Introduire une nouvelle commande

Résumé :

Ce document décrit la méthode pour introduire une nouvelle commande (opérateur ou procédure) dans *Code_Aster* . Il décrit la rédaction au format « python » du catalogue de la commande et de la routine FORTRAN associée.

Table des matières

1 Introduction....3
2 Vision utilisateur d'une commande....3
3 Rédaction du catalogue de commande....4
4 Définir le lien entre le catalogue et le programme FORTRAN associé....6
5 Définir les attributs des mots clés simples....6
6 Cas des mots clés facteurs....9
7 Exclure ou regrouper des mots clés : argument regles....10
8 Les blocs....11
9 Typer le concept produit et l'enrichir....12
9.1 Typer le concept produit....12
9.2 Enrichir le concept produit....14
10 Routine d'utilisation....16
10.1 Nom de la routine....16
10.2 Les deux étapes....16
10.3 Récupération des arguments de la commande....16

1 Introduction

Pour introduire une nouvelle commande dans le *Code_Aster* , il faut :

- écrire le catalogue associé à cette commande (Voir le paragraphe Rédaction du catalogue de commande),
- écrire la routine FORTRAN OPxxxx associée (Voir le paragraphe Typer le concept produit et l'enrichir).

Remarques sur les fichiers source

On trouve les fichiers qui décrivent le catalogue dans le répertoire code_aster/Cata .
Afin de préparer de futurs développements, il y a une indirection à ce niveau vers Legacy (actuel) ou Language (futur). Quand on parle de DataStructure.py , pour la version actuelle, le fichier importe les objets définis dans Legacy/DataStructure.py .

2 Vision utilisateur d'une commande

Prenons comme exemple la commande AFFE_MATERIAU qui permet d'affecter sur un maillage des caractéristiques de matériau. Voici une utilisation possible de cette commande dans le fichier de commandes fourni par l'utilisateur de *Code_Aster* :

```
cham = AFFE_MATERIAU  (   MAILLAGE  =  mail,
                          AFFE      =  _F(TOUT  = 'OUI',
                                          MATER = acier  )
                      )
```

Lors de l'utilisation d'une commande, il apparaît :

- le nom "utilisateur" du concept produit par la commande : cham
- le nom de la commande : AFFE_MATERIAU
- un ou des mots clés facteurs : AFFE
- des mots clés simples : TOUT , MATER , MAILLAGE
- des noms "utilisateurs" de concepts arguments : acier , mail
- des valeurs de type simple (entier, réel, texte, ...) seules ou en liste : ' OUI '

Du point de vue utilisateur, en écrivant un nom à gauche du signe "=" de la commande, on affecte ce nom au résultat de la commande.

A ce « nom utilisateur » est affecté un concept produit (ou structure de données) calculé par l'opérateur et dont le type est donné par le superviseur. Le type du concept produit est défini dans le catalogue de la commande (Voir le paragraphe Rédaction du catalogue de commande).

Par exemple cham est le nom utilisateur du résultat de la commande et à ce nom est associé le concept de type cham_mater.

3 Rédaction du catalogue de commande

Pour introduire une nouvelle commande, il est nécessaire de créer un catalogue associé dans lequel seront indiqués :

- le nom de la commande,
- sa description en quelques mots,
- la nature de la commande : opérateur (production de concept), procédure (pas de concept produit), macro-commande,
- le numéro de la routine FORTRAN associée à cette commande (Voir Définir le lien entre le catalogue et le programme FORTRAN associé).
- pour le concept produit :
  - les règles de détermination du type du concept (Voir Typer le concept produit et l'enrichir),
  - la possibilité de réutilisation (caractère ré-entrant).
- pour les mots clés (Voir Définir les attributs des mots clés simples et Exclure ou regrouper des mots clés : argument regles),
  - si leur présence est facultative ou obligatoire, s'ils s'excluent entre eux...
  - le type de l'argument,
  - le nombre d'arguments attendus de ce type,
  - la valeur par défaut (s'il y en a une),
  - la liste des valeurs admissibles (éventuellement),
  - la plage des valeurs admissibles (éventuellement), si on attend un entier ou un réel,
- pour les mots clés facteurs (Voir Cas des mots clés facteurs):
  - si leur présence est facultative ou obligatoire (ou présent par défaut),
  - le nombre minimum et maximum d'occurrences possibles,
- les blocs : regroupement logique de mots clés quand des conditions sur d'autres mots clés sont satisfaites (Voir Les blocs).

Remarque :

On ne parlera pas dans ce document de l'introduction d'une nouvelle macro-commande (voir [D5.01.02] - Introduire une nouvelle macro-commande)

Le langage utilisé pour écrire ce catalogue est le langage interprété Python : les commentaires sont écrits derrière le caractère "#", on voit des mots clés (identificateurs suivis du caractère "="), des parenthèses, des virgules pour séparer les mots clés...

Reprenons l'exemple de la commande AFFE_MATERIAU , le catalogue - c'est-à-dire la description de la commande fournie par son **développeur** - associé est :

```
AFFE_MATERIAU = OPER(
       nom="AFFE_MATERIAU", op=6, sd_prod=cham_mater,
       fr=tr("Affectation de caractéristiques de matériaux à un maillage"),
       reentrant='n',
       MAILLAGE  =     SIMP (statut='o', typ=maillage),
       MODELE    =     SIMP (statut='f', typ=modele),
       AFFE      =     FACT (statut='o', min=1,max='**',
          regles =(UN_PARMI(  'TOUT', 'GROUP_MA', 'MAILLE',
                              'GROUP_NO', 'NOEUD'),),
          TOUT      =SIMP(statut='f', typ='TXM',into=("OUI",) ),
          GROUP_MA  =SIMP(statut='f', typ=grma,max='**'),
          MAILLE =SIMP(statut='f', typ=ma,max='**'),
          GROUP_NO  =SIMP(statut='f', typ=grno,max='**'),
          NOEUD     =SIMP(statut='f', typ=no,max='**'),
          MATER     =SIMP(statut='o', typ=mater),
          TEMP_REF  =SIMP(statut='f', typ='R',defaut= 0.E+0 ),
       ),
```

)

La syntaxe de commande est décrite à l'aide des arguments suivants. Leur signification précise sera donnée tout au long du document.

OPER/PROC/MACRO	Pour préciser le type de commande (production de un, zéro voire plusieurs concepts dans le cas des macros)
nom	Pour indiquer le nom vu par l'utilisateur pour désigner la commande
op	Pour spécifier le numéro de la routine FORTRAN de haut niveau associée à la commande
sd_prod	Pour définir le type de concept produit
regles	Pour définir les règles logiques d'appariement ou d'exclusion de mots clés
UN_PARMI/...	Pour définir une liste de mots clés parmi lesquels la donnée doit se trouver exactement une fois.
fr	Pour décrire en une phrase (en français) le rôle de la commande, c'est le contenu de la bulle d'aide affichée par AsterStudy. Il est impératif d'utiliser la fonction tr() afin de permettre la traduction du texte dans d'autres langues.
reentrant	Pour spécifier si la commande crée un nouveau concept (valeur 'n'), modifie un concept existant (valeur 'o'), ou potentiellement les deux (valeur 'f')
SIMP	Pour spécifier un mot-clé simple de la commande
FACT	Pour spécifier un mot-clé facteur de la commande.
BLOC	Pour définir un bloc de mots clés dont l'apparition est soumis à une « condition ».

On peut décomposer l'écriture du catalogue de commande selon les étapes suivantes :

- **Spécifier le type du concept produit** : (lorsqu'il existe, c'est à dire pour une commande de type **OPER**)

Pour spécifier le type du concept produit d'un opérateur, il faut utiliser l'argument sd_prod (structure de donnée produite). Par exemple, l'affectation sd_prod=cham_mater indique que cham_mater est le type du concept produit de l'opérateur AFFE_MATERIAU.

Dans le cas d'une procédure, il n'y a pas de concept produit (et donc pas d'argument **sd_prod** dans le catalogue). Par exemple CALC_G est une commande dont le type du concept produit est table_sdaster , alors que IMPR_RESU est une procédure sans concept produit :

```
CALC_G = OPER(nom="CALC_G",op=100,
              sd_prod=table_sdaster, reentrant='f', …

IMPR_RESU = PROC(nom="IMPR_RESU",op=39, …
```

Si le type du concept produit dépend des arguments de l'opérateur, on consultera Typer le concept produit .

Si le concept produit peut être un concept réutilisé et enrichi, on l'indiquera en renseignant l'argument reentrant (Voir Enrichir le concept produit).

- **Définir le nom de la commande** :

Il est écrit à gauche du signe "=" dans le catalogue, à droite dans le fichier de commandes de l'utilisateur.

Généralement le préfixe désigne l'action, le suffixe le concept traité (par exemple AFFE_MATERIAU). Notons quelques préfixes fréquemment employés :

AFFE	affectations sur le maillage ou le modèle,
DEFI	définitions d'objets qui ne sont pas des champs,
CALC	commandes appelant la routine CALCUL et produisant des champs de grandeurs.

Le nom d'une commande ne doit pas dépasser 16 caractères. Ce nom est celui utilisé par l'utilisateur dans un fichier de commandes.

- **Définir le numéro de la routine FORTRAN réalisant la commande** : (Voir Définir le lien entre le catalogue et le programme FORTRAN associé)
- **Décrire les différents mots clés** : (Voir les §5, 6 et 7) . C'est le cœur du catalogue.
- **Fermer** la parenthèse ouverte après la définition **OPER/PROC/MACRO**.

4 Définir le lien entre le catalogue et le programme FORTRAN associé

L'argument op permet l'appel à la routine FORTRAN OPxxxx qui réalise la tâche de la commande (Voir Typer le concept produit et l'enrichir). L'argument de op est un entier strictement positif compris entre 1 et 199. Le numéro est attribué par l'équipe code (cf [A2.01.02]).

Sur l'exemple considéré la routine OP0006 sera appelée lors de l'exécution de la commande AFFE_MATERIAU .

5 Définir les attributs des mots clés simples

La syntaxe générale pour déclarer un mot clé simple est :

```
MOT_CLE  = SIMP(statut=..., typ=..., into=(...,), defaut=...
                min=..., max=..., val_min=..., val_max=..., validators=...
                 ),
```

Parmi les attributs attachés à un mot clé, seuls statut et typ sont obligatoires pour tout mot clé simple :

- **Le statut**

La définition du statut par l'attribut `statut` est obligatoire.

Les statuts reconnus sont uniquement :	
`'o'`	Obligatoire : dans ce cas le mot clé devra apparaître obligatoirement dans le corps d'appel de la commande de l'utilisateur (sauf si ce mot clé est sous un mot clé facteur facultatif auquel cas le mot clé simple est obligatoire dès que le mot clé facteur apparaît).
`'f'`	Facultatif : dans le cas contraire.
`'c'`	Caché : le mot-clé n'est ni écrit dans le fichier de message, ni visible dans AsterStudy. Eviter cet usage. En général, on l'utilise pour transmettre un paramètre qui peut dépendre d'autres mots-clés et qui serait compliqué de déduire dans le fortran.
`'d'`	Défaut : Signifie que le mot-clé sera présent par défaut. N'a d'intérêt que pour les mots-clés facteurs.

- **Le type**

La déclaration de type par l'attribut `typ` est obligatoire.

Les types reconnus sont :	
`typ = 'I'`	pour les entiers
`typ = 'R'`	pour les réels
`typ = 'C'`	pour les complexes
`typ = Type_de_concept ( sans cotes !)`	pour les concepts
`typ = 'TXM'`	pour les textes
`typ = 'L'`	pour les logiques
`typ = UnitType()`	pour les unités logiques

Remarques sur les concepts

Le type de concept attendu est un type de concept créé par une autre commande que la commande en cours. Il figure parmi la liste des concepts importés dans le catalogue de déclaration des concepts (`code_aster/Cata/DataStructure.py`)

Remarque pour les unités logiques

Pour les unités logiques (de type `UnitType()`), il est nécessaire de déclarer un attribut supplémentaire `inout` qui vaut `'in'` si le fichier est lu ou `'out'` s'il est écrit par la commande.

Le type du concept attendu n'est pas nécessairement unique. Il peut être une liste, ce qui signifie que l'un ou l'autre des types sera produit. Cette liste se déclare ainsi :

```
MATR_ASSE = SIMP ( ...
                   typ=(matr_asse_depl_r, matr_asse_depl_c, ... ) ,
                   ...
                 )
```

La syntaxe documentaire de cet exemple est :

```
♦ MATR_ASSE = m          /    [matr_asse_depl_r]
                         /    [matr_asse_depl_c]
```

- **Valeur par défaut pour un mot clé**

Il est possible d'affecter une valeur par défaut à un mot clé ne recevant pas un argument de type "concept". La déclaration se fait par l'argument `defaut`

Exemples :

```
PRECISION     =SIMP( statut='f',typ='R', defaut=1.E-3 ),

FICHIER       =SIMP( statut='f',typ='TXM', defaut="RESULTAT"),
```

- **Liste de valeurs admissibles :**

Pour que le superviseur contrôle la validité du contenu de certains mots-clés, il est possible de déclarer les valeurs des arguments attendus. Cette déclaration se fait par l'argument `into`

Exemples :

```
INFO   =SIMP( statut='f', typ='I', defaut= 1 ,into=(1,2) ),
```

Le mot clé `INFO` est facultatif, sa valeur par défaut est 1 et les seules valeurs acceptées sont 1 et 2. La syntaxe documentaire est :

```
♦ INFO:    /    1    [DEFAUT]
           /    2
```

- **Nombre de valeurs attendues :**

Les arguments `min` et `max` permettent de contrôler la longueur de la liste des arguments attendus derrière le mot clé simple. Par défaut, si rien n'est précisé dans le catalogue, on attend une et une seule valeur derrière un mot clé simple (`max = 1`). Attention, déclarer `min = 1` n'apporte rien et ne revient surtout pas à rendre le mot clé obligatoire. Si un nombre potentiellement illimité d'éléments est attendu, la syntaxe est `max='**'` .

Exemples :

```
MAILLE      =SIMP( statut='f', typ=ma, max='**'),
```

L'utilisateur peut entrer ici autant de noms de mailles qu'il souhaite.

```
CENTRE      =SIMP(statut='f',typ='R',defaut=(0.,0.,0.),
                  min=3, max=3),
```

On attend ici un vecteur (liste de exactement trois réels).

- **Plage de valeurs admissibles**

Pour les entiers et les réels, on peut préciser les valeurs minimum et/ou maximum admises :

```
NU    =SIMP(statut='o',typ='R',
            val_min=-1E+0, val_max=0.5E+0),
```

Sur cet exemple, `NU` doit appartenir à l'intervalle [-1 , 0.5]. Les valeurs données par les deux arguments sont incluses dans l'intervalle.

- **Critères plus compliqués**

Outre les plages de valeurs et le cardinal de la liste, on peut imposer des critères plus compliqués sur la valeur fournie par l'utilisateur, ce sont les `validators`, définis dans `Noyau/N_VALIDATOR.py`.

On peut en programmer de nouveaux, suivant les besoins. Les principaux `validators` sont :

`PairVal()`	les entiers fournis doivent être pairs
`NoRepeat()`	vérification de l'absence de doublons dans une liste
`Compulsory(liste)`	vérification que tous les éléments de `liste` ont été fournis
`LongStr(low, high)`	vérification de la longueur d'une chaîne de caractères
`OrdList(ordre)`	vérification qu'une liste est croissante ou décroissante
`AndVal(val1, val2, ...)`	condition ET logique entre les `validators` de la liste
`OrVal(val1, val2, ...)`	condition OU logique entre les `validators` de la liste

6 Cas des mots clés facteurs

Les mots clés facteurs sont obligatoires ou facultatifs. Il est possible de contrôler les nombres minimum et maximum d'occurrences d'un mot clé facteur.

Les déclarations se font grâce au mot clé `FACT`

- **Le statut**

Il est lié au mot clé `facteur` .

Les statuts reconnus sont uniquement :

`'o'`	Obligatoire
`'f'`	Facultatif
`'d'`	Facultatif mais utilisé par défaut, i.e. facultatif pour l'utilisateur à la saisie mais obligatoire pour le fonctionnement du code. Les valeurs par défaut des mots clés simples doivent définir la syntaxe de tout le mot clé facteur vu du code quand l'utilisateur ne renseigne rien. L'utilisateur n'a pas besoin de renseigner le mot clé facteur et ses mots clés simples pour qu'il existe et soit visible du superviseur à l'exécution.

- **Le nombre d'occurrences**

Comme pour les mots clés simples, les arguments `min` et `max` permettent de préciser les occurrences attendues des mots clés facteurs. Si on ne met rien, la situation par défaut est `max=1`, le mot clé facteur n'est alors pas répétable.

Exemples :

```
MCFACT = FACT ( statut ='f', min =3, max =3, . . . )
```
le mot clé facteur est obligatoire et doit apparaître exactement trois fois.

```
MCFACT = FACT ( statut ='f', max= '**', . . . )
```
le mot clé facteur est facultatif mais peut apparaître autant de fois que l'on veut.

MCFACT = FACT (statut ='d', max =1, . . .)
le mot clé facteur est facultatif et non répétable mais si l'utilisateur ne le précise pas, il est néanmoins pris en compte et les valeurs des mots clés simples (sous le mot clé facteur) sont affectées par défaut.

7 Exclure ou regrouper des mots clés : argument regles

L'usage dans les catalogues de commandes de l'argument regles décrit ci-dessous et des blocs (paragraphe suivant) permet de reproduire entièrement la logique d'enchaînement des mots clés décrite dans le paragraphe syntaxe de la documentation d'utilisation. Il ne devrait donc y avoir que très peu vérifications de syntaxe (tests sur la présence ou le contenu de mots clés) au niveau des routines FORTRAN op0nnn.f .

Les règles, présentes sous l'argument regles , qui suivent permettent d'assurer une cohérence sur la présence simultanée des mots clés de la commande. Derrière ces définitions de règles (EXCLUS, UN_PARMI, ENSEMBLE, ...), on trouve une liste de mots clés qui sont, soit des mots clés simples (sous un même mot clé facteur), soit des mots clés facteurs. Dans la suite de ce paragraphe on n'utilisera plus que le vocable «mot clé».

EXCLUS	mc1, mc2,...,mcn Les mots clés s'excluent mutuellement.
UN_PARMI	mc1, mc2,...,mcn Un des mots clés de la liste doit être obligatoirement présent et un seul.
ENSEMBLE	mc1, mc2,...,mcn Si un des mots clés est présent, tous doivent apparaîtrent.
AU_MOINS_UN	mc1, mc2,...,mcn Il faut qu'au moins un mot clé parmi la liste soit présent. Il est licite d'en avoir plusieurs présents.
PRESENT_PRESENT	mc1, mc2,...,mcn Si le mot clé mc1 est présent alors les mots clés mc2,...,mcn doivent être présents.
PRESENT_ABSENT	mc1, mc2,...mcn Si le mot clé mc1 est présent alors les mots clés mc2,...,mcn doivent être absents.

Remarques

PRESENT_PRESENT *est différent de* ENSEMBLE *puisque pour* PRESENT_PRESENT mc2 *peut être présent, sans que* mc1 *le soit.*
PRESENT_ABSENT *est diffèrent de* EXCLUS *puisque pour* PRESENT_ABSENT mc2, …, mcn *peuvent être présents ensembles si* mc1 *est absent.*

```
regles = (   UN_PARMI('NOEUD','GROUP_NO','MAILLE'),
             PRESENT_PRESENT('MAILLE','POINT'), ),

NOEUD     =SIMP( . . . ),
MAILLE    =SIMP( . . . ),
POINT     =SIMP( . . . ),
GROUP_NO  =SIMP( . . . ),
```

Le superviseur vérifie que l'utilisateur a bien donné un et un seul des mots clés parmi NOEUD , GROUP_NO et MAILLE et, s'il a donné MAILLE , que POINT soit aussi présent.

Attention

Les mots clés manipulés dans l'argument `regles` doivent être définis au même niveau (c'est à dire à la racine principale de la commande, sous le même mot-clé facteur ou le même bloc). Plusieurs définitions de `regles` peuvent être présentes dans un même catalogue, à la racine principale de la commande ou sous des mots clés facteurs.

8 Les blocs

Les blocs se présentent sous la forme d'un regroupement de mots-clés. Ils permettent deux choses :

- traduire dans le catalogue de la commande des règles logiques portant sur la valeur ou le type du contenu des mots clé simples ; alors que les conditions sous l'argument `regles` ne portent que sur la présence ou l'absence des mots clés. On peut donc regrouper des mots clés ensemble ou leur affecter des attributs (valeur par défaut…) particuliers sous certaines conditions.
- regrouper les mots clés par familles pour plus de clarté dans AsterStudy. Ces mots-clés ne seront alors visibles à l'utilisateur que lorsque la condition sera remplie.

Exemples :

```
SOLVEUR =FACT(statut='d',

  METHODE=SIMP(statut='f',typ='TXM',defaut="MULT_FRONT",
               into=("MULT_FRONT","LDLT") ),

  b_mult_front   =BLOC(condition = "equal_to('METHODE', 'MULT_FRONT')",
                       fr=tr("Paramètres de la méthode multi frontale"),
                       RENUM=SIMP(statut='f',typ='TXM',defaut="MDA",
                                  into=("MD","MDA","METIS") ),
                        ),
  b_ldlt         =BLOC(condition = "equal_to('METHODE', 'LDLT')",
                       fr=tr("Paramètres de la méthode LDLT"),
                       RENUM=SIMP( statut='f',typ='TXM',defaut="RCMK",
                            into=("RCMK","SANS") ),
                            TAILLE=SIMP(statut='f',typ='R',defaut= 400. ),
                                  ),
                        ),
```

Les blocs sont nommés par le développeur. Leur nom doit commencer par « `b_` ». Dans l'exemple, si `METHODE` vaut `MULT_FRONT`, alors le mot clé simple facultatif `RENUM` apparaîtra avec trois valeurs possibles déclarées sous `into` . Si par contre `METHODE` vaut `LDLT` , le même mot clé sera présent mais avec deux valeurs possibles différentes ; de plus il sera alors possible de renseigner le mot clé simple `TAILLE` . Ces mots clés et leurs attributs respectifs n'apparaîtront dans AsterStudy qu'après que l'utilisateur aura affecté une valeur au mot clé simple `METHODE` .

```
b_nomdubloc=BLOC(
    condition="exists('MOTCLE1') and is_type('MOTCLE2')==grma",
    …
```

On montre sur cet exemple que la condition peut être multiple (articulée par `or / and`) et peut porter aussi sur la présence du mot clé (`exists('MOTCLE1')`) ou le type de ce qu'il contient (`is_type('MOTCLE2')== grma`).
La condition est une expression Python (fournie sous forme d'une chaîne de caractères) qui est évaluée au niveau immédiatement supérieur au bloc lui-même.

Les conditions de blocs ne doivent pas manipuler les mots-clés directement mais font appel à des fonctions qui s'appuient sur le nom des mots-clés.

Ces fonctions sont les suivantes :

- `exists('MOTCLE')` est vérifiée si `MOTCLE` existe, c'est à dire a été renseigné par l'utilisateur.
  Exemple : `exists('TYPE_MATR_TANG').`
- `is_in('MOTCLE', VALEURS)` est vérifiée si l'intersection entre la ou les valeurs de `MOTCLE` et `VALEURS` est non vide.
  Exemple : `is_in("CRIT_COMP", ('EQ','NE').`
- `equal_to('MOTCLE', VALEUR)` est strictement identique à `is_in` mais plus naturel quand `MOTCLE` et `VALEUR` ne contien nent qu'une valeur.
  Exemple : `equal_to('METHODE', 'MUMPS').`
- `is_type('MOTCLE')` retourne le type de `MOTCLE` .
  Exemple : `is_type('FONCTION') in (fonction, fonction_c).`
- `value('MOTCLE', default='')` retourne la valeur de `MOTCLE` s'il existe, sinon retourne la valeur `default` qui est une chaîne de caractères vide par défaut .
  Exemple : `value("RELATION").startswith('META_').`
- `length('MOTCLE')` retourne le nombre de valeurs fournies pour `MOTCLE` . Retourne 0 si on ne sait pas calculer la longueur de `value('MOTCLE')` ou s'il n'existe pas.
  Exemple : `length('FREQ') > 2.`
- `less_than('MOTCLE', VALEUR)` est vérifiée si la valeur de `MOTCLE` est inférieure `VALEUR` . Retourne `False` si `MOTCLE` n'existe pas.
  Exemple : `less_than('COEF_MULT', 0).`
- `greater_than('MOTCLE', VALEUR)` est vérifiée si la valeur de `MOTCLE` est supérieure `VALEUR` . Retourne `False` si `MOTCLE` n'existe pas.
  Exemple : `greater_than('COEF_MULT', 0).`

Attention **:**

Les mots clés manipulés dans la condition du `BLOC` doivent être au même niveau que le bloc lui-même dans l'arborescence définie par les mots clés facteurs et les blocs. Plusieurs mots clés `BLOC` peuvent être présents dans un même catalogue, à la racine principale de la commande, sous des mots clés facteurs ou dans d'autres blocs. Dans l'exemple ci-dessus, les mots clés simples `RENUM` et `TAILLE_BLOC` sont au même niveau, inférieur à celui de `METHODE` , `b_mult_front` et `b_ldlt` , lui même inférieur à celui du mot clé facteur `SOLVEUR` . Les conditions testées dans les deux blocs portent ainsi uniquement sur le mot clé simple `METHODE` de niveau immédiatement supérieur aux blocs eux-mêmes.
Il faut faire attention aux conflits possibles lorsqu'un même mot clé est présent sous deux blocs différents. Les conditions d'activation des deux blocs doivent alors s'exclure. C'est le cas de l'exemple ci-dessus avec le mot clé simple `RENUM` : il ne peut y avoir conflit puisque les deux conditions `METHODE='MULT_FRONT '` et `METHODE='LDLT '` ne peuvent être simultanément satisfaites. Dans un cas où les conditions seraient satisfaites au même moment, une erreur se produirait à l'exécution.

9 Typer le concept produit et l'enrichir

9.1 Typer le concept produit

L'argument `sd_prod` permet d'effectuer la déclaration du type de concept produit. Si la commande produit toujours la même structure de données quelque soit le contexte, `sd_prod` est suivi du type de concept correspondant, déjà déclaré dans le catalogue de déclaration des concepts : `DataStructure.py` et `SD/co_xxx.py`. Tous les concepts pouvant être produits par des commandes et/ou utilisés dans des mots clés sont déclarés dans ce fichier.

Exemple :

Dans le catalogue de la commande :

```
CALC_G = OPER( nom="CALC_G", op=100, sd_prod=table_sdaster,...
```

Le type `table_sdaster` est défini dans `SD/co_table.py`.

Dans certains cas le développeur veut que l'opérateur produise un concept dont le type est déterminé dynamiquement (i.e. à l'exécution) par la présence d'un mot clé ou en fonction du type ou de la valeur d'un mot-clé. Dans ce cas, `sd_prod` contient une fonction Python. La fonction reçoit en arguments les mots-clés de l'opérateur ou procédure et doit retourner le type du concept produit.

Entête du catalogue :

```
def operateur_prod( MCLE1 , MCLE2 , MCLE3 , **args):
    if args.get('__all__'):
        return (type1, type2, typ4)
    if  MCLE1 == 'VALEUR1':
        return type1
    if MCLE2 is not None:
        return type2
    if AsType(MCLE3) == type3:
        return type4
    ...
raise AsException("type de concept résultat non prévu")
```

Corps de la commande :

```
NOM_COMMANDE=OPER( nom=" NOM_COMMANDE ",
                   op=54, sd_prod= operateur_prod,...
```

Exemple :

```
def asse_matrice_prod(MATR_ELEM, **args):
    if args.get('__all__'):
       return (matr_asse_depl_r, matr_asse_depl_c,
               matr_asse_temp_r, matr_asse_pres_c)
   if AsType(MATR_ELEM) == matr_elem_depl_r : return matr_asse_depl_r
   if AsType(MATR_ELEM) == matr_elem_depl_c : return matr_asse_depl_c
   if AsType(MATR_ELEM) == matr_elem_temp_r : return matr_asse_temp_r
   if AsType(MATR_ELEM) == matr_elem_pres_c : return matr_asse_pres_c
   raise AsException("type de concept résultat non prévu")

ASSE_MATRICE=OPER(nom="ASSE_MATRICE", op=12,
                  sd_prod=asse_matrice_prod, ...
                 )
```

Dans ce cas, si le mot clé `MATR_ELEM` est de type `matr_elem_depl_r` alors le concept produit est du type `matr_asse_depl_r`. Sinon, si le mot clé `MATR_ELEM` est de type `matr_elem_depl_c` alors le concept produit est du type `matr_asse_depl_c`, etc.

Toute fonction `sd_prod` doit pouvoir être appelée avec l'argument `__all__=True`. Cela permet notamment à *AsterStudy* de savoir quels types la commande est susceptible de retourner sans lui fournir aucun mot-clé.

Elle doit donc commencer par le test « si `__all__=True` » et retourner dans ce cas, la liste de **tous les types possibles** pour le résultat.

9.2 Enrichir le concept produit

L'argument `reentrant` permet de préciser si le concept produit par un opérateur est créé ou réemployé puis enrichi. Dans ce dernier cas, pour signaler dans le fichier de commande qu'on réutilise un concept, l'argument `reuse` suivi du nom du concept sera présent.

Trois situations sont possibles :

`reentrant='n'`	La commande en cours produit nécessairement un nouveau concept. Exemple : `LIRE_MAILLAGE=OPER(sd_prod=maillage, reentrant='n',`

`reentrant='o'`	La commande modifie systématiquement un concept déjà existant.
	
	Exemple :
	`CALC_META=OPER(sd_prod=evol_ther,`
	`reentrant='o:RESULTAT',`
	
	Dans ce cas, la structure de donnée `evol_ther` doit obligatoirement être créée au préalable par l'opérateur de thermique pour être enrichi ici par la commande de post-traitement métallurgique.

`reentrant='f'`	Les deux situations sont possibles. C'est le cas des commandes calculant des évolutions (structures de données evol_***). On peut vouloir enchaîner un deuxième calcul transitoire derrière un premier et compléter la structure de données des nouveaux instants de calcul obtenus.
	
	Exemple :
	
	Dans le catalogue :
	`STAT_NON_LINE=OPER(sd_prod=evol_noli,`
	`reentrant='f:RESULTAT',`
	`reuse=SIMP(statut='f', typ=CO), …`
	
	Dans le fichier de commandes :
	`U=STAT_NON_LINE( . . . )`
	`U=STAT_NON_LINE( reuse=U, . . . )`

Il est nécessaire d'indiquer quel mot-clé fourni le concept qui sera enrichi. C'est ce qu'on indique après « `o:` » ou « `f:` » :

- `o:MAILLAGE` indique que l'objet enrichi est fourni par le mot-clé simple `MAILLAGE`.
- `o:ASSE:CHAM_GD` indique que l'objet enrichi est fourni par le mot-clé simple `CHAM_GD` sous le mot-clé facteur `ASSE`.
- `f:BETON|NAPPE:MATER` indique que l'objet enrichi est fourni par le mot-clé simple `MATER` soit sous le mot-clé facteur `BETON`, soit sous le mot-clé facteur `NAPPE`.

Remarque

Si la commande est réentrante, il est nécessaire d'ajouter dans le catalogue, le mot-clé reuse comme dans l'exemple avec `STAT_NON_LINE` ci-dessus.

Cette possibilité que l'on offre à l'utilisateur doit être mûrement réfléchie et doit rester une exception à la règle générale qui veut que l'on ne modifie pas un concept fourni en entrée. En effet, lorsqu'un concept est modifié, les concepts qui avaient été créés en l'utilisant (avant le changement) risquent de

perdre la cohérence qu'ils avaient avec lui. Cela peut donc conduire à une base de donnée incohérente.

Aujourd'hui les seules modifications de concepts autorisées sont des enrichissements : on ajoute de l'information sans modifier l'existant, ou la destruction complète du concept. La seule exception à cette règle est la factorisation en place des MATR_ASSE (opérateur FACTORISER), cette exception est justifiée par des problèmes d'encombrement de la mémoire.

10 Routine d'utilisation

10.1 Nom de la routine

L'OPxxxx est la routine qui réalise la commande associée. Le numéro de la routine opxxxx.f est choisi parmi les numéros libres. xxxx est un numéro codé sur quatre chiffres.

10.2 Les deux étapes

Le superviseur procède en 2 étapes :

- une 1ère étape : construction de l'arbre des objets python : jeu de commandes, commandes, mots clés, vérification syntaxique python, vérification de la cohérence avec le catalogue,
- une 2 ème étape : appel à l ' OPxxxx demande d'exécution des calculs

L'appel aux opérateurs, depuis le superviseur, se fait automatiquement en fonction de l'attribut op=xxxx renseigné dans le catalogue, par :

```
CALL OPxxxx (IER)
```

10.3 Récupération des arguments de la commande

Les arguments réels (ceux que l'utilisateur a écrit derrière les mots clés dans son fichier de commandes) sont récupérés par des requêtes faites au superviseur.
Il est conseillé de regrouper la lecture des mots-clés dans une routine appelée par l' OPXXXX (éventuellement dans l' OPXXXX lui-même), puis d'exécuter les calculs nécessaires.

- **Requêtes d'accès aux valeurs :**

Un ensemble de sous-programmes spécifiques à chaque type connu du superviseur est disponible :

GETVIS	récupération de valeurs entières,
GETVR8	récupération de valeurs réelles,
GETVC8	récupération de valeurs complexes,
GETVLS	récupération de valeurs logiques,
GETVID	récupération de concepts (leur nom),
GETVTX	récupération de valeurs textes,
GETLTX	récupération des longueurs des valeurs textes,
GETTCO	récupération des types d'un concept.

- **Requête d'accès au résultat :**

Le sous-programme GETRES permet d'obtenir le nom utilisateur du concept produit, le type du concept associé au résultat et le nom de l'opérateur ou de la commande.

Ces routines sont décrites dans [D6.03.01] - Communication avec le Superviseur d'exécution : routines GETXXX