# Procédure POURSUITE

## 1 But

Poursuivre une étude à partir de la sauvegarde au format JEVEUX.

La syntaxe apparemment complexe de cette procédure ne doit pas inquiéter l'utilisateur, l'appel avec les opérandes par défaut, est suffisant dans la plupart des cas : POURSUITE ( )

L'usage de cette commande est tout à fait semblable à celui de DEBUT. Les mots-clés dont le comportement est identique dans DEBUT et POURSUITE sont uniquement décrits dans [U4.11.01].



Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

# 2 Syntaxe

```
POURSUITE
      (  ◊  IMPR_MACRO = /   'NON',                          [DEFAUT]
                         /   'OUI',

         ◊  LANG = lang,                                     [TXM]

         ◊  BASE = _F (   ♦  FICHIER = 'VOLATILE',

                          ◊  /    | LONG_ENRE= lenr,         [I]
                             |    | NMAX_ENRE= nenr,         [I]
                             |    | LONG_REPE= lrep,         [I]
                          ),
         ◊  CODE =     /   'NON',                            [DEFAUT]
                       /   'OUI',

         ◊  ERREUR = _F   (  ERREUR_F  =  /   'ABORT',       [DEFAUT]
                                          /   'EXCEPTION',
                          ),
            IGNORE_ALARM =   l_vale,                         [l_Kn]

         ◊  DEBUG = _F(   ◊  JXVERI =     /   'OUI',
                                          /   'NON',
                          ◊  ENVIMA =         'TEST',        [l_Kn]
                          ◊  JEVEUX =      /   'OUI',
                                          /   'NON',
                          ◊  SDVERI =      /   'OUI',
   ),

         ◊  MESURE_TEMPS = _F (… voir commande DEBUT),

         ◊  MEMOIRE = _F( … voir commande DEBUT),

         ◊  RESERVE_CPU =_F( / VALE =   vale                 [R]
                                / POURCENTAGE = pcent        [R]

                                   ◊ BORNE  =  /   bv,       [R]
                                               /   900.      [DEFAUT]
                          ),
       )
```



Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

# 3 Principe de fonctionnement

Cette procédure affecte, en outre, les ressources mémoire nécessaires à la poursuite du calcul.

Les opérandes de la commande sont homologues de ceux de la procédure DEBUT [U4.11.01]. Ils permettent de préciser certaines ressources affectées à la nouvelle exécution.

L'étude engagée précédemment se poursuit par un ensemble de commandes commençant par POURSUITE et se terminant par FIN [U4.11.02].

La procédure POURSUITE qui est exécutée effectue les tâches suivantes :

- définition des unités logiques des fichiers utilisés en impression,
- allocation des fichiers associés aux bases de données gérées par JEVEUX,
- lecture des catalogues de commandes mais pas des catalogues des éléments qui ont été recopiés sur la base de données lors de la première exécution.

Les opérandes sont à utiliser pour dérouter les différents fichiers sur des numéros d'unité logique différents des numéros affectés par défaut ou pour ajuster certains paramètres de fichiers.

Les concepts de python simples (de type variable) créés lors d'une exécution précédente sont conservés dans un fichier associé à la base JEVEUX (pick.*). Lors de l'exécution de la procédure POURSUITE ces concepts sont regénérés et peuvent donc être utilisés sous le nom sous lequel ils ont été créés.

**Remarque**

*Ne sont pas sauvegardés dans le pickle, les objets python de type classe, fonction et type. Voir le paragraphe Exemples pour un mode d'utilisation possible.*

# 4 Opérandes

Les mots clés LANG et DEBUG sont identiques à ceux de la procédure DEBUT [U4.11.01].

Le mot clé BASE est différent pour la procédure POURSUITE.

## 4.1 Mot clé BASE

BASE =

La fonctionnalité de ce mot clé est de redéfinir les valeurs des paramètres des fichiers d'accès direct associés aux «base de données» dans le cas où l'on ne désire pas utiliser ceux fixés par défaut. La taille maximum du ou des fichiers associés, et par conséquent le nombre maximum d'enregistrements, peut être redéfini à l'aide du paramètre passé sur la ligne de commande derrière le mot clé

En mode POURSUITE, certaines caractéristiques de la base GLOBALE ne peuvent plus être modifiées. Valeurs par défaut des paramètres associés aux bases de données

| VOLATILE | | |
|---|---|---|
| NMAX_ENRE | 62914 | |
| LONG_ENRE | 100 | K mots |
| LONG_REPE | 2000 | |

Le mot vaut 8 octets sur plate-forme 64 bits et 4 octets sur plate-forme 32 bits.



Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Sous Linux 64, la procédure POURSUITE, avec les valeurs par défaut, allouera un fichier d'accès direct d'au plus $62914$ enregistrements de $100\,Kmot$ (le $K$ vaut $1024$) pour la base 'VOLATILE'.

**Remarque :**

*La taille réelle du fichier est dynamique ; elle dépend du volume d'informations à stocker effectivement. Cette taille est limitée par les conditions d'exploitation et un paramètre défini parmi les valeurs caractérisant la plate-forme. Sur la plate-forme de référence Linux 64 bits, la taille initiale est fixée à 48 Go. Cette valeur est utilisée pour dimensionner 2 objets utilisés par le gestionnaire de mémoire, elle sera modifiée automatiquement en cours d 'exécution si besoin. Il est possible de modifier cette valeur en passant un argument sur la ligne de commande de l'exécutable derrière le mot clé « --max_base taille » où taille est une valeur réelle mesurée en Mo.*
*Pour la base Globale, qui peut être sauvegardée et ré-utilisée en donnée d'un calcul, la taille maximum initiale en « POURSUITE » est conservée telle quelle si le paramètre « --max_base » n'est pas utilisé, mais peut-être redéfini au besoin de cette manière.*

### 4.1.1 Opérande FICHIER

♦ FICHIER =
Nom symbolique de la base considérée.

Seul le paramètre de la base de données 'VOLATILE' peut être redéfini.

### 4.1.2 Opérandes LONG_ENRE / NMAX_ENRE / LONG_REPE

Définition des paramètres de la base de données (fichiers d'accès direct).

/ | LONG_ENRE = lenr

lenr est la longueur des enregistrements en $Kmots$ des fichiers d'accès directs utilisés.

**Remarque :**

*Le gestionnaire de mémoire JEVEUX utilise ce paramètre pour déterminer deux types d'objets : les gros objets qui seront découpés en autant d'enregistrements que nécessaire, et les petits objets qui seront accumulés dans un tampon de la taille d'un enregistrement avant d'être déchargé.*

| NMAX_ENRE = nenr

nenr est le nombre d'enregistrements par défaut, cette valeur est déterminée à partir de LONG_ENRE et d'un paramètre d'exploitation sous LINUX64 fixé à $12\,Go$ (51 539 607 552 octets) pour la taille maximale du fichier associé à une base de données.

**Remarque :**

*Les deux opérandes LONG_ENRE et NMAX_ENRE doivent être utilisés avec précaution, un mauvais usage pouvant conduire à l'arrêt brutal du programme par saturation des fichiers d'accès direct. La cohérence entre la taille maximale du fichier et la valeur résultant du produit des deux paramètres LONG_ENRE et NMAX _ENRE est vérifiée en début d'exécution.*

| LONG_REPE = lrep

lrep est la longueur initiale du répertoire (nombre maximal d'objets adressables par JEVEUX), elle est gérée dynamiquement par le gestionnaire de mémoire qui étend la taille du répertoire et de tous les objets système associés au fur et à mesure des besoins.

## 4.2 Mot clé CODE



Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Ce mot clé permet d'activer la génération du fichier .code. Réservé aux cas-tests, ce mot clé est une version simplifié du mot clé CODE de DEBUT.

# 5 Exemples d'utilisation

L'utilisation standard de cette procédure est :

```
POURSUITE()
```

Python ne sauvegarde pas les fonctions et classes définies dans le fichier principal (il se souvient simplement dans quel module Python elles ont été définies). C'est donc le cas du fichier de commandes de code_aster.
Pour utiliser des fonctions ou classes et les retrouver en poursuite, il faut les définir dans un module externe :
- Soit le module userpkg/usermod.py :

```
def pyfunc(x) :
    return x * 2
```

- On ferait dans le premier jeu de commandes :

```
DEBUT()
from userpkg import usermod
form = FORMULE(NOM_PARA='X', VALE='pyfunc(X)', pyfunc=usermod.pyfunc)
assert form(2) == 4
FIN()
```

- Puis :

```
POURSUITE(PAR_LOT='NON')
assert form(2) == 4
FIN()
```

Pour cela, il suffit d'ajouter dans l'étude le répertoire userpkg de type nom.
Il faut bien comprendre que, lors de la relecture, Python va chercher à importer le module userpkg.usermod. Il doit donc le retrouver au même endroit.
De plus, en important ce module, Python exécute les instructions qu'il y trouve.
Pour cette raison, **il est déconseillé de créer des concepts dans le corps principal d'un module importé** (car ils seront créés à chaque import et encombreront la base). Il faut le faire dans des fonctions qui seront uniquement exécutées sur demande.



Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)