To introduce a new structure of data

Summary:

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

Contents

1 Introduction

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

- SD local with the orders. These SD do not leave the operators and the user does not have access there. They are only present in space FORTRAN.
- SD total, also called “concepts” in documentation. 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 SD local

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”). SD local are necessarily created on the volatile basis JEVEUX , they do not leave an order (the volatile base is cleaned automatically by the supervisor of execution at the end of each order).

3 SD total

All structures of total data 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).
- SD total mixed. They exist in space Python, have specific methods and/or attributes. Moreover, they exist in space FORTRAN.

The declaration of SD total is made in the files code_aster/Catastrophes/Legacy/DS/co_** .py.
These SD total which is transmitted of an order to another “concepts are baptized”: 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 “structure of Jeveux data catalogues”) the class Python specifies which defines the structure of data FORTRAN.
For example:

```
cata_sdj = 'SD.sd_fonction.sd_fonction_aster'
```

who indicates that the structure of data FORTRAN of the functions is defined by the class sd_fonction_aster module sd_fonction.py in the library SD.

Thereafter, one is interested in the objects JEVEUX SD and with their checking.

The mechanism of checking of SD total is activated in most CAS-tests, except when this checking is too expensive or fails. Activation is done in two manners:

- All in all, on all the command file, via the keyword SDVERI=' YES ' in the order BEGINNING
- Locally, between each order, via the keyword SDVERI=' YES ' in the order DEBUG

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

3.1 SD purely in Python

There exists very little about it. It is recommended to declare this kind of classes, in the command file (or in a module Python imported in the command file). There exist currently four classes of this type. Here their declaration in DataStructure.py :

```
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 the file N_GEOM library bibpyt/Core. They are only used to define a specific type for the geometrical entities for the management of the keywords GROUP_MA/GROUP_NO in the syntactic decoder of the catalogue of the orders.

3.2 SD with space FORTRAN

All these classes are imported in the heading of the catalogue (code_aster/Catastrophes/DataStructure.py). 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 who is clean for him.

The classes can also inherit other classes. For example, the class evol_noli inherit the class evol_sdaster , which inherits itself the class resultat_sdaster.

Like known as previously, the attribute cata_sdj declare the class used to describe and check objects JEVEUX of SD FORTRAN associated with the concept (mode SDVERI). These classes all are gathered in the repertoire (library python) bibpyt/SD, under the name sd_ **** .py.

4 Classes of description of SD

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

- the name of SD nomj
- an attribute to classify SD as being "optional" or not: optional
- a method to allot it nomj : setname
- a method to check SD : check
- various methods for the impression (overload of *repr*)

By heritage, the object is defined JEVEUX basic: class OJB (AsBase). This class contains:

- the description of the attributes JEVEUX (see D6.02.01 Management memory: JEVEUX) in protected attributes
- an attribute protected on the existence or not from the object JEVEUX : __exists

As it is seen, attributes of the class OJB are protected, one reaches it via classes derived from OJB who describe the existing objects. There are three basic classes:

- OJBVect : a simple object with the direction JEVEUX
- OJBPtnom : a pointer object of names to the direction JEVEUX
- OJBCollec : a collection object with the direction JEVEUX

There exists “alias” of these classes, for reasons of compatibility and legibility:

- AsObject is another name of the 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 class OJBVect 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 entireties INTEGER*8
- AsVR : the object is a vector of realities REAL*8
- AsVC : the object is a vector of complexes COMPLEX*16
- AsVL : the object is a vector of Boolean LOGICAL*8
- AsVK8 : the object is a vector of characters CHARACTER*8
- AsVK16 : the object is a vector of characters CHARACTER*16
- AsVK24 : the object is a vector of characters CHARACTER*24
- AsVK32 : the object is a vector of characters CHARACTER*32
- AsVK80 : the object is a vector of characters CHARACTER*80

Note: types used in *Code_Aster* are defined in the file asterf.config.h

The manufacturer of these objects can contain all the attributes JEVEUX usual. For example, for a collection:

```
TAVA = AsColl (SDNom (debut=19), acces=' NU', stockage=' CONTIG',
               modelong=' CONSTANT', type=' K', ltyp=8,)
```

5 Use of the classes Python for the checking

5.1 Naming

Initially, it is advisable to name the object JEVEUX who is used as a basis for SD. One reminds that each object JEVEUX a name is allotted, which is a character string length 24 (CHARACTER*24). It is also pointed out that the user names his concepts in the command file via a character string length 8 (CHARACTER*8). One of the basic principles for the construction of SD total in *Code_Aster* is of **always** to prefix the name of the data JEVEUX relating to the concept (produced by the order) by the name of this last. For example, the grid contains a vector of realities with cordonnées of the nodes, it is created as follows:

```
COOVAL = E-MAIL (1:8)/'.COORDO    .VALE'
CAL WKVECT (COOVAL, 'G V R', 3*NBNO, JCOOR)
```

Here E-MAIL is the name of the concept given by the user (E-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 SD :

```
class sd_maillage (sd_titre):
    nomj = SDNom (fin=8)
```

Thus nomj will be the prefix of all the objects JEVEUX contents in SD. One builds it by taking the first 8 characters of SD (fin=8). Then, it is a question of checking the presence of objects in SD. For example, sd_maillage obligatorily an object contains TITHE, which is a vector of entireties length 6:

```
class sd_maillage (sd_titre):
    nomj = SDNom (fin=8)
    louse = AsVI (SDNom (nomj='.DIME'), lonmax=6,)
```

The object louse is a vector of entireties (AsVI) of which the attribute LONMAX is worth 6. One could also have written in a more compact way:

```
class sd_maillage (sd_titre):
    nomj = SDNom (fin=8)
    TITHE = AsVI (lonmax=6,)
```

This last construction (TITHE = AsVI (lonmax=6,)) is a facility offered to the developer, based on the fact that the name of an object JEVEUX same manner is always built. In an implicit way, when one writes TITHE = AsVI () , an authority of name is built TITHE class AsVI of which the object JEVEUX has as a name nomj (1:8)/'.DIME ' . This instanciation must be privileged, in order to facilitate the reading of the catalogue. It happens sometimes that the attributes of an object (as its type) can be variable according to the 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'),)
```

.VALE of a field to the nodes can contain complex values, whole, real or even of the character strings length 8.16 or 24.

5.2 Optional objects

One can declare that an object is optional in SD, for example, there can not be GROUP_NO in SD grid :

```
class sd_maillage (sd_titre):
```

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

The function `Optional` position the attribute `optionnal` with `True`. By defaults, all the objects are obligatory. Like the mechanism of checking of `SD` (`SDVERI`) traverses the whole of the objects `JEVEUX` attached to the concept, it is imperative that all the objects (obligatory or optional) were declared in the class of `SD` !

5.3 Method of existence " `exists` "

It is sometimes useful to know if one `SD` exist. For that, one can overload the method `exists` who 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 an obligatory object here (`REFE`).

It is important to note that in the class `AsBase` (from which all the others derive), `exists` is one **attribute** (taking a logical value `True` or `False`). It is built by calling on low the level with routine FORTRAN `JEEXIN (OBJECT, IRET)`.
When `exists` is overloaded like above, it becomes one **method**. Consequently it is imperative to call it like such, i.e. without forgetting the opening and closing brackets "()".
For example `sd_ligrel` the method redefines `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 ()
```

5.4 Methods of checking " `check_` "

All classes derived from `AsBase` contain the method `check`. By default, this function is satisfied to check the conformity of `SD` with the attributes of the object `JEVEUX`. For example:

```
class sd_maillage (sd_titre):
  nomj = SDNom (fin=8)
  TITHE = AsVI (lonmax=6,)
```

One is satisfied to check that the object `JEVEUX nomj (1:8)/'.DIME '` is well a vector of entireties length 8. It is nevertheless possible (and desirable!) to overload a method `check` to make more thorough checks. For example, always in `sd_grid`, one would like to check that the pointer of name `E-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)
  TITHE   = AsVI (lonmax=6,)
  NOMNOE = AsPn (ltyp=8)

  def check_NOEUDS (coil, checker):
      tithe     = self.DIME.get ()
      nb_no    = tithe [0]
      assert self.NOMNOE.nomuti == nb_no
```

It was well declared that the object NOMNOE was a pointer of names (contained in chains length ltyp=8). Then a new method is declared check_NOEUDS, 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 that one checks. It should be noted that two obligations:

- the method must obligatorily start with check_
- the method must have an object checker in argument

The class checker a dictionary of all the objects contains JEVEUX already checked, it is enough for that to use the data member names :

```
yew checker.names.has_key (nomsd):  return
```

That results in: if the object JEVEUX of name nomsd was already checked, then return.

5.5 Access to the contents of objects JEVEUX " get () "

With the preceding example, we introduced another control mechanism, it acts of the line tithe = self.DIME.get (). It is indeed possible to reach the contents of the objects FORTRAN in order to recover information of them. For that, the supervisor uses the two methods of the module aster : getvectjev and getcolljev.

It goes without saying it is completely possible to define attributes and methods specific to SD that one describes. For example, in sd_grid, there exists a function member u_dime giving generic information:

```
class sd_maillage (sd_titre):
  nomj   = SDNom (fin=8)
  TITHE   = 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 () turn over a list Python, if the object is a collection, get () turn over a dictionary Python.

5.6 Heritage and typing

All classes describing them SD can be used in other classes. For example:

```
class 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: sd_grid drift of sd_title description is:

```
class sd_titre (AsBase):
   TITR = AsVK80 (SDNom (debut=19), optional=True)
```

sd_title only one vector contains of K80 stored in the object JEVEUX 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 one SD of type cham_no :

```
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 = AsVK24 ()
   DESC = AsVI (docu=' CHNO',)
```

Attention with the circular references (SD grid an object contains cham_no who contains an object grid). It is with the developer there to take care (see for example sd_cham_no).

5.7 Utilities

A certain number of operations of checking are available in the module sd_util :

- sdu_assert (ojb, checker, bool, comment=) : check that the Boolean one (bool) is true;
- sdu_compare (ojb, checker, val1, comp, val2,) : 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 (chains) of the sequence are all “not white”;
- 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 size of number (numgd);
- sdu_licmp_gd (numgd) : turn over the list of the cmps of the size of number (numgd);
- sdu_nb_ec (numgd) : turn over the number of entireties coded to describe the components of the size (numgd);
- sdu_ensemble (lojb) : check that the objects JEVEUX of lojb exist simultaneously.