Operator TO DESTROY

1 Goal

To destroy concepts users.

After destruction, the concept cannot thus be naturally more used in Lbe following orders. The destruction of an object is not so natural that can appear. It is important to understand its operation well describes in this documentation.

Destruction of unutilised objects allows to reduce obstruction of the files associated with the base 'TOTAL'. The mechanism of retassage is dealt with by the manager of memory during work. However, another mechanism of retassage can be started by the user using the keyword RETASSAGE = 'YES' within the procedure END [U4.11.02].

2 Syntax

```
TO DESTROY       (
                 ♦  NAME          = lco,           [l_co]

                 ◊  INFORMATION =    / 1,
                                     / 2,

       )
```

3 Operands

3.1 Keyword NAME

♦ NAME = lco

List of objects to destroy. One removes all the direct references to the objects provided in the context where TO DESTROY is called upon. See below detailed operation.

3.2 Operand INFORMATION

◊ INFORMATION = information

If INFO=2, the list of the destroyed objects is printed in the file MESSAGE.

4 How does the destruction function?

4.1 Example

One creates a list of realities of name `listr`

```
listr = DEFI_LIST_REEL (…)
```

One destroys the concept of name `listr`

```
TO DESTROY (NOM=listr)
```

Let us note that to re-use the same variable Python `listr` the same effect has as `TO DESTROY` .

4.2 In detail

The objects created by code_aster (a grid, a result) are Python objects.
`TO DESTROY` remove the references to the object in the command set. Python removes really this object when there is no more reference towards this object.

Let us take the example:

```
mesh = LIRE_MAILLAGE ()
alias = mesh
lst = [mesh]
```

An object was created *grid* , `mesh` is a variable which reference (or “which gives access to”, or “which points towards”) this object. `alias` is a variable which reference the same object. `lst` is a list, created with the operator [], who contains only one element, and this element reference also the same grid object.

If one makes:

```
TO DESTROY (NOM=mesh)
```

All the direct references towards the grid object are removed command set.
Thus, variables `mesh` and `alias` are removed.
Moreover, during the execution, one sees:

```
Suppression of the reference: 'mesh'
Suppression of the reference: 'alias'
```

On the other hand, the reference via the list is not seen (it would be necessary to traverse all the types of objects!) and thus, in accordance with the operation of Python, the grid object is not actually destroyed since there exists still a reference towards this object.

If the list is removed:

```
del lst
```

or the element of the list:

```
del lst [0]
```

Then there is no more reference towards the object which is then actually destroyed. One can see:

```
Deleting 00000004 <MAILLAGE_SDASTER> mesh
```

Let us note that:

```
TO DESTROY (NOM= lst [0] )
```

would have had exactly the same behavior (suppression of `mesh` and `alias` but not of `lst [0]`) because one passes the object to `TO DESTROY` and not the list `lst` or index 0.

There would be also the same behavior with a dictionary containing a grid object or any object referring the grid.

Indeed, there is only the container (the object lists, dictionary or other) which knows how to remove one of its elements.

4.3 Dependence between objects

Lbe structures of data (field to node-classification, field by element-model, etc…) the ones are based on the others. TO DESTROY a structure of data which is used by another thus does not have any immediate effect. On the other hand, when all the references to a structure of data are removed, this one then is actually removed.