AtoutFox se mobilise pour sauver Eddy MAUE :
Après celles de Christophe Chenavier,
voici une autre contribution sur les

COLLECTIONS[1]

Sommaire

1 Ce document a été créé avec Open Office.

1- Définition

Une collection est un ensemble de valeurs ou d'objets ayant quelque chose en commun. Au minimum, c'est comme un tableau (array) unidimensionnel, au maximum ... c'est énorme !

Une collection est une classe de base proche du container (elle ne comporte toutefois pas de méthode addobject() par exemple). On peut donc y ranger des valeurs et/ou des objets, les retrouver soit par leur indice soit par une clef, et les supprimer.

La classe COLLECTION n'est disponible qu'à partir de VFP8.

2- Avec ou sans clef ?

La méthode .ADD est le seul moyen qui permette l'ajout d'un élément dans une collection. Mais elle a une option très importante :

1. **sans clef : le seul moyen de retrouver un objet ou une valeur est son indice; exactement comme dans un tableau.**
2. **avec une clef caractère (alphanumérique) : à chaque objet ou valeur ajouté est lié une clef exactement comme un index dans une table. Pour rechercher une valeur ou un objet on peut soit utiliser son indice numérique soit sa clef alphabétique. Attention la clef doit être unique (c'est une clef 'primaire'); elle est sensible à la casse (une majuscule est différente de sa minuscule). L'utilisation de la clef pour retrouver une donnée est très intéressante.**

Lorsque vous utiliser FOR EACH pour parcourir votre collection, la propriété KEYSORT vous permet de définir l'ordre du parcours.

C'est la manière d'ajouter le premier élément de la collection qui détermine si l'ensemble de la collection est avec clefs ou sans.

3- Quelques exemples

Vous trouverez de nombreux exemples dans les contributions de Christophe Chenavier et dans l'aide de VFP9. D'autre part, de nombreux objets VFP peuvent être considérés comme des collections : les forms par exemple. Voici quelques exemples simples (que vous trouverez dans le fichier « demo_collection.prg » ci-joint) :

```
PROCEDURE demo1
   * on peuple une collection avec des valeurs sans clef
   locollection = CREATEOBJECT("collection")
   locollection.add(1)
   locollection.add(2)
   locollection.add(3)
   locollection.add(4)
   * en fait on vient de créer un tableau unidimensionnel avec 4 valeurs dedans
   * on imprime le contenu dans l'ordre croissant puis dans l'ordre décroissant
   * en utilisant les syntaxes possibles
   locollection.keysort = 0 && index ascendant
   ? "ordre ascendant"
   FOR EACH value IN locollection
     ? value
   NEXT
   locollection.keysort = 1 && index descendant
   ? "ordre descendant"
   FOR EACH value IN locollection
     ? value
   NEXT
   ? "avec un indice selon les 2 syntaxes possibles"
   FOR m.lnindice = 1 TO m.locollection.count
    ? TRANSFORM( m.locollection( m.lnindice)) + " "+ ;
     TRANSFORM(m.locollection.item( m.lnindice))
   NEXT
   ? "on supprime le 3éme élément et on reimprime "
   locollection.remove(3)
   locollection.keysort = 0 && index ascendant
   FOR EACH value IN locollection
     ? value
   NEXT
   ? "on ne peut pas insérer un élément dans la collection "
   ? "car elle n'est pas indexée"
  ENDPROC && demo1


PROCEDURE demo2
   * même chose que demo1 mais on utilise une collection avec clefs
   locollection = CREATEOBJECT("collection")
   * je met des clefs dans le désordre pour la démo
   locollection.add(1, "5")
```

```
  locollection.add(2, "8")
  locollection.add(3, "3")
  locollection.add(4, "2")
  locollection.keysort = 0 && index ascendant
  ? "index ordre ascendant"
  FOR EACH value IN locollection
    ? value
  NEXT
  locollection.keysort = 2 && clef ascendant
  ? "clef ordre ascendant"
  FOR EACH value IN locollection
    ? value
  NEXT
  ? "avec l'indice"
  FOR m.lnindice = 1 TO m.locollection.count
   ? TRANSFORM( m.locollection( m.lnindice))
  NEXT
  ? "on supprime le 3éme élément et on reimprime "
  locollection.remove(3)
  locollection.keysort = 0 && index ascendant
  FOR EACH value IN locollection
    ? value
  NEXT
  ? "on insère un élément avant celui de clef '8' et on imprime"
  ? "(et pour corser un peu l'élément ajouté est un texte)"
  locollection.add("  texte", "99", "8")
  locollection.keysort = 0 && index ascendant
  ? "index ordre ascendant"
  FOR EACH value IN locollection
    ? value
  NEXT
  locollection.keysort = 2 && clef ascendant
  ? "clef ordre ascendant"
  FOR EACH value IN locollection
    ? value
  NEXT
  ? "avec l'indice"
  FOR m.lnindice = 1 TO m.locollection.count
   ? TRANSFORM( m.locollection( m.lnindice))
  NEXT
ENDPROC && demo2
```

Un autre exemple simple va vous montrer un usage formidable des collections. On suppose que l'on doit facturer des repas pour une cantine. On a d'une part le tarif des repas qui dépend d'une caractéristique des clients et d'autre part une table contenant les clients. Le but de l'exemple est d'imprimer le nombre de repas, leur tarif et le coût pour le client. Rien n'est plus simple :

```
PROCEDURE demo_tarif
  LOCAL lotarif
```

```
 * on suppose que l'on doit facturer des repas à des clients
 * d'une cantine selon 2 tarifs différents le 'A' pour les
 * habitués et le 'B' pour les occasionnels
 lotarif = CREATEOBJECT("collection")
 * bien sûr on pourrait mettre à jour la collection à partir d'une table
 lotarif.add( 5.5, "A") && 5€50 le repas pour les habitués
 lotarif.add( 6.2, "B") && 6.20€ pour les occasionnels
 * on crée un curseur de clients aléatoires
 CREATE CURSOR clients( nom C(30), tarif C(1), nombre N(3,0))
 INSERT INTO clients VALUES ("MAURICE", "A", 1)
 INSERT INTO clients VALUES ("CHENAVIER", "B", 1)
 INSERT INTO clients VALUES ("LEVY", "A", 10)
 INSERT INTO clients VALUES ("NIVELET", "B", 10)
 INSERT INTO clients VALUES ("LEPAGE", "A", 22) && c'est un gros mangeur !
 SCAN ALL
  ? clients.nom+ STR(clients.nombre,3)+"*"+ ;
   TRANSFORM(m.lotarif( clients.tarif), "99.99") + "=" +;
   TRANSFORM(clients.nombre * m.lotarif( clients.tarif), "9999.99") + "€"
   * Vous ne trouvez pas que c'est vachement simple ?
 ENDSCAN
 USE
ENDPROC && demo_tarif
```

Encore plus fort : supposons que pour chaque tarif on ait 3 valeurs : une pour les repas normaux, une pour les absences et la troisième pour les repas exceptionnels. On va résoudre ce problème simplement en créant une collection de 3 valeurs à la place de la valeur de l'exercice précédent (on aura donc une collection qui contient d'autres collections ...)

```
PROCEDURE demo_tarif_2
 LOCAL lotarif
 * même problème que ci-dessus sauf que chaque tarif contient
 * 3 valeurs : une pour les repas normaux, l'autre pour les
 * repas exceptionnels et la troisième pour les absences
 lotarif = CREATEOBJECT("collection")
 * bien sûr on pourrait mettre à jour les collections à partir d'une table
 * notez bien que l'on peut utiliser un createobject comme
 * paramètre de la méthode add
 lotarif.add( CREATEOBJECT("collection"), "A")
 lotarif("A").add(5.5, "N")
 lotarif("A").add(7, "E")
 lotarif("A").add(5, "A")
 lotarif.add( CREATEOBJECT("collection"), "B")
 lotarif("B").add(6.2, "N")
 lotarif("B").add(8, "E")
 lotarif("B").add(6, "A")
 * on pourrait utiliser WITH ....
 * on crée un curseur de clients aléatoires
 CREATE CURSOR cl( nom C(30), tarif C(1), normaux N(3,0), ;
  abscence N(3,0), Exception N(3,0))
 INSERT INTO cl VALUES ("MAURICE", "A", 1, 0, 0)
```

```
 INSERT INTO cl VALUES ("CHENAVIER", "B", 1, 1, 1)
 INSERT INTO cl VALUES ("LEVY", "A", 10, 2, 2)
 INSERT INTO cl VALUES ("NIVELET", "B", 10, 0, 1)
 INSERT INTO cl VALUES ("LEPAGE", "A", 22, 0, 0)
 SCAN ALL
   ? cl.nom+ STR(cl.normaux,3)+"*"+ ;
    TRANSFORM(m.lotarif( cl.tarif).item("N"), "99.99") + "+" +;
    STR(cl.abscence,3)+"*"+ TRANSFORM(m.lotarif( cl.tarif).item("A"), "99.99")+ ;
    "+" + STR(cl.exception,3)+"*"+ TRANSFORM(m.lotarif( cl.tarif).item("E"), "99.99")+ ;
    " = "+ TRANSFORM(cl.normaux * m.lotarif( cl.tarif).item("N")+ ;
     cl.abscence*m.lotarif( cl.tarif).item("A")+ ;
     cl.exception * m.lotarif( cl.tarif).item("E"), "9999.99")+ "€"
   * Vous ne trouvez pas que c'est vachement puissant ?
   * Bien sûr on pourrait optimiser ...
 ENDSCAN
 USE
ENDPROC && demo_tarif
```

4- Provisoirement

Je n'irai pas plus loin par manque de temps mais petit à petit je compléterai ce document avec des collections de plus en plus complexes, l'ajout de propriétés, etc . Si par ailleurs vous avez des exemples ... on peut les intégrer ici !

5- Conclusion

VFP nous donne beaucoup de moyens pour mémoriser les données d'une application. Les collections sont intéressantes car elles introduisent la notion d'index qui est très puissante.