To introduce a new command

Summarized:

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

Contents

1 Introduction

to introduce a new command into *the Code_Aster* , it is necessary:

- to write the catalog associated with this command (See the § Drafting of the catalog of command),
- to write associated routine FORTRAN OPxxxx (See the § and to enrich it To typify the product concept).

We will speak here only about the first two points.

2 Vision user of a command

Let us take as example the command AFFE_MATERIAU which makes it possible to affect on a mesh of the characteristics of material. Here a possible use of this command in the command file provided by the user of *Code_Aster* :

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

During the use of a command, it appears:

- the name "user" of the product concept by the command: cham
- the name of the command: AFFE_MATERIAU
- key keys factors: AFFE
- of the single-ended spanner keys: TOUT , MATER , MAILLAGE
- of the names "users" of concepts arguments: steel , mail
- of the 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 "=" of the command, one assigns this name to result of the command.

A this "name user" is affected a product concept (or data structure) calculated by the operator and whose type is given by the supervisor. The type of the product concept is defined in the catalog of the command (See the § Drafting of the catalog of command).

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

3 Drafting of the catalog of command

to introduce a new command, it is necessary to create an associated catalog in which will be indicated:

- the name of the command,
- its description in a few words,
- the category of ranking by families of the commands for display in EFICAS,
- nature of the command: operator (production of concept), procedure (not of product concept), macro-command,
- the number of routine FORTRAN associated with this command (See the § To define the restrain between the catalog and associated FORTRAN program).
- for the product concept:
  - rules of determination of the type of the concept (See the § and to enrich it To typify the product concept),
  - the possibility of re-use (D-entering character).
- for the keywords (See the § To define the attributes of the simple keywords and gather keywords: argument rules the use in),
  - if their presence is optional or compulsory, 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 integer or a reality,
- for the keywords factors (See the § the keywords factors the keywords):
  - if their presence is optional or compulsory (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 the § the blocks are appeared).

Note:

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

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

of the command Take again the example AFFE_MATERIAU , the catalog - i.e. the description of the command provided by its **developer** - associated is:

```
    AFFE_MATERIAU = OPER     ( nom= " AFFE_MATERIAU",  op=6,
sd_prod=cham_mater,
          fr= " Assignment of characteristics of materials to a mesh",
          reentrant=',  UIinfo= {"groups": ("Modelization",)},
             MAILLAGE=       SIMP  (statut=' o', typ=maillage),
             MODELE=            SIMP  (statut=' f', typ=modele),
             AFFE=           FACT  (statut=' o', min=1, max=' ** ",
                 rules = (UN_PARMI ( "TOUT", "GROUP_MA", "MESH",
                                     "GROUP_NO", "NOEUD"),),
                 TOUT=SIMP      (statut=' F", typ=' TXM', into= ("YES",)),
                 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),
                                ),
```

)

the syntax of command is described using the following arguments. Their meaning specifies will be given throughout the document.

OPER/PROC/MACRO	to specify the type of command To indicate (production of one, zero even several concepts in the case of macros
)	name the name seen by the user to indicate the command
op	to specify the number of high level routine FORTRAN associated with the command
sd_prod	to define the type of product 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	of the command to describe in a sentence (in French) the role, it is the contents of the bubble of assistance displayed by EFICAS
UIinfo	Useful only for the displays in EFICAS, to specify the family of ranking of the command: Postprocessings, Modelization…
reentrant	to specify if the command creates a new concept (value "N"), modifies an existing concept (value "O"), or potentially the two (value "F")
SIMP	to specify a key word simple of the command
FACT	to specify one factor key word of the command.
BLOCK	to define a block of keywords of which the appearance is subjected to a "condition ".

One can break up the writing of the catalog of command according to the following stages (see [D1.02.01] - §1.2: Instruction manual of the agla):

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

to specify the type of the product concept of an operator, it is necessary to use the argument sd_prod (produced data structure). For example, the assignment sd_prod=cham_mater indicates that cham_mater is the type of the product concept of operator AFFE_MATERIAU.

In the case of a procedure, it does not have there a product concept (and thus not of argument **sd_prod** in the catalog). For example CALC_G is a command whose type of the product concept is table_sdaster , whereas IMPR_RESU is a procedure without product 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 product concept depends on the arguments of the operator, one will consult the §the argument sd .

If the product concept can be a re-used concept and nouveau riche, one will indicate it by informing the argument reentrant (See the §the argument reentrant makes it possible).

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

He is written on the left sign "=" in the catalog, 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 mesh or the model,
DEFI	definitions of objects which are not fields,
CALC	commands calling the routine CALCUL and producing fields of variables.

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

- **To define the number of routine FORTRAN carrying out the command** : (See the §To define the restrain between the catalog and associated FORTRAN program)
- **To describe the various keywords** : (See the §5,656 and 7) . It is the heart of the catalog.
- **To close** the open bracket after definition **OPER/PROC/MACRO**.

4 To define the restrain between the catalog and associated FORTRAN program

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

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

5 To define the attributes of the simple keywords

general syntax to declare a key word simple is:

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

Among the attributes attached to a key word, alone statute and typ are compulsory for any simple key word:

- **The statute**

the definition of the statute by the attribute `statute` is compulsory.

The recognized statutes are only:	
"O"	Compulsory: in this case the key word will have of the command to appear obligatorily in the body of call of the user (except if this key word is under a key word optional factor in which case the simple key word is compulsory as soon as the key word factor appears).
"F"	Optional: in the contrary case.

- **The type**

the declaration of the type by the attribute `typ` is compulsory.

The recognized types are:	
`typ = "I"`	for the integers
`typ = "R"`	for realities
`typ = "C"`	for the complexes
`typ = Type_de_concept (without dimensions!)`	for the concepts
`typ = "TX"`	for the texts
`typ = "L"`	for logics

Remarks on the concepts:

The type of expected concept is a kind of concept created by another command that the pending order. It appears among the list of the concepts defined in the catalog of declaration of the concepts (`accas.capy` and all the files SD `/co_xxxx.py`) 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] Default value
```

- **for a key word It is**

possible to assign a default value to a key word not receiving an argument of type “concept”. The declaration is done by the argument default `Examples`

: accuracy

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

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

- **of acceptable values: So that**

the supervisor controls the validity of the contents of certain key words, it is possible to declare the values of the expected arguments. This declaration is done by the argument into Examples

: INFO

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

INFO is optional , its default value is 1 and the only accepted values are 1 and 2. Documentary syntax is: ♦ INFO

```
:    /1     [DEFAULT       ]/2
              Number of values
```

- **waited: The arguments**

min and max make it possible to control the length of the list of the arguments expected behind the simple key word. By default, if nothing is specified in the catalog, one expects one and only one value behind a simple key word (max = 1). Attention , to declare min = 1 does not bring anything and does not amount especially making the key word compulsory. If a potentially unlimited number of elements is expected, syntax is max=' ** ". Examples

: NET

```
=SIMP       (statut=' F", typ=ma, max=' ** "), L" user
```

can enter as many here names of meshes 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 integers and the real, one can specify the values allowed minimum and/or maximum: NU=SIMP

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

example, NU 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 the validators , defined in Core /N_VALIDATOR.py. One can

program the new ones, according to the needs. Principal the validators is: PairVal (

) the provided	integers must be even NoRepeat

() checking	of the absence of duplicates in a Compulsory list
(list) checking	that all the 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 (val
1, val2,…) condition	AND logic enter the validators of the list OrVal (val
1, val2,…) condition	OR logic between the validators of the list Case of

6 the keywords factors the keywords

factors is compulsory or optional. It is possible to control the minimum numbers and maximum of occurrences of a key word factor. The declarations

are done thanks to key word FACT the statute

- **It is related**

to the key word factor . The recognized

statutes are only: “O”

Compulsory	“F”
Optional	“D”
Optional	but used by default, i.e optional for the user with the seizure but compulsory for the operation of the code. The values by default of the simple keywords must define the syntax of all the key word factor seen of the code when the user does not inform anything. The user does not need to inform the key word factor and his simple keywords so that there exists and is visible of the supervisor with the execution. The number D

- **“occurrences As for**

the simple keywords, the arguments min and max make it possible to specify the expected occurrences of the keywords factors. If nothing is put, the situation by default is max=1, the key word factor N” is then not répétable. Examples:

MCFACT = FACT

(statute = “ F”, min =3, max =3, …) the key word
factor is compulsory and must appear three times exactly. MCFACT = FACT

(statute = “ F”, max= “** “ ,…) the key word
factor is optional but can appear as many times as L” one wants. MCFACT = FACT

(statute = ” of, max =1,…) the key word
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 key word factor) are affected by default. To exclude or

7 gather keywords: argument rules the use in

the catalogs of commands of the argument rules describes below and of the 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 routines FORTRAN op 0nnn.f. The rules , present

under the argument rules, which follow make it possible of the command to ensure a coherence on the simultaneous presence of the keywords. 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 key word factor), or of the keywords factors. In the continuation of this paragraph one will use nothing any more but the term “key word”. EXCLUDED mc1, mc

2,…, 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 mc
1, mc2,…, mcn	It is necessary that at least a key word among the list is present. It is licit to have several present of them. PRESENT_PRESENT
mc1, mc2,…, mcn	If the key word mc1 is present then the keywords mc2,…, mcn must be present. PRESENT_ABSENT
mc1, mc2,… mcn	If the key word mc1 is present then the keywords mc2,…, mcn must miss . Remarks PRESENT

_PRESENT

is different *from GROUP since* for PRESENT *_PRESENT* mc2 can be present , *without mc1 being it. PRESENT _ABSENT*
east differ *from EXCLUDED since* for *PRESENT_ABSENT* mc2,…, mcn can be *present sets if mc1 is absent* . rules = *(UN_PARMI*

```
("NOEUD   ", "GROUP_NO", "MESH"), PRESENT_PRESENT
             ("MESH", "POINT"),), NOEUD =SIMP (

…)      , MESH =SIMP (
…)     , POINT =SIMP (
…)      , GROUP_NO =SIMP
(…)   , the supervisor
```

checks that the user gave one and only one of the key keys well among NOEUD, GROUP_NO and MESH and, if it gave MESH, that POINT is also present . Caution:

The keywords

handled in the argument rules must be defined on the same level (i.e. with the principal root of the command, under the same one factor key word or the same block). Several definitions of rules can be present in the same catalog, with the principal root of the command or under keywords factors. The blocks

8 the blocks are appeared

as a regrouping of key words. They allow two things: of the command to translate

- in the catalog logical rules relating to the value or the type of the contents of the simple key words; whereas the conditions under the argument rules relate **only** to the presence or the absence of the keywords. One can thus gather keywords together or affect to them attributes (defaults…) individuals under certain conditions. to gather
- the keywords by families for more clearness in EFICAS. These key words will be then visible with the user only when the condition is met. Examples: solver

=FACT (

```
statut=' of, min=1, max=1, METHODE=SIMP (statute

  = ' f', typ=' TXM', defaut= " MULT_FRONT", into= ("MULT_FRONT
               ", "LDLT")), b_mult_front=BLOC

  (condition  = "METHODE == "MULT_FRONT" ", fr= " Parameters
                    of the frontal method multi", RENUM=SIMP (statute
                    = ' f', typ=' TXM', defaut= " MDA", into= ("MD",
"MDA
                                ", "METIS")),), b_ldlt=BLOC
                     (condition
          = "METHODE == "LDLT" ", fr= " Parameters
                    of method LDLT", RENUM=SIMP (statute
                    = ' f', typ=' TXM', defaut= " RCMK", into= ("RCMK",
"
                         SANS")), TAILLE=SIMP (statute
                         = ' f', typ=' R', defaut= 400. ),),), the
blocks
```

are named by the developer. Their name must start with "b_ ". In the example , if METHODE is worth MULT_FRONT , then key word simple optional RENUM will appear with three possible values declared under into. So on the other hand METHODE is worth LDLT, the same key word will be present but with two different possible values; moreover it will be then possible to inform the simple key word CUTS. These key keys and their respective attributes will appear in EFICAS only after the user will have affected a value with key word simple METHODE. b_nomdubloc =BLOC

```
(condition= " MOTCLE1! = Nun gold Astype (MOTCLE2) == grma", One shows
on
```

this example that the condition can be multiple (pin-jointed by gold/and) and can also relate to the presence of key word (MOTCLE1! = Nun) or the type of what it contains (Astype (MOTCLE2) == grma). The condition is
a statement Python (provided in the form of a character string) which is evaluated on the level immediately higher than block itself. To simplify
the writing of the conditions, two dedicated functions are available: au_moins_un (KEYWORD

- , VALUES): who is checked if the intersection between the values of KEYWORD and VALUES is non empty. no (KEYWORD,
- VALUES): who is checked if the intersection is empty. One of the interests

to use these functions is that KEYWORD and VALUES can be simple values or lists of values. There is thus no test to make on the type of KEYWORD. Caution: -

the keywords

handled under the condition of the 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 catalog, with the principal root of the command, under keywords factors or in other blocks. In the example above, single-ended spanner keys RENUM and TAILLE_BLOC are on the same level, lower than that of METHODE, b_mult will _front and b_ldlt, him even lower than that of the key word factor solver. The conditions tested in the two blocks carry thus only on key word simple METHODE of level immediately higher than the blocks themselves, - it is however
possible to be freed from this rule while informing for a simple key word with the root of the command, the total attribute position=' ". It will be then visible under all the conditions of blocks. - it is necessary to pay
attention to the possible conflicts lorsqu" the same key word 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 key word simple RENUM: there cannot be conflict since 2 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. To typify the product concept

9 and to enrich it To typify the product concept

9.1 the argument sd

_prod allows D" to carry out the declaration of the type of product concept. If the command always produces same data structure some is the context, sd_prod is followed by the name of corresponding concept, already declared in the catalog of declaration of the concepts: accas.capy and SD /co_xxx.py. All the concepts being able to be produced by commands and/or used in keywords are declared in this file which is managed as a catalog of command. Example: In the catalog

of the command

: CALC_G = OPER (nom="

```
CALC_G", op=100, sd_prod=table_sdaster,… The table_sdaster type
```

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 key word or according to the type or value of a key word. In this case, sd_prod contains the function Python . The function receives in arguments the key words of the operator or procedure and must turn over the type of the product concept. Catalog makes dizzy: def

operateur_prod (MCLE1

```
, MCLE2, MCLE3, ** arguments): yew MCLE1 == "VALEUR1": return
 type1 yew (MCLE2!        =       Nun): return
type2 yew (AsType (          MCLE       3) == type3
): return type4……   raise       AsException
     ("standard of
result concept not envisaged") Body of the command: NOM
```

_COMMANDE=OPER (nom=" NOM

```
_COMMANDE", op=54, sd_prod= operateur_
```

prod,… Example: def mode_iter_inv

_prod (MATR_A

```
, MATR_C, TYPE_RESU, ** arguments): yew TYPE_RESU == "MODE_FLAMB
   ": return mode_flamb yew AsType (MATR_C) == matr_
   asse_depl_r: return mode_meca_c yew AsType (MATR_A) == matr_
   asse_depl_r: return mode_meca yew AsType (MATR_A) == matr_
   asse_pres_r: return mode_acou yew AsType (MATR_A) == matr_
   asse_gene_r: return mode_gene raise AsException ("standard of
   result concept not envisaged") MODE_ITER_INV=OPER (nom= " MODE

_ITER_INV", op=44, sd_prod=mode_iter_inv_prod
                   ,… In this case, if key word
```

TYPE_RESU has as an argument text " MODE_FLAMB" then the product concept is of the mode_flamb type. If not, if the type of the concept present behind key word simple MATR_C is matr_asse_depl _r, then the product concept is of the mode_meca_c type, etc Enhance the product concept

9.2 the argument reentrant makes it possible

to specify if the product concept 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=' N "the pending order

produces	necessarily a new concept. Example: LIRE_MAILLAGE=OPER (sd_prod =maillage, reentrant=", reentrant=' O "the command
modifies	an already existing concept systematically. Example: CALC_META=OPER (sd_prod=evol _ther, reentrant=" o', In this case, the data structure evol_ther must obligatorily be created as a preliminary by the operator of thermal to be enriched here by the command by metallurgical postprocessing. reentrant=' F "the two
situations are possible	. C" is the calculating case of the commands of the evolutions (data structures evol_ ***). One can want to connect the second transient computation behind a first and to supplement data structure of new times of computation obtained. Example: In the catalog : STAT_NON _LINE=OPER (sd_prod =evol_noli, reentrant=' f', In the command file : U=STAT_NON_LINE (…) U =STAT_NON_LINE (reuse=U, …) This possibility that one

offers to the user must be maturely considered and must remain departures from the general rule which wants that one does not modify a concept provided in entry. Indeed, when a concept is modified, the concepts which had been created by using it (before the change) are likely to lose coherence that 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 the factorization in core of the MATR_ASSE (operator TO FACTORIZE), this exception is justified by problems of obstruction of data bases. Routine of Name use

10 of the routine OPxxxx

10.1 is the routine which

carries out the associated command. The number of the routine opxxxx.f is selected among the free numbers. xxxx is a number coded on four digits. The two stages the supervisor

10.2 proceeds in 2

stages: one 1st stage: construction

- of the shaft of the python objects: command, command set, keywords, syntactic checking python, checking of coherence with the catalog, one 2nd stage: call to
- L " OPxxxx requires D" execution of computations the call to the operators, since

the supervisor, is done automatically according to the attribute op=xxxx well informed in the catalog , by: OPxxxx CAL (1ST) Recovery

of the arguments

10.3 of the command 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 key words in a routine called by the (possibly in the itself), then to carry out computations 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 result

- **: Subroutine GETRES makes it possible**

to obtain the name user of the product concept, the type of the concept associated with result and the name of the operator or of the command. These routines are described in

[D6.03.01] - Communication with the Supervisor of execution: routines GETXXX