To introduce a new data structure

Summarized:

This document describes the method to introduce a new data structure into *Code_Aster* , in particular the drafting with the format “python” of the catalog of data structure.

Contents

1 Introduction

There are two large type of (SD) data structures in *Code_Aster* :

- SD local with the commands. These SD do not leave the operators and the user does not have access there. They are only present in space FORTRAN.

- SD total. These SD are used to communicate information between the operators and the executions of *Code_Aster* . They must comply with rules much more strict, to be accompanied by tools for checking of their integrity (mode SDVERI) and documented (D4 section of the documentation of development).

2 The SD local

the SD local are under the only responsibility of the developer, which is Master of its choices. However, a certain number of recommendations are given in D2.05.01 documentation (“Rules concerning the Structuring of the Data”). The SD local are necessarily created on volatile basis JEVEUX , they do not leave a command (the volatile base is cleaned automatically by the supervisor of execution at the end of each command).

3 The SD total

All the total data structures exchanged in *Code_Aster* exist in space Python. There exist three types of SD total:

- SD total strictly Python: they exist only in space Python and do not have their during in space FORTRAN.

- SD total strictly FORTRAN: they exist in space Python but are simply declared as being types. They have neither method, nor specific attribute (i.e. others that those of the class hat ASSD).

- Mixed SD total. They exist in space Python, have specific methods and/or attributes. Moreover, they exist in space FORTRAN.

The statement of the SD total is made in the files $ASTER_ROOT/catapy/entete/co_ **** .capy.
These SD total which are transmitted of a command to another are baptized “concepts”: they are Python the known objects of the command set. The concepts derive all, directly or indirectly, of the class hat ASSD. The attribute of class cata_sdj (as “catalogues Jeveux data structure”) specifies the class Python which defines data structure FORTRAN.
For example:
cata_sdj = "SD.sd_fonction.sd_fonction_aster" which indicates
that data structure FORTRAN of the functions is defined by the class sd_fonction _aster modulus sd_fonction.py in the library SD. Thereafter

, one is interested in the JEVEUX objects of the SD and their checking. The mechanism

of checking of the SD total is activated in most benchmark, except when this checking is too expensive or fails. Activation is done of two ways: All in all

- , on all the command file, via keyword SDVERI=' YES "in the command debut Locally

- , between each command, via keyword SDVERI=" YES "in command DEBUG the checking

is spring of the developer of data structure total. As a class Python is used, the checks can be as thorough as necessary (name and type of the JEVEUX objects, coherence of dimensions, SD specialized according to the operators, etc). SD purely

3.1 in Python There exists very

little about it. It is recommended to declare this kind of classes, in the command file (or in a modulus Python imported in the command file). There exist currently four classes of this type. Here their declaration in accas.capy : class No (

```
GEOM): not  class grno (
GEOM): not class my (
GEOM): not  class grma (
GEOM): not These classes
```

derive all from the class mother GEOM described in file N_GEOM of the library bibpyt/Core . They are not used qu" to define a specific type for the geometrical entities for the management of keywords GROUP_MA/GROUP _NO in the syntactic decoder of the catalog of the commands. SD with space

3.2 FORTRAN All these

classes are declared in the heading of the catalog (catapy/entete/co_ **** .capy). Examples: class cabl_

```
precont (ASSD): cata_sdj =
    "SD.sd_cabl_precont.sd_cabl_precont" cabl_precont is
```

a class purely FORTRAN, no specific method is not defined. cham_elem is a class containing , for example, a method EXTR_COMP which is clean for him . The classes can

also inherit other classes. For example, the class evol_noli inherits the class evol_sdaster, which inherits itself the class resultat_sdaster. Like known as previously

, the attribute cata_sdj declares the class used to describe and check the JEVEUX objects of SD FORTRAN associated with the concept (mode SDVERI). These classes all are gathered in the directory (library python) bibpyt/SD, under the name sd_ **** .py. Classes of description

4 of the SD the class Python

sd_* thus contains the description of the SD in terms of objects FORTRAN, it derives from the AsBase object. The AsBase class contains : The name of the SD

- nomj an attribute
- to classify the SD as being " optional " or not: optional a method
- to allot the nomj: setname a method
- to check the SD: check Various methods
- for the printing (overload of repr) By heritage , one

defines the basic JEVEUX object: class OJB (AsBase) . This class contains: The description of

- attributes JEVEUX (see D6.02 .01 Management memory: JEVEUX) in protected attributes an attribute protected

- on the existence or not from the JEVEUX object: __exists As it is seen,

the attributes of class OJB are protected , one reaches it via classes derived from OJB which describe the existing objects. There are three basic classes: OJBVect: a simple

- object with meaning JEVEUX OJBPtnom: a pointer
- object of names to meaning JEVEUX OJBCollec : a collection
- object with meaning JEVEUX There exists "

alias" of these classes, for reasons of compatibility and legibility: AsObject is another

- name of class OJB AsPn is another
- name of the class OJBPtnom AsVect is another
- name of the class OJBVect AsColl is another
- name of the class OJBCollec the OJBVect class

is derived in objects even more elementary, which makes it possible to approach the syntax of routine FORTRAN WKVECT: AsVI: the object

- is a vector of integers INTEGER AsVR: L "object
- is a vector of realities REAL*8 AsVC: L" object
- is a vector of complexes COMPLEX*16 AsVL: the object
- is a vector of Boolean LOGICAL AsVK8: the object
- is a vector of characters CHARACTER*8 AsVK16 : the object
- is a vector of characters CHARACTER*16 AsVK 24: the object
- is a vector of characters CHARACTER*24 AsVK 32: the object
- is a vector of characters CHARACTER*32 AsVK 80: the object
- is a vector of characters CHARACTER*80 the manufacturer of

these objects can contain all usual attributes JEVEUX. For example , for a collection: TAVA = AsColl (SDNom

```
(debut=19), acces=' NU', stockage=' CONTIG', modelong=' CONSTANT
              ", type=' K', ltyp=8,) Use of the classes
```

5 Python for the checking Naming Initially

5.1 ,

it is appropriate to name the JEVEUX object which is used as a basis for the SD. One recalls qu "to each JEVEUX object is allotted a name, which is a character string length 24 (CHARACTER*24). It is also pointed out that L" user names his concepts in the command file via a character string length 8 (CHARACTER*8). One of the guiding principles for the construction of the SD total in Code_Aster is *always* to prefix **the name** of data JEVEUX relating to the concept (produced by the command) by the name of this last. For example, the mesh contains a vector of realities with cordonnées of the nodes, it is created as follows: COOVAL = MAIL (1:8

```
)/".COORDO .VALE" CAL WKVECT    (COOVAL
, "G V R", 3*NBNO, JCOOR) Here MAIL is
```

the name of the concept given by the user (MAIL = LIRE_MAILLAGE (...)). When one is in the class Python which will be instanciée for the concept that one must check, the first thing to be made is to determine the name of all the objects which will belong to the SD: class sd_maillage

```
(sd_titre): nomj = SDNom (fin=
    8) Thus nomj will be
```

the prefix of all the JEVEUX objects contained in the SD. One builds it by taking the first 8 characters of the SD (fin=8). Then , it is a question of checking the presence of objects in the SD. For example, the sd_maillage contains obligatorily an object DIME, which is a vector of integers length 6: class sd_maillage

```
(sd_titre): nomj = SDNom (fin=
    8) louse = AsVI (SDNom
    (nomj='.DIME'), lonmax=6,) the louse object is
```

a vector of integers (AsVI) whose attribute LONMAX is worth 6. One could also have written in a more compact way: class sd_maillage

```
(sd_titre): nomj = SDNom (fin=
    8) DIME = AsVI (lonmax
    =6,) This last construction
```

(DIME = AsVI (lonmax=6,)) is a facility offered to the developer, based on the fact that the name of a JEVEUX object is always built same way. In an implicit way, when one writes DIME = AsVI (), one builds instance name DIME of the AsVI class whose JEVEUX object has as a name nomj (1:8)/".DIME ". This instanciation must be privileged, in order to facilitate the reading of the catalog. It happens sometimes that the attributes D" an object (as its type) can be variable according to L" operator creating this SD. In this case, one can use the function member Among (). For example: VALE = AsVect (ltyp

```
=Parmi (4,8,16,24), type=Parmi ("It, "I", "K', "R"), docu=Parmi (", "2
                ", "3"),) the .VALE of
```

a field at nodes can contain complex, whole, real values or even of the character strings length 8,16 or 24. Optional objects

5.2 One can declare

that an object is optional in the SD, for example, it can not there not have a GROUP_NO in the SD mesh: class sd_maillage

```
(sd_titre): nomj = SDNom (fin=
  8) GROUPENO = Optional
  (AsColl (acces=' NO', stockage=' DISPERSE', modelong=' VARIABLE
                                  ", type=' I",)) The function Optional
```

positions the attribute optionnal in True . By defaults , all the objects are compulsory. How the mechanism of checking of SD (SDVERI) traverses all the JEVEUX objects attached to the concept , it is imperative that all the objects (compulsory or optional) were declared in the class of the SD! Method of existence

5.3 "exists" It is sometimes useful

to know if a SD exists. For that , one can overload the method exists which turns over Boolean : def exists (coil):

```
# turns over "true"
   if the SD seems to exist (and thus that it can be # checked) return
   self.REFE.exists to check
   the existence,
```

simplest is to control the presence of a compulsory object (here REFE). It is important to note that in

the AsBase class (from which all the others derive), exists is an attribute (taking a logical value **True** or False). It is built by calling on low the level with routine FORTRAN JEEXIN (OBJET, IRET). When exists is overloaded like
above , it becomes a method. Consequently it is imperative **to call it** like such, i.e. without forgetting the opening and closing brackets "()". For example the sd_ligrel redefines
the method exists, one must thus call it as follows : yew self.contact_resolu (): # not to forget

```
them () because sd_ligrel.exists
   is a method assert self.LIGRE.exists () Methods of checking "
   check_" All the classes
```

5.4 derived from AsBase contain

the method check. By default, this function is satisfied to check the conformity of the SD to the attributes of the JEVEUX object. For example: class sd_maillage (sd_titre): nomj = SDNom (fin=8) DIME =

```
AsVI (lonmax=6,) One is satisfied
  to check that
  the JEVEUX object nomj (1
```

: 8)/".DIME" is well a vector of integers length 8. It is nevertheless possible (and desirable!) to overload a method check to make more thorough checks. For example , always in the sd_maillage, one would like to check that the pointer of name MAIL (1:8)/".NOMNOE", which contains the name of the nodes is well length equal to the number of nodes: class sd_maillage (sd_titre): nomj = SDNom (fin=8) DIME

```
= AsVI (lonmax=6,) NOMNOE = AsPn
  (ltyp   =8) def check_
  NODES   (coil, checker): tithe
  = self.DIME.get () nb_no = tithe

  [0] assert self.NOMNOE.nomuti == nb_no One declared
```

```
well     that object
NOMNOE    was
a pointer of names (contained in
```

character strings length ltyp=8). Then a new method check_NOEUDS is declared, of which one of the arguments is obligatorily checker (this basic class for the checks contains in particular a mechanism to control the depth of the checks and to avoid controlling several times the same objects). All the functions members which start with check_ will be carried out at the time of the instanciation of the class SD which one checks . It should be noted that two obligations: The method must obligatorily start with check_ the method must have a checker

- object in argument the class checker contains a dictionary
- of all the already checked JEVEUX objects

, it is enough for that to use the data member names: yew checker.names.has_key (nomsd): return That results in: if the JEVEUX object of name nomsd

```
were already checked, then return  . Access
```

to the contents of the JEVEUX objects "get ()" With the preceding example, we introduced another

5.5 mechanism of control, it acts of line

the dime = self.DIME.get (). It is indeed possible to reach the contents of objects FORTRAN in order to recover information of them. For that , the supervisor uses the two methods of the modulus aster : getvectjev and getcolljev. It goes without saying it is completely possible to define attributes and methods specific to the SD which

one describes. For example, in the sd_maillage, there exists a function member u_dime who gives generic information: class sd_maillage (sd_titre): nomj = SDNom (fin=8) DIME = AsVI (lonmax =6,) def u_dime (coil): dime=self.DIME.get () nb_no =dime [0]

```
nb_nl =dime [1] nb_ma =dime [
  2] nb_    Sm =dime [3] Nb
  _sm_mx    =dime [4] dim_coor
  =dime [5] return Nb
      _no, nb_nl, nb_ma, Nb
      _sm, Nb     _sm_mx, dim
      _coor    Note:
      : if    the object
      is    a simple object
      , get () turns over
      a python list
      , if the object is a collection, get () turns over
```

a dictionary Python. Heritage and typing All the classes describing the SD can be used in other classes. For example: class

5.6 sd_maillage (sd_titre

): nomj = SDNom (fin=8) COORDO = sd_ cham _no () One sees in this example a double mechanism. The first is

```
the classical heritage:
   the sd_maillage drift
```

of the sd_titre of which

description is: class sd_titre (AsBase): TITR = AsVK80 (SDNom (debut=19), optional=True) the sd_titre contains only one vector of K80 stored in

the JEVEUX object whose
name starts with the 19th character. This object

is optional . The second mechanism uses the concept of typing of the data suitable for an object language like Python. Indeed, the object nomj

(1:8)/".COORDO" is a SD of the cham_no type: class sd_cham_no (sd_titre): nomj = SDNom (fin=19) VALE = AsVect (ltyp=Parmi (4,8,16,24), type=Parmi ("It , "I", " K', "R"), docu =Parmi (

'', "2", "3"),) REFE = AsVK
24 () DESC = AsVI (docu
= ' CHNO',) Attention with the circular references (the SD mesh
contains a cham_no object which
contains a mesh
object). It is with the developer

there to take care (see for example sd_cham_ No). Utilities A certain number of operations of checking are available in the modulus sd_util: sdu_assert (ojb, checker ,

5.7 bool, how

=): check that the Boolean one (bool) is true; sdu_compare (ojb, checker, val1, comp , val 2,

-): check how the relation of comparison between val1 and val2 is respected with comp
- = "=="/"! ="/">="/">"/ "<="/"<"; sdu_tous_differents (ojb, checker, sequence=None,): check that the elements all of the sequence are different; sdu_tous_non
- _blancs (ojb, checker, sequence=None,): check that the elements (character strings) of the sequence are all "not blanks"
- ; sdu_tous_compris (ojb, checker, sequence=None, vmin =None, vmax=None,): check that all the values of the sequence lie between
- vmin and vmax; sdu_monotone (seqini): check that a sequence is sorted by order ascending (or decreasing); sdu_nom_gd (numgd): turn over
- the name of the quantity of number (numgd); sdu_licmp_gd (numgd): turn over the list of the cmps of
- the quantity of number (numgd); sdu_nb_ec (numgd): turn over the number of integers
- coded to describe the components of the quantity (numgd); sdu_ together (lojb
-): check that the JEVEUX objects of lojb exist simultaneously.
-