To introduce a new size (or a new component)

Summary:

This document describes what it is necessary to do to introduce a new size into *Code_Aster* or a new component in an existing size.

In a few words, to add a size, it is necessary:

- to modify the catalogue describing the sizes,
- to add the name of the size in a catalogue of orders.

To introduce a component into an existing size, it is enough:

- to enrich the catalogue by the sizes.

Contents

1 Introduction

For Code_Aster, a size is made up of an identifier (its name), of a type and a list of components.

Example of size :

- DEPL_R : name of the size of real displacements to the nodes. One associates with this size the real type R and components DX, DY, DZ, DRX, DRY MARTINI, DRZ,….

In *Code_Aster*, the simple sizes and the elementary sizes are distinguished. The elementary sizes are attached to the vectors or matrices elementary. They are built starting from the simple sizes. All these sizes (simple and elementary) are described in the file Commons/physical_quantities.py. The description of the elementary sizes is made with [the §4]

We will try to answer the questions:

- what is necessary to make to add a new size?
- what is necessary to make to add a new component in an existing size?
- which catalogues is necessary it to modify?

The introduction of a new size requires two actions:

- modification of the catalogue of the sizes: physical_quantities.py,
- modification of the catalogue of the orders: c_nom_grandeur.capy

We will detail these two actions successively.

2 Modification of the catalogue of the sizes "physical_quantities.py"

One has in this paragraph the structure of the catalogue physical_quantities.py, and one described how one proceeds to add a size or a component.

2.1 Description of the catalogue

The catalogue physical_quantities.py is present in the repertoire Commons repertoire catalo/cataelem. We present below an extract of this catalogue:

```
ABSC_R   = PhysicalQuantity (type=' R', components= ('ABSC [4]',),
        comment= """  ABSC_R  Type: R  Curvilinear X-coordinate along a
telegraphic grid
       ABSC1: curvilinear X-coordinate of the 1st node of a SEG or a POI1
       ABSC2: curvilinear X-coordinate of the 2nd node of a SEG
       ABSC3: curvilinear X-coordinate of the 3rd node of a SEG (if there
exists)
       ABSC4: curvilinear X-coordinate of the 4th node of a SEG (if there
exists)
""")
```

The block defining a size contains:

- An argument " type=' R' ". It represents type FORTRAN of the components of the size.
- An argument " components= (...) " to define the list of the names of the components of the size. The order of the components is important.
  The names of the components are chains having with more the 8 characters. When there exists a "series" of components whose names "are numbered", one can (and one must!) use a condensed syntax. For example: 'ABSC [4]'replaces ('ABSC1', 'ABSC2', 'ABSC3', 'ABSC4',).
- An argument " comment= " allowing to comment on the size and its components. It is important to document ALL the components. This comment perhaps printed in certain error messages to help the user. When these comments are written, it is initially to the user that it is necessary to think.

2.2 Addition of a size

The developer eager to add a size (sometimes invited "simple size" to distinguish it from an "elementary" size) will have to choose a name of size, to allot a type to him, and to choose the names components of this size.

2.2.1 Choice of the type

The type to be associated with the size is type FORTRAN of the values of its components. The developer will have to choose a type among the choices below:

- R : reality,
- I : entirety,
- C : complex,
- K8 : chain of 8 characters,
- K16 : chain of 16 characters,
- K24 : chain of 24 characters,

- …

2.2.2 Name of the size

The first thing to be made is to define a name of size which sufficiently explicit and is not used in this catalogue.

How to define a name of size?

The names are represented by a chain of with more the 8 characters. In the preceding example, the two definite sizes have as a name: ABSC_R and STAOUDYN.

Note:

It is advised to introduce the type of the size into its name (for example ABSC_R), because it is simpler for the developer to handle a size whose type is implicitly known for him. Usually, one makes it precede by the character "_" (underscore) in order to dissociate it from the reference symbol of the size.

2.2.3 Names of the components

The second thing to be made is to define the list of the components. One must put the following questions:

- do I need to name each component explicitly?
- does one know by advance the component count?

First case (simplest): one knows exactly the component count and one wishes to name each one of them with a significant name. For that, one defines for each component a name of in more the 8 characters. It is the case of the following example:

```
XCONTAC  = PhysicalQuantity (type=' R',
    components= ('RHON', 'DRIVEN', 'RHOTK', 'INTEG', 'COECH',…
```

Second case : one wishes to define a size of which the component count is consequent and of which the name of each one of them imports little. With this intention, there exists in Code_Aster of the "neutral" sizes. Sizes NEUT_X (where X represent the type of the size).

Example:

```
NEUT_R   = PhysicalQuantity (type=' R', components= ('X [30]',),
    comment= """  Standard NEUT_R: R Size 'neutral' of real type
       The significance of the components varies from an option has the
other. This
       size 'pass key' is not useful which the introduction has to avoid of
       many sizes without much interest.
       X (1)  : component 1
       …
       X (30): component 30
""")
```

This example describes a "neutral" size of type R being able to collect with more the 30 components.

When the component count exceeds 30, it is also possible to use the "neutral" sizes N120_R , N120_I and N480_I. Those can contain 120 components of the type respectively R , 120 components of the type I and 480 components of the type I .

Note:

One informs in physical_quantities.py the maximum number of components which a size can lay out. The component count of a size necessary to elementary calculation is defined in each catalogue of element.
The use of the sizes "NEUT_X" allows to avoid the multiplication of the sizes.

One **third case** can present itself: there exists in Code_Aster **one** "special" size being able to have an unspecified number of components. It is the size **VARI_R** :

```
VARI_R   = PhysicalQuantity (type=' R',
components= ('VARI',),
comment= """  Standard VARI_R: R… """)
```

This size has apparently only one component ('VARI'), but actually, this number is variable (from one element to another for example). When this size is printed, the components are then named: 'V1', 'V2', 'V3' ,...

Note:

Size VARI_R is often used to represent the internal variables of the laws of nonlinear behavior. But it can have other uses.
There exists also the size VAR2_R similar to VARI_R. The developer is invited to consult the catalogue physical_quantities.py for more information on this size .

2.3 Addition of a component in an existing size.

It is enough to enrich the list by the components to the size with a new name. You will not forget to comment on this new component.

Note: it is more advisable to add the new component at the end of the list.

3 Modification of the catalogue "c_nom_grandeur.capy"

A second phase not to be forgotten during the introduction of a new size (simple) into Code_Aster, is the modification of the catalogue c_nom_grandeur.capy.

This catalogue is present in the repertoire commun run repertoire catapy.

It counts all the simple sizes present in Code_Aster.

Extract:

```
def C_NOM_GRANDEUR (): return  (
"ABSC_R",
"ADRSJEVE",
"ADRSJEVN",
"CAARPO",
"CACABL",
"CACOQU",
"CADISA",
"CADISK",
…
)
```

For what is used it?

It is used by the supervisory python to check the seizure of the users in the command files. For example, during the creation of a field (operator CREA_CHAMP), it is necessary to provide to the keyword TYPE_CHAM a character string describing the type of field to be built (localization and size). If TYPE_CHAM = 'ELNO_SIEF_R' (real stress field to the nodes by element), Code_Aster exploits the catalogue c_nom_grandeur.capy in order to check if the character string 'SIEF_R' seized by the user corresponds to a size.

4 "Elementary" sizes

Until now, we spoke only about the "simple" sizes. A "simple" size is a list of named components.

Fields with the nodes (cham_no_xxx), cards (carte_xxx) and fields by elements (cham_elem_xxx) all are associated with a "simple" size. For example, a field with the nodes of displacement is associated with the size DEPL_R. On each node of the grid, this field can "carry" one (or several) component of the size DEPL_R.

"Elementary" sizes are those which are associated with the structures of data resuelem (elementary vectors or elementary matrices).

One resuelem is a "field" containing 1 elementary vector (or 1 elementary matrix) per mesh. The size associated with this field is an elementary size.

4.1 Syntax

The description of the elementary sizes in the file physical_quantities.py is made at the end of the catalogue (because one uses there the definite simple sizes at the beginning of the file).

Examples:

```
MDEP_C   = ArrayOfQuantities (elem=' MS', phys= DEPL_C)
MDNS_R   = ArrayOfQuantities (elem=' MR', phys= DEPL_R)
MPRE_C   = ArrayOfQuantities (elem=' MS', phys= PRES_C)
VDEP_C   = ArrayOfQuantities (elem=' V', phys= DEPL_C)
VDEP_R   = ArrayOfQuantities (elem=' V', phys= DEPL_R)
```

An elementary size of standard "vector" uses elem=' V'.
An elementary size of type "matrix" uses elem=' MS' (if it is symmetrical) or 'MR.' (if it is nonsymmetrical).

4.1.1 Size "elementary vector"

A size of type "elementary vector" (for example VDEP_R above) is simply described by elem=' V' and size simple partner with the elementary size (here DEPL_R).

4.1.2 Size "stamps elementary"

A size of the type "stamps elementary" (for example MDNS_R above) is described by elem=' MR' (nonsymmetrical matrix) and size simple partner with the elementary size (here DEPL_R).

4.2 Conventions of storage

The scalars (in general of realities) which make it possible to represent an elementary size are numerous: for example, the matrix of rigidity (symmetrical) of a machine element MECA_HEXA20 contains (60*61/2) real.

It is not question of naming all these realities (like the components of a simple size).

Conventions are necessary concerning the storage of these scalars.

For an elementary vector (intended to be assembled to give a second member), one seeks to store something which resembles a field “with the nodes” for each element. The convention of storage is natural. For example:

N1			N2			N3			…
DX	DY	DZ	DX	DY	DZ	DX	DY	DZ	…
1	2	3	4	5	6	7	8	9	…

For an elementary matrix, two conventions are necessary :

For a symmetrical matrix, one stores in the following order:

1				
2	3			
4	5	6		
7	8	9	10	
11	12	13	14	15

For a not-symmetrical matrix, one stores in the following order:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Knowing that the order of the lines and columns is the same one as that of an elementary vector.