To introduce a new order

Summary:

This document describes the method to introduce a new order (operator or procedure) into *Code_Aster* . It describes the drafting with the format “python” of the catalogue of the order and the routine FORTRAN associated.

Contents

1 Introduction

To introduce a new order into *Code_Aster* , it is necessary:

- to write the catalogue associated with this order (See it paragraph Drafting of the catalogue of order),
- to write the routine FORTRAN OPxxxx associated (See it paragraph To typify the concept produce and to enrich it).

Remarks on source files

One finds the files which describe the catalogue in the repertoire code_aster/Catastrophes.
In order to prepare future developments, there is a indirection on this level towards Legacy (current) or Language (future). When one speaks about DataStructure.py , for the current version, the file imports objects defined in Legacy/DataStructure.py.

2 Vision user of an order

Let us take as example the order AFFE_MATERIAU who allows to affect on a grid of the material characteristics. Here a possible use of this order in the command file provided by the user of *Code_Aster* :

```
cham = AFFE_MATERIAU  (  GRID   =  e-mail,
                         AFFE      =  _F (  ALL  = 'YES',
                                            MATER = steel  )
                      )
```

During the use of an order, it appears:

- the name “user” of the concept produced by the order: cham
- the name of the order: AFFE_MATERIAU
- keywords factors: AFFE
- simple keywords: ALL , MATER , GRID
- names “users” of concepts arguments: steel , e-mail
- values of the simple type (whole, real, text,…) only or in list: ' YES '

From the point of view user, by writing a name on the left sign “=” order, one assigns this name to the result of the order.

This “name user” is affected a produced concept (or structure of data) calculated by the operator and whose type is given by the supervisor. The type of the produced concept is defined in the catalogue of the order (See it paragraph Drafting of the catalogue of order).

For example cham is the name user of the result of the order and with this name the concept of the type is associated cham_mater.

3 Drafting of the catalogue of order

To introduce a new order, it is necessary to create an associated catalogue in which will be indicated:

- the name of the order,
- its description in a few words,
- the nature of the order: operator (production of concept), procedure (not of produced concept), macro-order,
- the number of the routine FORTRAN associated with this order (See To define the link between the catalogue and the program FORTRAN associated).
- for the produced concept:
  - rules of determination of the type of the concept (See To typify the concept produce and to enrich it),
  - the possibility of re-use (D-entering character).
- for the keywords (See To define the attributes of the simple keywords and To exclude or gather keywords: argument rules),
  - if their presence is optional or obligatory, if they are excluded between them…
  - the type of the argument,
  - the number of expected arguments of this type,
  - the value by default (if there is one of them),
  - the list of the acceptable values (possibly),
  - the beach of the acceptable values (possibly), if one expects an entirety or a reality,
- for the keywords factors (See Case of the keywords factors):
  - if their presence is optional or obligatory (or present by default),
  - the minimum and maximum number of possible occurrences,
- blocks: logical regrouping of keywords when conditions on other keywords are satisfied (See Blocks).

Note:

One will not speak in this document about the introduction about a new macro-order (see [D5.01.02] - To introduce a new macro-order)

The language used to write this catalogue is the language interpreted Python: the comments are written behind the character "#", one sees keywords (identifying follow-ups of the character "="), brackets, commas to separate the keywords…

Let us take again the example of the order AFFE_MATERIAU , the catalogue - i.e. the description of the order provided by sound **developer** - associated is:

```
AFFE_MATERIAU = OPER (
       nom= " AFFE_MATERIAU", op=6, sd_prod=cham_mater,
       fr=tr ("Assignment of material characteristics to a grid"),
       reentrant=',
       GRID   =     SIMP (statut=' o', typ=maillage),
       MODEL      =     SIMP (statut=' f', typ=modele),
       AFFE       =     FACT (statut=' o', min=1, max=' ** ',
          rules = (UN_PARMI ( 'ALL', 'GROUP_MA', 'MESH',
                              'GROUP_NO', 'NODE'),),
          ALL    =SIMP (statut=' f', typ=' TXM', into= ("YES",)),
          GROUP_MA  =SIMP (statut=' f', typ=grma, max=' ** '),
          MESH   =SIMP (statut=' f', typ=ma, max=' ** '),
          GROUP_NO  =SIMP (statut=' f', typ=grno, max=' ** '),
          NODE       =SIMP (statut=' f', typ=no, max=' ** '),
          MATER      =SIMP (statut=' o', typ=mater),
          TEMP_REF  =SIMP (statut=' f', typ=' R', defaut= 0.E+0),
       ),
```

)

The syntax of order is described using the following arguments. Their significance specifies will be given throughout the document.

OPER/PROC/MACRO	To specify the type of order (production of one, zero even several concepts in the case of macros)
name	To indicate the name seen by the user to indicate the order
op	To specify the number of the routine FORTRAN high level associated with the order
sd_prod	To define the type of produced concept
rules	To define the logical rules of pairing or exclusion of keywords
UN_PARMI/…	To define a list of keywords among which the data must be exactly once.
Fr	To describe in a sentence (in French) the role of the order, it is the contents of the bubble of assistance displayed by AsterStudy. It is imperative to use the function tr () in order to allow the translation of the text in other languages.
reentrant	To specify if the order creates a new concept (value 'N'), modifies an existing concept (value 'O'), or potentially both (value 'F')
SIMP	To specify a simple keyword of the order
FACT	To specify a keyword ratio control.
BLOCK	To define a block of keywords of which the appearance is subjected to a "condition".

One can break up the writing of the catalogue of order according to the following stages:

- **To specify the type of the produced concept** : (when there exists, i.e. for an ordering of type **OPER**)

To specify the type of the produced concept of an operator, the argument should be used sd_prod (structure of produced data). For example, the assignment sd_prod=cham_mater indicate that cham_mater is the type of the produced concept of the operator AFFE_MATERIAU.

In the case of a procedure, it does not have there a produced concept (and thus not of argument **sd_prod** in the catalogue). For example CALC_G is an order whose type of the produced concept is table_sdaster , whereas IMPR_RESU is a procedure without produced concept:

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

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

If the type of the produced concept depends on the arguments of the operator, one will consult To typify the produced concept .

If the produced concept can be a re-used concept and nouveau riche, one will indicate it by informing the argument reentrant (See To enrich the produced concept).

- **To define the name of the order** :

He is written on the left sign "=" in the catalogue, on the right in the command file of the user.

Generally the prefix indicates the action, the suffix the treated concept (for example AFFE_MATERIAU). Let us note some prefixes frequently employed:

AFFE	assignments on the grid or the model,
CHALLENGE	definitions of objects which are not fields,
CALC	orders calling the routine CALCULATION and producing fields of sizes.

The name of an order should not exceed 16 characters. This name is that used by the user in a command file.

- **To define the number of the routine FORTRAN carrying out the order** : (See To define the link between the catalogue and the program FORTRAN associated)
- **To describe the various keywords** : (See the §5, 6 and 7) . It is the heart of the catalogue.
- **To close** the open bracket after the definition **OPER/PROC/MACRO**.

4 To define the link between the catalogue and the program FORTRAN associated

The argument op the call to the routine allows FORTRAN OPxxxx who carries out the task of the order (See To typify the concept produce and to enrich it). The argument of op is a strictly positive entirety ranging between 1 and 199. The number is allotted by the team codes (cf [A2.01.02]).

On the example considered the routine OP0006 will be called during the execution of the order AFFE_MATERIAU .

5 To define the attributes of the simple keywords

General syntax to declare a keyword simple is:

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

Among the attributes attached to a keyword, only statute and typ are obligatory for any simple keyword:

- **The statute**

The definition of the statute by the attribute `statute` is obligatory.

The recognized statutes are only:	
`'O'`	Obligatory: in this case the keyword will have obligatorily to appear in the body of call of the ordering of the user (except if this keyword is under a keyword optional factor in which case the simple keyword is obligatory as soon as the keyword factor appears).
`'F'`	Optional: in the contrary case.
`'it`	Hidden: the keyword neither is written in the file of message, nor visible in AsterStudy. To avoid this use. In general, is used it to transmit a parameter which can depend on other keywords and which would be complicated to deduce in FORTRAN.
`'of`	Defect: Mean that the keyword will be present by default. Has interest only for the keywords factors.

- **The type**

The declaration of the type by the attribute `typ` is obligatory.

The recognized types are:	
`typ = 'I '`	for the entireties
`typ = 'R '`	for realities
`typ = 'C '`	for the complexes
`typ = Type_de_concept (without dimensions!)`	for the concepts
`typ = 'TX M '`	for the texts
`typ = 'L '`	for logics
`typ = UnitType ()`	for the logical units

Remarks on the concepts

The type of expected concept is a kind of concept created by another order that the pending order. It appears among the list of the concepts imported in the catalogue of declaration of the concepts (`code_aster/Catastrophes/DataStructure.py`)

Notice for the logical units

For the logical units (of type `UnitType ()`), it is necessary to declare an attribute additional `inout` who is worth `'in'` if the file is read or `'out'` if he is written by the order.

The type of the expected concept is not necessarily single. It can be a list, which means that one or the other of the types will be produced. This list is declared as follows:

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

The documentary syntax of this example is:

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

- **Value by default for a keyword**

It is possible to assign a value by default to a keyword not receiving an argument of type “concept”. The declaration is done by the argument defect

Examples :

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

FILE       =SIMP (statut=' f', typ=' TXM', defaut= " RESULT”),
```

- **List of acceptable values:**

So that the supervisor controls the validity of the contents of certain keywords, it is possible to declare the values of the expected arguments. This declaration is done by the argument into

Examples :

```
INFORMATION   =SIMP (statut=' f', typ=' I', defaut= 1, into=
(1.2)),
```

The keyword INFORMATION is optional, its value by default is 1 and the only accepted values are 1 and 2. Documentary syntax is:

♦ INFORMATION: / 1 [DEFECT]
/ 2

- **Many expected values:**

Arguments min and max allow to control the length of the list of the arguments expected behind the simple keyword. By default, if nothing is specified in the catalogue, one expects one and only one value behind a simple keyword (max = 1). Attention, to declare min = 1 do not bring anything and does not amount especially making the keyword obligatory. If a potentially unlimited number of elements is expected, syntax is max=' ** '.

Examples :

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

The user can enter as many here names of meshs it wishes.

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

A vector here is expected (list of exactly three realities).

- **Beach of acceptable values**

For the entireties and realities, one can specify the values allowed minimum and/or maximum:

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

On this example, NAKED must belong to the interval [-1, 0.5]. The values given by the two arguments are included in the interval.

- **More complicated criteria**

In addition to the beaches of values and the cardinal of the list, one can impose more complicated criteria on the value provided by the user, they are them validators, defined in Noyau/N_VALIDATOR.py.

One can program the new ones, according to the needs. The principal ones validators are:

PairVal ()	the provided entireties must be even
NoRepeat ()	checking of the absence of doubled blooms in a list
Compulsory (list)	checking that all elements of list were provided
LongStr (low, high)	checking length of a character string
OrdList (order)	checking which a list is increasing or decreasing
AndVal (val1, val2,…)	condition AND logic enter them validators list
OrVal (val1, val2,…)	condition OR logic enters them validators list

6 Case of the keywords factors

The keywords factors are obligatory or optional. It is possible to control the minimum numbers and maximum of occurrences of a keyword factor.

The declarations are done thanks to the keyword FACT

- **The statute**

It is related to the keyword factor .

The recognized statutes are only:

'O'	Obligatory
'F'	Optional
'D'	Optional but used by default, i.e optional for the user with the seizure but obligatory for the operation of the code. The values by default of the simple keywords must define the syntax of all the keyword factor seen of the code when the user does not inform anything. The user does not need to inform the keyword factor and his simple keywords so that there exists and is visible of the supervisor with the execution.

- **The number of occurrences**

As for the simple keywords, the arguments min and max allow to specify the expected occurrences of the keywords factors. If nothing is put, the situation by default is max=1, the keyword factor is then not répétable.

Examples:

```
MCFACT = FACT ( statute = ' f', min =3, max =3,…)
```
the keyword factor is obligatory and must appear three times exactly.

```
MCFACT = FACT ( statute = ' f', max=' ** ',…)
```

the keyword factor is optional but can appear as many times as one wants.

MCFACT = FACT (statute = ' of, max =1,…)
the keyword factor is optional and not répétable but if the user does not specify it, he nevertheless is taken into account and the values of the simple keywords (under the keyword factor) are affected by default.

7 To exclude or gather keywords: argument rules

The use in the catalogues of orders of the argument rules described below and of blocks (following paragraph) allows to entirely reproduce the logic of sequence of the keywords described in the paragraph syntax of the documentation of use. There should thus be only very little checks of syntax (tests on the presence or the contents of keywords) on the level of the routines FORTRAN op0nnn.f .

Rules, present under the argument rules , which follows make it possible to ensure a coherence on the simultaneous presence of the keywords of the order. Behind these definitions of rules (EXCLUDED, UN_PARMI, TOGETHER,…), one finds a list of keywords which are, either of the simple keywords (under the same keyword factor), or of the keywords factors. In the continuation of this paragraph one will use nothing any more but the term “keyword”.

EXCLUDED	mc1, mc2,…, mcn The keywords are excluded mutually.
UN_PARMI	mc1, mc2,…, mcn One of the keywords of the list must be obligatorily present and only one.
TOGETHER	mc1, mc2,…, mcn If one of the keywords is present, all must apparaîtrent.
AU_MOINS_UN	mc1, mc2,…, mcn It is necessary that at least a keyword among the list is present. It is licit to have several present of them.
PRESENT_PRESENT	mc1, mc2,…, mcn If the keyword mc1 is present then the keywords mc2,…, mcn must be present.
PRESENT_ABSENT	mc1, mc2,… mcn If the keyword mc1 is present then the keywords mc2,…, mcn must miss.

Remarks

PRESENT_PRESENT *is different from* TOGETHER *since for* PRESENT_PRESENT mc2 *can be present, without* mc1 *is.*
PRESENT_ABSENT *east differ from* EXCLUDED *since for* PRESENT_ABSENT mc2,…, mcn *can be present units if* mc1 *is absent.*

```
rules = ( UN_PARMI ('NODE', 'GROUP_NO', 'MESH'),
             PRESENT_PRESENT ('MESH', 'NOT'),),

NODE      =SIMP (…),
MESH     =SIMP (…),
NOT      =SIMP (…),
GROUP_NO  =SIMP (…),
```

The supervisor checks that the user gave one and only one of the keywords well among NODE , GROUP_NO and MESH and, if it gave MESH , that NOT that is to say also present.

Caution

Keywords handled in the argument `rules` must be defined on the same level (i.e. with the principal root of the order, under the same keyword factor or the same block). Several definitions of `rules` can be present in the same catalogue, with the principal root of the order or under keywords factors.

8 Blocks

The blocks are appeared as a regrouping of keywords. They allow two things:

- to translate in the catalogue of the order of the logical rules relating to the value or the type of the contents of the simple keywords; whereas conditions under the argument `rules` relate only to the presence or the absence of the keywords. One can thus gather keywords together or affect attributes to them (value by defect…) individuals under certain conditions.
- to gather the keywords by families for more clearness in AsterStudy. These keywords will be then visible with the user only when the condition is met.

Examples:

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

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

  will b_mult_front =BLOC (condition = "equal_to ('METHODE',
'MULT_FRONT')",
                      fr=tr ("Parameters of the frontal method multi"),
                      RENUM=SIMP (statut=' f', typ=' TXM', defaut= "
MDA",
                                 into= ("MANDELEVIUM", "MDA",
"MONGREL")),
                       ),
  b_ldlt         =BLOC (condition = "equal_to ('METHODE', 'LDLT')",
                      fr=tr ("Parameters of method LDLT"),
                      RENUM=SIMP (statut=' f', typ=' TXM', defaut= "
RCMK",
                           into= ("RCMK", "WITHOUT")),
                           TAILLE=SIMP (statut=' f', typ=' R', defaut=
400. ),
                                 ),
                       ),
```

The blocks are named by the developer. Their name must start with " `b_` ". In the example, if `METHOD` is worth `MULT_FRONT`, then the optional simple keyword `RENUM` will appear with three possible values declared under `into` . So on the other hand `METHOD` is worth `LDLT` , the same keyword will be present but with two different possible values; moreover it will be then possible to inform the simple keyword `SIZE` . These respective keywords and their attributes will appear in AsterStudy only after the user will have affected a value with the simple keyword `METHOD` .

```
b_nomdubloc=BLOC (
    condition="exists ('MOTCLE1') and is_type ('KEYWORD2') ==grma",
    …
```

One shows on this example that the condition can be multiple (articulated by `however/and`) and can also relate to the presence of the keyword (`exists ('MOTCLE1')`) or the type of what it contains (`is_type ('MOTCLE2') == grma`).

The condition is an expression Python (provided in the form of a character string) which is evaluated at the level immediately higher than the block itself.

Conditions of blocks do not have to handle the keywords directly but call on functions which are based on the name of the keywords.
These functions are the following ones:

- `exists ('KEYWORD')` is checked if `KEYWORD` exist, i.e. was informed by the user.
  Example: `exists ('TYPE_MATR_TANG')`.
- `is_in ('KEYWORD', VALUES)` is checked if the intersection between the values of `KEYWORD` and `VALUES` is nonempty.
  Example: `is_in ("CRIT_COMP", ('EQ', '')`.
- `equal_to ('KEYWORD', VALUE)` is strictly identical to `is_in` but more natural when `KEYWORD` and `VALUE` contien nent that a value.
  Example: `equal_to ('METHOD', 'MUMPS')`.
- `I s_type ('KEYWORD' )` turn over the type of `KEYWORD` .
  Example: `is_type ('FUNCTION') in (function, fonction_c)`.
- `valley ue ('KEYWORD', défaut=' ' )` turn over the value of `KEYWORD` if there exists, if not turns over the value `defect` who is a character string empties by default .
  Example: `been worth ("RELATION") .startswith ('META_')`.
- `length (' KEYWORD ' )` turn over the number of provided values for `KEYWORD` . Turn over 0 if one cannot calculate the length of `valley ue (' MOTCLE' )` or if there does not exist.
  Example: `length('FREQ') > 2`.
- `less_than ('KEYWORD', VALUE )` is checked if the value of `KEYWORD` is lower `VALUE` . Turn over `False` if `KEYWORD` do not exist.
  Example: `less_than('COEF_MULT', 0)`.
- `greater _than ('KEYWORD', VALUE )` is checked if the value of `KEYWORD` is higher `VALUE` . Turn over `False` if `KEYWORD` do not exist.
  Example: `greater_than('COEF_MULT', 0)`.

***Caution* :**

Lbe keywords handled in condition `BLOCK` must be on the same level as the block itself in the tree structure defined by the keywords factors and the blocks. Several keywords `BLOCK` can be present in the same catalogue, with the principal root of the order, under keywords factors or in other blocks. In the example above, the keywords simple `RENUM` and `TAILLE_BLOC` are on the same level, lower than that of `METHOD` , `will b_mult_front` and `b_ldlt` , him even lower than that of the keyword factor `SOLVEUR` . The conditions tested in the two blocks relate thus only to the simple keyword `METHOD` of level immediately higher than the blocks themselves.
IL is necessary to pay attention to the possible conflicts when the same keyword is present under two different blocks. The conditions of activation of the two blocks must then be excluded. It is the case of the example above with the simple keyword `RENUM` : there cannot be conflict since them two conditions `METHODE=' MULT_FRONT '` and `METHODE=' LDLT '` cannot be simultaneously satisfied. In a case where the conditions would be satisfied at the same time, an error would occur with the execution.

9 To typify the concept produce and to enrich it

9.1 To typify the produced concept

The argument `sd_prod` allows to carry out the declaration of the type of produced concept. If the order always produces the same structure of data some is the context, `sd_prod` is followed by type of corresponding concept, already declared in the catalogue of declaration of the concepts:

DataStructure.pthere and SD/co_xxx.py. All the concepts being able to be produced by orders and/or used in keywords are declared in this file.

Example:

In the catalogue of the order:

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

The type table_sdaster is defined in SD/co_table.py .

In certain cases the developer wants that the operator produces a concept whose type is dynamically given (i.e with the execution) by the presence of a keyword or according to the type or value of a keyword. In this case, sd_prod contains one function Python. The function receives in arguments the keywords of the operator or procedure and must turn over the type of the produced concept.

Heading of the catalogue:

```
def operateur_prod (MCLE1, MCLE2, MCLE3, ** arguments):
    yew args.get ('__all'):
        return (type1, type2, typ4)
    yew  MCLE1 == 'VALEUR1':
        return type1
    yew MCLE2 is not Nun:
        return type2
    yew AsType (MCLE3) == type3  :
        return type4
    …
raise AsException ("standard of concept Résultat not Préconsidering")
```

Body of the order:

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

Example:

```
def asse_matrice_prod (MATR_ELEM, ** arguments):
    yew args.get ('__all'):
       return (matr_asse_depl_r, matr_asse_depl_c,
               matr_asse_temp_r, matr_asse_pres_c)
   yew AsType (MATR_ELEM) == matr_elem_depl_r: return matr_asse_depl_r
   yew AsType (MATR_ELEM) == matr_elem_depl_c: return matr_asse_depl_c
   yew AsType (MATR_ELEM) == matr_elem_temp_r: return matr_asse_temp_r
   yew AsType (MATR_ELEM) == matr_elem_pres_c: return matr_asse_pres_c
   raise AsException ("standard of concept Résultat not Préconsidering")

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

In this case, if the keyword MATR_ELEM is of type matr_elem_depl_r then the produced concept is of the type matr_asse_depl_r. If not, if the keyword MATR_ELEM is of type matr_elem_depl_c then the produced concept is of the type matr_asse_depl_c, etc.

Any function sd_prod must be able to be called with the argument __all__=True. That allows in particular *AsterStudy* of knowing which types the order is likely to turn over without him to provide any keyword.
It must thus start with the test "if __all__=True " and to turn over in this case, the list of **all possible types** for the result.

9.2 To enrich the produced concept

The argument reentrant allows to specify if the concept produced by an operator is created or employed again then enriched. In this last case, to announce in the command file which one re-uses a concept, the argument reuse followed by the name of the concept will be present.

Three situations are possible:

reentrant='	The pending order produces necessarily a new concept. Example: LIRE_MAILLAGE=OPER (sd_prod=maillage, reentrant=',
reentrant=' o'	The order modifies an already existing concept systematically. Example: CALC_META=OPER (sd_prod=evol_ther, reentrant=' O: RESULTAT', In this case, the structure of data evol_ther must obligatorily be created as a preliminary by the operator of thermics to be enriched here by the metallurgical ordering of postprocessing.
reentrant=' f'	The two situations are possible. It is the calculating case of the orders of the evolutions (structures of data evol_ ***). One can want to connect the second transitory calculation behind a first and to supplement the structure of data of the new moments of calculation obtained. Example: In the catalogue: STAT_NON_LINE=OPER (sd_prod=evol_noli, reentrant=' F: RESULTAT', reuse=SIMP (statut=' f', typ=CO),… In the command file: U=STAT_NON_LINE (…) U=STAT_NON_LINE (reuse=U,…)

It is necessary to indicate which keyword provided the concept which will be enriched. It is what one indicates after " O: " or " F: ":

- O: GRID indicate that the enriched object is provided by the simple keyword GRID.
- O: ADZE: CHAM_GD indicate that the enriched object is provided by the simple keyword CHAM_GD under the keyword factor ADZE.
- F: CONCRETE|TABLECLOTH: MATER indicate that the enriched object is provided by the simple keyword MATER that is to say under the keyword factor CONCRETE, that is to say under the keyword factor TABLECLOTH.

Notice

If the order is rééntrante, it is necessary to add in the catalogue, the keyword reuse as in the example with STAT_NON_LINE above.

This possibility that one offers to the user must be maturely considered and must remain a general exception to the rule which wants that one does not modify a provided concept as starter. Indeed, when a concept is modified, the concepts which had been created by using it (before the change) risk to lose the coherence which they had with him. That can thus lead to a base of incoherent data.

Today the only modifications of concepts authorized are enrichments: one adds information without modifying existing it, or the complete destruction of the concept. The only exception to this rule is factorization in place of MATR_ASSE (operator TO FACTORIZE), this exception is justified by problems of obstruction of memory.

10 Routine of use

10.1 Name of the routine

OPxxxx is the routine which carries out the associated order. The number of the routine opxxxx.f is selected among the free numbers. xxxx is a number coded on four digits.

10.2 Two stages

The supervisor proceeds in 2 stages:

- one 1era stage: construction of the tree of the python objects: order, command set, keywords, syntactic checking python, checking of coherence with the catalogue,
- one 2 ème stage: call to L ' OPxxxx request for execution of calculations

The call to the operators, since the supervisor, is done automatically according to the attribute op=xxxx informed in the catalogue, by:

```
OPxxxx CAL (1ST)
```

10.3 Recovery of the arguments of the order

The real arguments (those which the user wrote behind the keywords in his command file) are recovered by requests made to the supervisor.
It is advised to gather the reading of the keywords in a routine called by OPXXXX (possibly in OPXXXX itself), then to carry out calculations necessary.

- **Requests of access to the values:**

A set of subroutines specific to each known type of the supervisor is available:

GETVIS	recovery of whole values,
GETVR8	recovery of actual values,
GETVC8	recovery of complex values,
GETVLS	recovery of logical values,
GETVID	recovery of concepts (their name),
GETVTX	recovery of values texts,
GETLTX	recovery lengths of the values texts,
GETTCO	recovery of the types of a concept.

- **Request of access to the result:**

The subroutine GETRES allows to obtain the name user of the produced concept, the type of the concept associated with the result and the name of the operator or order.

These routines are described in [D6.03.01] - Communication with the Supervisor of execution: routines GETXXX