Manuel de Descriptif Informatique
Fascicule D2.06 :
Document D2.06.01

Usage de JEVEUX

Résumé :

Il s'agit ici d'indiquer quelques notions de fonctionnement du gestionnaire de mémoire JEVEUX afin de préciser l'usage des routines "utilisateur", d'indiquer les routines les mieux appropriées à certaines actions et de signaler les difficultés d'usage. On présente ici les règles d'usage en italique dans chaque paragraphe.

Table des matières

1 Usage des bases

Les objets simples peuvent être créés par les routines JECREO et WKVECT, les collections par la routine JECREC.

WKVECT permet d'enchaîner les trois appels JECREO, JEECRA et JEVEUO pour un objet de genre vecteur.

La notion de base permet d'associer les différents objets à un fichier sauvegardé ou non en fin de travail. Les objets à conserver en fin de travail seront créés sur la base GLOBALE associée à la classe G. Cette base permet de conserver les structures de données et d'effectuer des poursuites.

Les objets de travail seront créés sur la base VOLATILE associée à la classe V. Cette base est détruite à la fin du travail. Par convention, on utilisera les caractères && au début du nom de tout objet associé à cette base.

La base LOCALE associée à la classe L est réservée au Superviseur.

On rappelle qu'il n'est pas possible de disposer d'objets de nom identique sur des bases différentes. Les routine JE.... ne possèdent pas parmi leurs arguments le nom de la classe et la recherche du nom s'effectue dans l'ensemble des répertoires associés aux différentes bases ouvertes.

base GLOBALE	nom de classe G	objets conservés
base VOLATILE	nom de classe V	objets temporaires
base LOCALE	nom de classe L	réservée au Superviseur

2 Accès par nom

L'accès aux objets gérés par JEVEUX est effectué à l'aide du nom. On utilise une fonction de codage qui fournit à partir du nom et de différents paramètres une clef d'accès (un entier), cette clef permet ensuite d'accéder aux différents attributs. L'accès par nom est relativement coûteux (décodage de caractères, gestion de collision, etc...) aussi conserve-t-on dans une variable le dernier nom (d'objet simple, de collection et d'objet de collection) et l'identificateur (obtenu à partir de la clef) associé, pour éviter un nouvel appel à la fonction de codage.

Remarque :

Il est donc recommandé d'effectuer toutes les requête pour un même objet JEVEUX de façon séquentielle afin de bénéficier de cette possibilité.

3 Accès aux segments de valeurs

Les routines WKVECT et JEVEUO renvoient à l'utilisateur une adresse relative dans l'un des tableaux ZR, ZI, ZC, ou ZK, (on notera par la suite Z? l'une de ces variables Fortran). Cette adresse est valide tant qu'il n'y a pas eu de libération.

La notion d'accès en écriture ou en lecture permet d'éviter le déchargement systématique sur disque du segment de valeurs et limite ainsi le nombre des entrées/sorties sur disque. Les objets accédés en lecture ne seront pas sauvegardés sur disque au moment de la libération. L'appel à WKVECT effectue une requête en écriture.

Un segment de valeurs a pu être accédé en écriture puis libéré, le gestionnaire le conserve alors en mémoire et diffère son déchargement sur disque lors d'une prochaine recherche de place. Un nouvel accès en lecture renvoie l'adresse du segment déchargeable, une modification du contenu peut donc avoir lieu à l'insu de l'utilisateur si ce dernier affecte le contenu du tableau Z? à l'adresse indiquée.

Règle d'usage :

L'utilisateur, lorsqu'il effectue un accès en lecture, ne doit pas modifier le contenu du tableau Z? à l'adresse fournie lors de la requête et doit éviter de le passer en argument d'un sous-programme dont il n'aurait pas l'entière maîtrise.

Les appels à la routine JEVEUO renvoient une adresse par rapport à une variable Z? du même type que l'objet JEVEUX (cette adresse est mesurée dans la longueur du type). Le commun normalisé doit figurer dans toute unité de programme effectuant ce type d'appel.

```
INTEGER          ZI
COMMON/IVARJE/   ZI(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)
```

Remarque :

Ne pas modifier le nom des variables du commun de référence.

L'accès à un segment de valeurs est réalisé de la façon suivante : si JTAB désigne l'adresse renvoyée par la routine JEVEUO pour un objet de genre vecteur et de type I, KTAB celle pour un objet de type C (complexe) :

ZI (JTAB)	est la première valeur d'un vecteur d'entiers,
ZI (JTAB + I - 1)	est la Iéme valeur d'un vecteur d'entiers,
ZC (KTAB + I - 1)	est la Iéme valeur d'un vecteur de complexes.

4 Initialisation des valeurs

Lors de l'appel à JEVEUO ou à WKVECT le gestionnaire de mémoire effectue une recherche de place en mémoire centrale. Si l'objet ne possède pas d'image sur disque (c'est-à-dire lors de son premier accès en écriture), le segment de valeur est initialisé suivant le type de l'objet : 0. pour les réels, (0.,0.) pour les complexes, 0 pour les entiers, ' ' (blanc) pour les caractères.

Règle d'usage :

Il est inutile d'effectuer une boucle d'initialisation avant la première utilisation du segment de valeurs.

5 Libération des segments de valeurs et notion de marque

Si rien n'est fait (pas d'appel à JELIBE), les segments de valeurs ramenés en mémoire y restent et cela peut conduire rapidement à sa saturation. D'un autre côté, si on libère un objet sans prendre de précautions (une marque), on risque de rendre invalide l'adresse d'un objet demandé en amont de la programmation. Une solution systématique à ce problème n'est pas encore adoptée aujourd'hui.

Ce qui est fait actuellement (et qui pourra être remis en cause) :

- on libère peu d'objets (les gros) ce qui permet de s'assurer que les libérations ne sont pas dangereuses (bonne connaissance de l'usage de ces objets) ;
- on utilise les marques pour libérer les objets.

Les règles d'usage concernant ce problème seront choisies prochainement. En attendant :

Règle d'usage :

Ne pas libérer systématiquement les objets.

6 Sauvegardes

La routine JESAUV permet en théorie de protéger les objets que l'on vient de créer (éviter de réécrire leur contenu). Dans la pratique, cette routine n'est presque jamais utilisée. Pour l'homogénéité du code il nous semble qu'une règle devrait être choisie : sauver (ou non) tous les concepts. Cette décision n'a pas été encore prise.

Règle d'usage :

Ne pas utiliser JESAUV et JEDELI.

7 L'allocation d'un grand segment de valeurs

Un manque de place dans l'espace géré par JEVEUX conduit à l'arrêt brutal du programme (message d'erreur suivant : <S> <JJALLS02> FERMETURE DES BASES SUR ERREUR JEVEUX), seuls les concepts créés par les opérateurs précédents sont conservés dans la base GLOBALE. Il est possible d'éviter ce type de problème en prenant la précaution de faire appel à la routine JEDISP afin d'obtenir les dimensions maximales dans l'ordre décroissant des n zones d'espace disponible en mémoire (n étant un entier fourni par l'utilisateur).

Règle d'usage :

Cet appel doit être réservé à l'allocation de très gros objets dans des portions de programmation sensible (par exemple les solveurs).

En effet la recherche des zones de mémoire nécessite un parcours complet du chaînage des segments de valeur et peut se révéler coûteuse.

8 Détection des écrasements mémoire

JEVEUX gère dynamiquement une zone mémoire séquentielle dont la taille est déterminée par le paramètre mémoire défini à l'exécution du travail. Les différents segments de valeurs associés aux objets sont déposés dans cette zone les uns derrière les autres. Chaque segment de valeurs est encadré par 4 entiers devant et 4 derrière. Ces entiers permettent de définir un chaînage avant et un chaînage arrière, de stocker l'état et le statut, l'identificateur et la classe ainsi qu'un décalage pour gérer les types de longueur supérieure à l'unité d'adressage.

L'écrasement des entiers situés autour du segment provoque la rupture du chaînage et la perte de l'identité de l'objet associé au segment de valeurs. Un tel écrasement est généralement détecté à l'occasion d'une recherche de place en mémoire (au moment où l'on parcourt le chaînage) et non au moment du débordement. Il se traduit par un des messages d'erreur suivants :

```
<S> <JJLIRS> <ECRASEMENT AMONT POSSIBLE ADRESSE> nnnn
```

Dans ce cas on a écrasé un des entiers situés devant le segment de valeurs.

```
<S> <JJLIRS> <ECRASEMENT AVAL POSSIBLE ADRESSE> nnnn
```

Dans ce cas on a écrasé un des entiers situés derrière le segment de valeurs.

Règle d'usage :

Le développeur dispose alors de la routine JXVERI pour instrumenter son code.

Cette routine vérifie l'intégrité du chaînage et peut signaler le point de rupture. Elle détecte en plus une incursion hors de la zone mémoire licite. Il est possible de mettre en oeuvre à moindre frais dans la commande DEBUT ou POURSUITE l'appel à ce sous-programme avant chaque commande, ce qui permet de déterminer quelle commande effectue l'écrasement. Le développeur pourra alors, en instrumentant les routines associées à la commande, procéder par dichotomie pour déterminer la routine ou les instructions erronées.

L'écrasement peut aussi être moins important et n'affecter qu'un mot devant ou derrière le segment de valeurs sans briser le chaînage, c'est le cas lorsque l'on fait une erreur d'indice dans le tableau Z?. Le contenu de l'usage ou du statut du segment de valeur est alors affecté, cette information peut être obtenue en consultant le résultat des impressions du découpage de la mémoire par la routine JEIMPM.

9 Agrandir un vecteur

Règle d'usage :

L'utilisateur dispose de la routine JUVECA pour agrandir un objet simple de genre vecteur.

C'est une surcouche écrite à partir des routines utilisateur JEVEUX. Elle construit un objet temporaire et détruit l'original après recopie. On obtient la nouvelle adresse relative en retour parmi les arguments. Ces diverses opérations peuvent être assez coûteuses, il est donc préférable de minimiser le nombre d'appels pour un même vecteur et plutôt doubler la taille que de l'agrandir au fur et à mesure. Attention, cette routine détruit la marque associée à l'objet initial et affecte la marque courante au nouvel objet.

10 La recopie des objets

Il est possible de recopier les objets JEVEUX sur une même base ou d'une base à l'autre. La recopie des objets simples ne pose pas de problème particulier, par contre il est plus délicat de manipuler des collections. Une collection peut s'appuyer sur un répertoire de noms externe ou un pointeur de longueur externe. Ces objets simples doivent être créés et en partie gérés indépendamment (par exemple leur destruction doit être explicite). Leur nom peut donc être sans rapport avec le nom de la collection.

Deux cas sont possibles :

- la recopie s'effectue sur des bases différentes : les pointeurs externes seront dupliqués et deviendront des pointeurs internes à la collection,
- la recopie s'effectue sur une même base : les pointeurs externes peuvent être conservés ou bien il sont dupliqués et deviennent internes.

Si le réceptacle est déjà existant, il est détruit avant recopie.

Règle d'usage :

Utiliser `JEDUP0` ou `JEDUPC` - Ne pas utiliser `COPISD`, `COPI0B` ou `COPI0C`.

11 Les routines travaillant sur des groupes d'objets

L'organisation des structures de données d'*Aster* repose en grande partie sur les noms des objets. On manipule au sein du code des "concepts" bâtis à partir d'un nom fourni par l'utilisateur comme résultat des commandes. Il a donc paru commode de pouvoir manipuler des groupe d'objets en fournissant une sous-chaîne de caractères, qui est recherchée dans les noms de tous les objets présents dans les répertoires.

Les routines `JELIBC`, `JEDETC` et `JEDUPC` s'appliquent à des listes d'objets. Elles permettent, dans l'ordre, de libérer, de détruire et de dupliquer les objets de ces listes.

Ces routines offrent plus de souplesse au développeur pour gérer les objets (structures de données).

Remarque :

Il est peu recommandé de les utiliser dans les sous-programmes de bas niveau, le coût de la recherche à l'aide de la sous-chaîne pouvant affecter sensiblement les performances.