Methods Python of access to the objects *Aster*

Summary:

This document presents the methods Python giving access the contained information in the structures of data Aster. This information can be treated by programming python, or be useful for the conditional sequence of the following orders.

Contents

1 Introduction and precautions for use

In *Code_Aster*, most orders are programmed in FORTRAN. The structures of produced data are accessible only via the manager from memory JEVEUX, even written to him in FORTRAN. In a standard execution of the code, only the names of the concepts (and not of the objects carrying themselves calculated information) are transmitted to the level of the supervisor, of order with order by the keywords.

In a more advanced use of Python than the simple declaration of orders *Code_Aster*, the command file written in Python can use the contents of the structures of data suitable for *Code_Aster*. Indeed, Python can be used in the command files to create macro - orders and operations like loops (for, while,...), tests (yew,...), external executions of orders (via the module bone), etc... The page "Use/Examples/Examples of use of Python in Aster" of the Web site www.code - aster.org gather a certain number of cases of application. It is then interesting for the user to recover the product of calculations FORTRAN in space python, i.e. its command file. Several methods Python were developed in order to reach the contents of other structures of data.

To recover data calculated (in the memory JEVEUX), it is absolutely necessary that the instructions involving their obtaining were indeed carried out as a preliminary. In other words, it is essential to carry out the code in mode PAR_LOT=' NON' (keyword of the order BEGINNING or CONTINUATION). Indeed, in this case, there is no total analysis of the command file, but each instruction is carried out sequentially. When one arrives on an instruction, all the concepts preceding it already were thus calculated.

```
BEGINNING (PAR_LOT = 'NOT')
```

It should then be noted that the command file thus produced is not readable by EFICAS which tolerates only the files exclusively made up of orders suitable for Code_Aster. Only simple variables (realities, entireties, G-strings) defined in declaratory mode $a='toto'$ or algebraic $n=3+4$ are readable by EFICAS.

The information read again in the memory JEVEUX, produced of a preliminary calculation, can be exploited for example for (nonexhaustive list):

- To connect conditionally other orders (execution of loop while until obtaining a computed value of ultimate stress)
- To handle in python of the contents of a table, a function, at ends of calculations
- To recover the attributes of a grid: list of the groups of nodes and meshs, coordinates.

2 Tables

Structures of data table are produced in Aster by creation (CREA_TABLE), by reading since a file (LIRE_TABLE) or recovery in another concept (RECU_TABLE). They are functionally heterogeneous tables of figures (whole, real, character strings) whose columns are identified by names of label.

These are practical structures whose employment is generalized in the code. For example, most orders of postprocessing tables produce: to raise into cubes of the constraints loci given, to produce calculated macroscopic sizes (postprocessings of breaking process).

That is to say for example the table tab1 following exit of a calculation Aster:

NODE	NUME_ORDRE	DX
N2	14	0.93
N2	15	1.16
N1	3	0.70
N1	2	0.46
N1	1	0.23

Table 2-1

It could also have been directly created like concept Aster of the type counts by:

```
tab1=CRÉA_TABLE (LIST = (
          _F (      PARA=' NOEUD',          LISTE_K= ('N2', 'N2', 'N1', 'N1',
'N1'),),
          _F (      PARA=' NUMÉRIQUE_ORDRE',   LISTE_I= (14,15,3,2,1),),
          _F (      PARA=' DX',          LISTE_R= (0 .93,1.16, 0.70, 0.46,
0.23),),)
```

One can directly recover an unspecified value of the table which one knows the access key (name of label of column) and the number of line:

```
>>> print tab1 ['DX', 3]
0.70
```

It is also possible to recover the totality of the table in the environment python via a dedicated class, produced by the method EXTR_TABLE, attached to the class of the concept ASTER:

```
tab2 = tab1.EXTR_TABLE ()
```

tab2 is a python object, authority of the class Table module Utilitai.Table. It is easy to handle with the methods associated with this class; one will be able to make help (Table) to know the methods of this class.

The table tab2 could also have been directly defined by a dictionary:

```
From Utilitai.Table importation Counts
listdic = [   {'NODE': 'N2', 'NUMÉRIQUE_ORDRE': 14, 'DX': 0.93,},
              {'NODE': 'N2', 'NUMÉRIQUE_ORDRE': 15, 'DX': 1. 16,},
              {'NODE': 'N1', 'NUMÉRIQUE_ORDRE': 3, 'DX': 0.70,},
              {'NODE': 'N1', 'NUMÉRIQUE_ORDRE': 2, 'DX': 0.46,},
              {'NODE': 'N1', 'NUMÉRIQUE_ORDRE': 1, 'DX': 0.23,},  ]
listpara= ['NODE', 'NUMÉRIQUE_ORDRE', 'DX']
listtype= ['K8', 'I', 'R']
tab2=Table (listdic, will listpara, listtype)
```

Possible operations on tab2 are described hereafter.

2.1 Impression

```
>>> tab2
-------------------------------------------
NODE       NUME_ORDRE    DX
N2         14            9.30000E-01
N2         15            1.16000E+00
N1         3             7.00000E-01
N1         2             4.60000E-01
N1         1             2.30000E-01
```

Also possible:

```
>>> print tab2
```

Posting of only one parameter:

```
>>> t.DX
-------------------------------------------
DX
9.30000E-01
1.16000E+00
7.00000E-01
4.60000E-01
2.30000E-01
```

The order IMPR_TABLE exploit the features of impression offered by this class. The interested reader will be able to read the programming python of this macro-order. In particular the possibility of printing cross tables.

2.2 Creation or impression of a under-table extracted by filter

Extraction according to only one criterion:

```
>>> print tab2.NUMÉRIQUE_ORDRE <=5
-------------------------------------------
NODE       NUME_ORDRE    DX
N1         3             7.00000E-01
N1         2             4.60000E-01
N1         1             2.30000E-01
```

Extraction according to two criteria with logical association “&”/AND:

```
>>> print (t.NUMÉRIQUE_ORDRE < 10) & (t.DX>=0.3)
-------------------------------------------
NODE       NUME_ORDRE    DX
N1         3             7.00000E-01
N1         2             4.60000E-01
```

Extraction according to two criteria with logical association “ | ”/OR:

```
>>> print (t.NUMÉRIQUE_ORDRE < 2) | (t.DX<0.5)
-------------------------------------------
NODE       NUME_ORDRE    DX
N1         1             2.30000E-01
N1         2             4.60000E-01
```

Extraction of a restricted number of labels:

```
>>> T ['DX', 'NUMÉRIQUE_ORDRE']
-----------------------------------------
DX            NUME_ORDRE
9.30000E-01   14
1.16000E+00   15
7.00000E-01   3
4.60000E-01   2
2.30000E-01   1
```

Extraction according to a criterion of equality (here with value of the criterion deduced itself from the table)

```
>>> t.DX == max (t.DX)
-------------------------------------------
NODE       NUME_ORDRE    DX
N2         15            1.16000E+00
```

2.3 Sorting

Sorting of the whole table according to a label:

```
>>> t.sort ('NUMÉRIQUE_ORDRE')
>>> T

-------------------------------------------
NODE       NUME_ORDRE    DX
N1         1             2.30000E-01
N1         2             4.60000E-01
N1         3             7.00000E-01
N2         14            9.30000E-01
N2         15            1.16000E+00
```

For sorting according to several labels, the order of precedence being that in which the labels are declared, it is necessary to provide the labels in the form of list or of tuple:

```
>>> t.sort (['NUMÉRIQUE_ORDRE', 'DX'])
```

A second argument `order`, being worth `'GROWING'` or `'DECREASING'`, allows to specify the order of sorting:

```
>>> t.sort (['NUMÉRIQUE_ORDRE', 'DX'], 'DECREASING')
```

2.4 Access to the values

The contents of the table are accessible by the method `been worth ()` who produces a dictionary whose keys are the parameters of access of the table and the values the columns:

```
>>> tab2.values ()
{'NODE': ['N1', 'N1', 'N1', 'N2', 'N2'], 'NUMÉRIQUE_ORDRE': [1, 2,3,14,15],
'DX': [0.23, 0.46,0.70,0.93,1.156]}
```

The parameters are given by the attribute `para` (idem `tab2.values () .keys ()`)

```
>>> will tab2.para
['NODE', 'NUMÉRIQUE_ORDRE', 'DX']
```

3 Access methods on the concepts

3.1 Access to the contents of a SD lists

```
lst = [listr8]. Values ()
```

`lst` is a list python which contains the values of the list Aster: `lst = [0. , 1.1,2.3,…]`

3.2 Access to the contents of a SD function or a SD tablecloth

```
lst1, lst2, (lst3) = [function/tablecloth]. Values ()
```

`lst1` and `lst2` are two lists python which contain the X-coordinates and the ordinates. If the function is complex, one obtains a third list and `lst2` and `lst3` the lists of the real and imaginary parts will contain.

```
lst1 = [function]. Absc ()
```

`lst1` is the list of the X-coordinates, that is to say also the first list returned by `Values ()`.

```
lst2 = [function]. Ordo ()
```

`lst2` is the list of the ordinates, that is to say also the second list returned by `Values ()`.

```
dico1 = [function]. Parameters ()
```

turn over a dictionary containing the parameters of the function; the jeveux type (`FUNCTION`, `FONC_C`, `TABLECLOTH`) is not turned over, the dictionary can thus be provided to `CALC_FONC_INTERP` such as it is (see efica02a).

3.3 Evaluation of a SD function or formula

The functions and the formulas are appraisable simply within the space of name python, therefore the command file, as follows:

```
FONC1=FORMULE (  VALE=' (Y ** 2) + X',
                 NOM_PARA= ('X', 'Y',)),
                 );
>>> print FONC1 (1. , 2.)
5.
```

or with a function:

```
FONC2=DEFI_FONCTION (  NOM_PARA=' X',  VALE= (0. , 0. , 1. , 4. ,))
>>> print FONC2 (0.5)
2.
```

In the case of functions, it should be noted that a tolerance of 1.e-6 into relative is applied when the value of the parameter is very near to the terminals in order to avoid an error due to prolongation prohibited except for the round-off.

3.4 Access to the contents of a SD grid

Two methods make it possible to recover the list of the groups of meshs and nodes of a structure of data of type grid:

```
[(tuple),…]      = [grid] .LISTE_GROUP_MA ()
```

return a list of tuples, each one containing the name of each group of meshs, the number of meshs which it contains and the dimension (0, 1.2 or 3) highest of its meshs:
`tuple = ('GMA', Nb meshs, dim. meshs)`

`[(tuple),…] = [grid] .LISTE_GROUP_NO ()`

return the list of the groups of nodes in the form:
`tuple = (name of the group_no, Nb of nodes of the group_no)`

3.5 Access to the contents of oneE SD `matr_acee_*`

That is to say `matr` a structure of data `matr_asse_depl_r`.

One recovers a table numpy full matrix while making:
```
array = matr.EXTR_MATR ()
```

To recover the matrix with a hollow storage, one makes:

```
D ATA, lines, collars, dim = matr.EXTR_MATR (sparse=True)
```

One a:

`array` is of dimension `(dim, dim)`
many nonworthless terms: `len (dated) = len (lines) = len (collars)`
`dated [K] = array [lines [K], collars [K]]`

In the same way for `matrgene` a structure of data `matr_asse_gene_r` , for example, product E by the operator `PROJ_MATR_BASE`.

One recovers a table numpy full matrix while making:
```
array = matrembarrassment.EXTR_MATR_GENE()
```

3.6 Access to the keys of a SD `result`

If `EVOL` is a structure of data `result`, then:

`dictionary EVOL.LISTE_CHAMPS ()`	=	is a dictionary whose keys are the names of the fields which index the list of the calculated sequence numbers.

Table 3.6-1

Example:

```
>>> print dictionary ['DEPL']          (the field DEPL is calculated with the
[0,1,2]                                sequence numbers 0.1 and 2)
>>> print dictionary ['SIEF_ELNO']     (the field is not calculated)
[]
```

Table 3.6-2

`dictionary EVOL.LISTE_VARI_ACCES ()`	=	is a dictionary whose keys are the variables of access which indexes their own values.

Table 3.6-3

Example:

```
>>> print dictionary ['NUMÉRIQUE_ORDRE']   (sequence numbers of the result EVOL
```

`[0,1,2]`	are: 0.1 and 2)
`>>> print dictionary ['INST']` `[0. , 2. , 4.]`	(calculated moments of the result `EVOL` are: 0.s, 2.s and 4.s)

Table 3.6-4

`dictionary = EVOL.LISTE_PARA ()`	is a dictionary whose keys are the parameters of calculation which index the lists (of cardinal equal to the number of calculated sequence numbers) their values.

Table 3.6-5

Example:

`>>> print dictionary ['MODEL']` `['MO', 'MO', 'MO']`	(name of the concept models reference for each sequence number)
`>>> print dictionary ['ITER_GLOB']` `[4,2,3]`	(iteration count of convergence for each sequence number)

Tablewith 3.6-6

3.7 Access with the contents of a SD `tran_gene`

If `trangene` is a structure of data `tran_gene`, for example, produced by the operator `DYNA_VIBRA`, then:

`trangene.FORCE_AXIALE(inoli=-1)`	Matrix 1D of the type " *numpy array* " containing the evolution of the thrust load at the filed moments with `inoli` being the index of non-linearity, by default, `inoli` = -1 (the first non-linearity).
`trangene.FORCE_NORMALE(inoli=-1)`	Matrix 1D of the type " *numpy array* " containing the evolution of the force normal at the filed moments with `inoli` being the index of non-linearity, by default, `inoli` = -1 (the first non-linearity).
`trangene.FORCE_RELATION(inoli=-1)`	Matrix 1D of the type " *numpy array* " containing the evolution of the force non-linear relation in displacement or speed at the filed moments with `inoli` being the index of non-linearity, by default, `inoli` = -1 (the first non-linearity).
`trangene.INFO_NONL()`	ListE information of non-linearities, including, the index and the heading of non-linearities.
`trangene.LIST_ARCH()`	ListE Instants filed.
`trangene.VARI_INTERNE(inoli=-1)`	Standard matrix 2D " *numpy array* " all internal variables of the non-linearity given by the index <`inoli`

Table 3.7-1

3.8 Access with the contents of a SD `melasflu`

This method python makes it possible to extract the list speeds of the fluid for which the calculation of the parameters of coupling fluid-rubber band was carried out.

```
>>> base = CALC_FLUI_STRU (...)

>>> print base.VITE_FLUI ()
[1. , 1.5,2.5,3.]
```

List fluid speeds for which the coefficients of coupling were calculated.

3.9 Recovery in python of the fields by elements and fields with the nodes (EXTR_COMP)

Method EXTR_COMP, applied to a field, recovery in python of the contents of the field allows.

3.9.1 Arguments of the order EXTR_COMP

The order has 3 arguments:

ch1 = CHAMP.EXTR_COMP (comp=' ', lgma= [], topo=0), for the fields with the nodes,

ch1 = CHAMP.EXTR_COMP (comp, lgma, topo=0), for the fields by element,

comp	component of the field on the list lgma. . For the fields with the nodes, if comp is left by default, all the components are turned over. The result of the order is modified (see below).
lgma	list of groups of meshs, if vacuum then one takes all them group_ma (equivalent with TOUT=' OUI' in the orders Aster.
report	one reference of information on topology if >0 (optional, defect = 0).

Table 3.9.1-1

Note: for the fields with the nodes, one can launch the order in the following way: ch1 = CHAMP.EXTR_COMP (topo=1). In this case, one turns over all the components for all the topological entities of the fields FIELD.

3.9.2 Example of utilisatioN

With to leave the result U :

1) One creates a field (node or elXX) correspondent at one moment by CREA_CHAMP.
2) One extracts the component by the method EXTR_COMP (declared for cham_elem and them cham_no) who creates a new type of python object: post_comp_cham_el and post_comp_cham_no whose attributes are described hereafter. One can extract all the components in only once by not specifying this one (for the fields with the nodes only).

```
U = STAT_NON_LINE (…)

U104 = CREA_CHAMP (
       TYPE_CHAM   = 'NOEU_DEPL_R',
       OPERATION   = 'EXTR',
       RESULT    = U,
       NOM_CHAM    = 'DEPL',
       NUME_ORDRE  = 104,
       )

U104NP = U104.EXTR_COMP ('DX', ['S_SUP',])

print U104NP.valeurs
```

```
V104 = CREA_CHAMP (
       TYPE_CHAM   = 'ELGA_VARI_R',
       OPERATION   = 'EXTR',
       RESULT    = U,
       NOM_CHAM    = 'VARI_ELGA',
       NUME_ORDRE  = 104,
       )

V104NP = V104.EXTR_COMP ('V22', [], 1)

print V104NP.valeurs
print V104NP.maille
print V104NP.point
print V104NP.sous_point
```

3.9.3 Results of the order EXTR_COMP

`ch1.valeurs: Numeric.array` containing the values

- For the fields by elements, if there is request topology (`topo>0`) :
  - `ch1.maille` : number of meshs
  - `ch1.point` : number of the point in the mesh
  - `ch1.sous_point` : number of under point in the mesh
- For the fields with the nodes, if there is request topology (`topo>0`) :
  - `ch1.noeud` : number of the nodes
  - `ch1.comp` : if one asked for all the components of the fields (`comp = ''`, value by default), component associated with the value.

4 Access method to an unspecified structure of data

It is possible to recover any vector or any collection present in the memory, with the help of the knowledge of the structure of data.

Two methods are possible (and equivalent): by using the catalogue of the structure of data or directly name JEVEUX of the object.

In the first case, one uses the “property” `sdj` concept which makes it possible to surf in the structure of data (see example hereafter).

4.1 Access to a structure of data of type vector

Method `getvectjev` the access to a structure of data of type vector allows. It always applies to the object “aster”, and takes in argument the character string supplements (space including) defining the name of the object contained in the structure of data which one wants to reach. This one can be given thanks to the order Aster `IMPR_CO (CONCEPT=_F (NOM=nom))`.

Example: to recover the coordinates of the nodes of a named grid MY:

```
LMBO = aster.getvectjev ("MY      .COORDO    .VALE        ")
```

Equivalent syntax by using the catalogue of structure of data is:

```
LMBO = MA.sdj.COORDO.VALE.get ()
```

One obtains a list python containing the values of the vector.

4.2 Access to a structure of data of type collection

In a similar way, method `getcolljev` the consultation of the collections allows since python. It returns a dictionary whose keys are the names of the objects in the event of named collection, the numbers of index if not.

Example: to recover information concerning the connectivity of the elements of the grid MY:

```
LMBO = aster.getcolljev ("MY       .CONNEX          ")
```

Equivalent syntax by using the catalogue of structure of data is:

```
LMBO = MA.sdj.CONNEX.get ()
```

One obtains a dictionary resembling:

```
{3: (2, 1.5), 2: (6, 9,10,7,11,12,13,8), 1: (1, 6,7,2,3,8,5)}
```