JEVEUX dans la classe
in	nbloc	I	nombre maximum d'enregistrements du fichier d'accès direct associé

in	lbloc	I	longueur des enregistrements du fichier d'accès direct associé

Remarque :

Une classe peut être ouverte ou fermée à tout moment, jusqu'à concurrence du nombre maximum de classes gérables simultanément.

Pour fermer une classe, en cours d'application :

SUBROUTINE JELIBF (cond,nom_cl)

in	cond	K*	texte permettant de surcharger la valeur de stout définie dans JEINIF 'SAUVE' avec sauvegarde immédiate sur fichier 'DETRUIT' avec destruction immédiate
in	nom_cl	K*	nom de la classe à fermer (exemple : 'G', 'V')

Enfin, il est obligatoire de faire appel à la routine de clôture de l'application qui effectue la fermeture de toutes les classes encore ouvertes après sauvegarde des objets JEVEUX et des catalogues présents en mémoire et arrête l'application par l'instruction Fortran STOP. Cette routine est appelée au sein de la commande FIN par le superviseur.

SUBROUTINE JEFINI (cond)

in	cond	K8	condition de clôture : 'NORMAL' sauvegarde suivant la valeur de stout définie dans JEINIF 'ERREUR' sauvegarde des classes ouvertes, pour éventuelle analyse ultérieure

Remarques :

C'est le seul STOP utilisable dans toute l'application pour pouvoir réutiliser les bases de données. un utilisateur doit appeler cette routine pour arrêter son application après détection d'une erreur avec cond = 'ERREUR',

Au sein de Code_Aster un arrêt avec la condition cond = 'ERREUR' n'autorise pas la mise à jour de la base GLOBALE dans le répertoire de l'utilisateur, les concepts créés n'ayant pas été validés.

SUBROUTINE JETASS (nom_cl)

in	nom_cl	K1	nom de la classe associée (exemple : 'G', 'V')

Cette routine est destinée à récupérer les enregistrements devenus inutilisés à la suite de la destruction des objets JEVEUX associés. Cela ne concerne que les gros objets, c'est à dire ceux qui nécessitent au moins un enregistrement du fichier d'accès direct. Le seul effet est de réordonner séquentiellement les enregistrements, la récupération effective de la place doit ensuite être effectuée en recopiant le début du fichier d'accès direct, seul le superviseur d'Aster peut effectuer cette action.

4.22 Les routines de sauvegarde et de relecture utilisées par le superviseur

La base GLOBALE permet de sauvegarder et de relire sur une même plate-forme les objets Jeveux obtenu lors d'une exécution. Ce fichier binaire, composé d'un ou plusieurs sous-fichiers est adhérent à la plate-forme d'exécution (système d'exploitation 32 ou 64 bits). Il est possible, à l'aide des deux routines suivantes, d'enregistrer le contenu complet au format HDF et de le relire indépendemment du type de plate-forme.

SUBROUTINE JEIMHD (fichdf,nom_cl)

in	fichdf	K*	nom local du fichier HDF à créer
in	nom_cl	K1	nom de la classe associée (exemple : 'G', 'V')

SUBROUTINE JELIHD (nom_bas,fichdf,nom_cl)

in	nom_bas	K*	nom local de la base de données (exemple : 'GLOBALE', 'VOLATILE')
in	fichdf	K*	nom local du fichier HDF à relire
in	nom_cl	K1	nom de la classe associée (exemple : 'G', 'V')

4.23 Routines d'interrogation pour le superviseur

SUBROUTINE JELIAD (nom_cl, numr, nboct)

in	nom_cl	K1	nom de la classe associée (exemple : 'G', 'V')
out	numr	I	numéro de l'enregistrement
out	nboct	I	nombre d'octets avant l'enregistrement

Cette routine renvoie le numéro de l'enregistrement contenant l'objet système $$RNOM dans la base considérée. Ce répertoire de noms étant une caractéristique de la base, le superviseur utilise cette propriété pour « identifier » la base et faire des vérifications de cohérence en poursuite.

5 Exemples d'utilisation

On supposera, dans cette partie, s'être placé dans un environnement d'application (par exemple celui d'ASTER) et avoir ouvert les bases de données associées aux classes G, V et L. On ne mentionnera que les appels aux routines JEVEUX, les déclarations et les communs Fortran sont omis.

5.1 Création et réutilisation d'un vecteur de réel

Soit le problème suivant : on veut créer (dans la routine SUBA) un vecteur de réels contenant les coordonnées X et Y de nno nœuds. On appellera ce vecteur 'COORDO_XY' et on le réutilisera dans la routine SUBB.

```
                SUBROUTINE SUBA(...)
                  ...
                 #include « jeveux.h »
                 real(kind=8), pointer :: COOR(:) => null()

                 !  -  Début des instructions :
(a)                 CALL JEMARQ( )
                 !  -  Allocation du vecteur sur la base 'GLOBALE' :
(b)                 CALL WKVECT ('COORDO_XY', 'G V R', 2*nno, vr=COOR)
                 !  -  "remplissage du vecteur"
                    DO ino = 1, nno
                          COOR( 2*(ino-1)+1) = X
                          COOR( 2*(ino-1)+2) = Y
                    ENDDO
(c)                 CALL JEDEMA ( )
                 END


                 SUBROUTINE SUBB(...)
                 ...
                 real(kind=8), pointer :: COOR(:) => null()
                 !  - Début des instructions :
(d)                 CALL JEMARQ ( )
                 !  - Récupération de l'adresse du vecteur (en lecture) :
(e)                 CALL JEVEUO ('COORDO_XY', 'L', vr=COOR)
                 !  - Récupération des coordonnées du 27ème noeud :
                    X27 =   COOR( 2 * 26 + 1)
                    Y27 =   COOR( 2 * 26 + 2)
                    ...
(f)                 CALL JEDEMA ( )
                 END
```

Commentaires :

- lignes (a), (c), (d), (e) : les routines JEMARQ/JEDEMA permettent de libérer automatiquement les objets à la fin des routines [§4.11],
- ligne (b) :
- 'G V R':
  - 'G' : base “Globale” (base de données de l'utilisateur cf. [§3.3])
  - 'V': vecteur,
  - 'R': réel,
- 2*nno : longueur du vecteur
- COOR : pointeur sur le segment de valeurs
- ligne (e)
- 'L ' : accès en “lecture” de l'objet (cf. [§4.5]).

5.2 Création d'un répertoire de noms et insertion de deux noms

```
      CALL JECREO ( 'MES_NOMS', 'G N K8' )
      CALL JEECRA ( 'MES_NOMS', 'NOMMAX', ival=25)
C
      CALL JECROC ( JEXNOM('MES_NOMS','NOM_1') )
      CALL JECROC ( JEXNOM('MES_NOMS','NOM_5') )
C
      CALL JELIRA ('MES_NOMS', 'NUTIOC', ival=IVAL)
      CALL JENONU (JEXNOM('MES_NOMS','NOM_5'),NUM)
      CALL JENUNO (JEXNUM('MES_NOMS',1),NOM)
```

L'appel à JENONU renvoie la valeur 2 dans la variable NUM, L'appel à JENUNO renvoie la valeur 'NOM_1' dans la variable NOM.

5.3 Collection dispersée de vecteurs d'entiers

Les objets de cette collection sont nommés dans le répertoire, géré par l'utilisateur et créé dans l'exemple précédent ; ils sont de longueur variable.

```
      CALL JECREC ('MA_COLL','G V I','NO','DISPERSE','VARIABLE',25)
      CALL JECROC ( JEXNOM ('MA_COLL','NOM_13') )
      CALL JEECRA ( JEXNOM ('MA_COLL','NOM_13'), 'LONMAX', ival=125)
      CALL JEECRA ( JEXNOM ('MA_COLL','NOM_1') , 'LONMAX', ival=250)
```

Dans une autre routine, on utilise l'objet 'NOM_13' qui vient d'être créé :

```
      CALL JEVEUO ( JEXNOM ('MA_COLL','NOM_13'), 'E', JTAB)
      CALL JELIRA ( JEXNOM ('MA_COLL','NOM_13'), 'LONMAX', ival=LNOM13)
C
      DO 10 K = 1, LNOM13
        ZI ( JTAB - 1 + K ) = K
10 CONTINUE
..
      CALL JEDETR (JEXNOM ('MA_COLL','NOM_13'))
```

L'attribut NUTIOC de la collection est actualisé en même temps que celui de l'objet simple 'MES_NOMS'. La dernière instruction détruit l'objet de collection 'NOM_13' et le nom dans le répertoire 'MES_NOMS'.

5.4 Collection contiguë de vecteurs d'entiers

Les objets de cette collection sont nommés dans le répertoire de noms 'MES_NOMS', déjà utilisé pour les deux exemples précédents. Ils ont une longueur variable définie par un vecteur de longueurs 'LONGUEURS' géré par l'utilisateur.

Dans une première routine, on crée la collection et on ajoute un objet 'NOM_24'

```
CALL JECREC ('MA_COLL','G V I','NO MES_NOMS','CONTIG','VARIABLE',25)
CALL JECROC ( JEXNOM( 'MA_COLL', 'NOM_24') )
```

Dans une autre routine, on définit la longueur des objets 2 à 25, égale au numéro d'insertion de l'objet :

```
     CALL JEVEUO ( 'LONGUEURS' , 'E', JTAB)
 C
     DO 10 I = 2, 25
         CALL JEECRA ( JEXNUM ( 'MA_COLL', I ) , 'LONMAX', ival=I)
 10  CONTINUE
```

Dans une autre routine, on définit la longueur de l'objet 1 (égale à 50), et on accède à toute la collection pour définir la 13ième composante du premier objet :

```
      CALL JEECRA ( JEXNUM ( 'MA_COLL', 1 ) , 'LONMAX', ival=50)
C
      CALL JEVEUO ( 'MA_COLL' , 'E', vi=TABC)
      K = 13
      TABC( K ) = .....
```

Dans une autre routine, on alloue le 2ième objet pour définir toutes ses composantes :

```
    CALL JEVEUO ( JEXNUM ( 'MA_COLL', 2 ) , 'E', vi=TABOC)
    CALL JELIRA ( JEXNUM ( 'MA_COLL', 2 ) , 'LONMAX', ival=L2)
    DO 20 K = 1 , L2
       TABOC( K ) = .....
20  CONTINUE
```

Dans une autre routine, on alloue la collection entière pour définir la première composante du 3ième objet, auquel on accède par le vecteur des longueurs cumulées :

```
      CALL JEVEUO ( 'MA_COLL', 'E', JTABC)
C
      CALL JEVEUO ( JEXATR ('MA_COLL', 'LONCUM'), 'E', JTABCU)
C
      IOBJ = 3
      IAD  = ZI (JTABCU - 1 + IOBJ )
      ZI (JTABC - 1 + IAD ) = .....
```

On remarquera les deux manières qui permettent de définir la longueur d'un objet de collection :

- en utilisant l'appel à JEECRA
- en affectant directement la valeur dans le vecteur des longueurs.

La première requête à JEVEUO (dans le dernier cadre) effectue un accès à la collection contiguë globalement et renvoie donc l'adresse du premier objet, à charge pour l'utilisateur de se déplacer par rapport à cette adresse pour atteindre un objet particulier. La seconde requête permet d'accéder directement à un objet de collection et renvoie l'adresse de cet objet.

Dans le dernier exemple, on récupère les longueurs cumulées des objets de la collection, ce qui permet à l'utilisateur d'accéder ainsi à n'importe quel objet, sans multiplier les requêtes JEVEUO.

Remarque :

Dans les trois cas, tous les objets de la collection contiguë sont présents en mémoire.

6 Annexe 1 : liste des sous-programmes « utilisateur »

Fonctions de type CHARACTER*32

```
JEXNOM(nom_co, nom_oc)
JEXNUM(nom_co, num_oc)
JEXATR(nom_co, 'LONCUM')
```

Création des descripteurs

```
CALL JECREO(nom_os, lis_at)
CALL JECREC(nom_co, lis_at, acces, stock, modlon, nmaxoc)
```

Affectation d'un attribut

```
CALL JEECRA(nom_o, nom_at, ival=ival, cval=cval)
```

Création d'un nom dans un répertoire de noms

```
CALL JECROC      (JEXNOM(nom_os, nom))
```

Création d'un objet de collection

```
CALL JECROC      (JEXNOM(nom_co, nom_oc))
```

Requête d'allocation

```
CALL JEVEUO      (nom_o, cel, jtab, vi, vr, ...)
CALL WKVECT      (nom_o, lis_at, long, jtab, vi, vr, ...)
```

Agrandissement d'un vecteur

```
CALL JUVECA      (nom_os, long)
```

Recopie

```
CALL JEDUPO      (nom_in,nom_clo,nom_ou,dup_co)
CALL JEDUPC      (nom_cli,souchi,ipos,nom_clo,soucho,dup_co)
```

Requêtes de libération et de sauvegarde

```
CALL JEMARQ ()
CALL JEDEMA ()
CALL JELIBE (nom_o)
CALL JELIBZ ()
```

Requête d'existence

```
CALL JEEXIN (nom_o, iret)
```

Vérifier l'existence d'un nom et obtenir son numéro d'ordre

```
CALL JENONU (JEXNOM(nom_o, nom ), num)
```

Obtenir le nom associé à un numéro d'ordre

```
CALL JENUNO (JEXNUM(nom_o, num), nom)
```

Destruction des descripteurs

```
CALL JEDETR (nom_o)
CALL JEDETC (nom_cl, souch, ipos)
```

Place disponible

```
CALL JEDISP (nbp, lplace)
```

Consultations

```
CALL JELIRA (nom_o, nom_at, ival=ival, cval=cval)
CALL JELSTC (nom_cl, souch, ipos, nbmax, l_nom, nbval)
CALL JEVEMA (marque)
```

Impressions

```
CALL JEIMPO (unit, nom_o, param, cmess)
CALL JEIMPA (unit, nom_o, cmess)
CALL JEIMPR (unit, nom_cl, cmess)
CALL JEIMPM (unit)
CALL JEIMPD (unit, nom_cl, cmess)
CALL JEPRAT (unit, nom, nom_at, param, cmess)
```

Réinitialiser n valeurs d'un vecteur

```
CALL  JERAZO (nomlu, ni, i1)
```

Déverminage

```
CALL JXVERI (cfic, cmess)
```

Mise en œuvre et environnement d'exploitation (réservées au Superviseur)

```
CALL JEDEBU (nbases, lzone, cmess, cvig, idebug)
CALL JEINIF (stin, stout, nom_base, nom_cl, nmax, nbloc, lbloc)
CALL JELIBF (cond, nom_cl)
CALL JEFINI (cond)
CALL JETASS (nom_cl)
CALL JEIMHD (fichdf, nom_cl)
CALL JELIHD (nom_base, fichdf, nom_cl)
CALL JELIAD (nom_cl, numr, nboct)
```

7 Annexe 2 : liste des attributs accessibles

On utilise la codification suivante :

A	attribut affectable et non modifiable ensuite	(par JECREO, JECREC ou JEECRA)
M	attribut modifiable	(par JEECRA)
C	attribut consultable uniquement	(par JELIRA ou JEIMPA)
os	objet simple	
co	collection	
oc	objet de collection	
disper	collection dispersée	
contig	collection contiguë	
var	collection de longueur variable	
const	collection de longueur constante	

Lorsque l'attribut est uniquement accessible pour un genre donné, ce dernier est indiqué en première colonne § Les attributs génériques. Le blanc indique que l'attribut n'est pas accessible.

Dans le cas des collections et des objets de collection, on indique, s'il y a lieu, le type de collection concernée (contig & var pour une collection contiguë de longueur variable).

	valeurs possibles	**genre**	**os**	**co**	**oc**
CLAS			A	A	C
GENR	E , V ,ou N		A	A	C
TYPE (1)	I,S,R,C,L,K8, K16,K24,K32,K80		A	A	C
LTYP			C	C	C
LONMAX		V	A	A (const)	A (var)
NOMMAX		N	A		
LONUTI		V	M	M (const)	M (var)
NOMUTI		N	C	C	
DOCU			M	M	
DATE			C	C	
IADM			C	C(contig)	C(disper)
IADD			C	C(contig)	C(disper)
LONO			C	C(contig)	C(disper)
USAGE			C	C(contig)	C(disper)
ACCES	NO, NO nom_uti, NU (2)			A	
STOCKAGE	CONTIG DISPERSE			A	
MODELONG	CONSTANT VARIABLE nom_uti (3)			A	

LONT				A (contig & var)	
NMAXOC				A	
NUTIOC				C	

Remarques

(1) La routine JELIRA renvoie uniquement K pour la valeur de l'attribut TYPE pour les type K8, K16, K24, K32 et K80 . Il ensuite consulter l'attribut LTYP .

(2) La routine JELIRA renvoie soit « NO » pour un pointeur interne, soit « NO nom » où nom est un nom de pointeur de noms externe.

(3) La routine JELIRA renvoie soit « CONSTANT » , soit « VARIABLE » , soit « VARIABLE nom » où nom est un nom de pointeur de longueur externe.