PRADIS

WRITING [PLAGIN]- OBJECTS IN THE LANGUAGE PYTHON

PROGRAM SET FOR THE AUTOMATION OF THE SIMULATION OF NONSTATIONARY PROCESSES IN THE MECHANICAL SYSTEMS AND THE SYSTEMS OF OTHER PHYSICAL NATURE

VERSION 4.3

Content

1. Introduction.

In this document to be discussed the rules of writing [plagin] of objects (models and [prvp]) in the language python. On how to then connect them to the system catalog PRADIS, to speak in the document “use of a utility PARM”. In it so it is mentioned, that must precede the direct code on the python some commentaries, but here about them the speech not of [pojdjot].

2. Structure of the code.

The first line must be the following form:

```
from of pradis.ppl.model of import *
```

If you write [prvp], then “model” it is necessary to replace by “ovp”.

Class further must be declared. Its name will be the name of your object. This class must be inherited from the class *of model* (if you write model) or *of ovp* (if you you write [prvp]). For example:

```
class Of myOVP (ovp):
```

Further must be [pereopredeljon] the method *Of execute*. It has the input parameters. For the models and [prvp] they different. The determination of method for the model must appear thus:

```
def Of execute (COMMON, the I, Y, the X, the V, A, PAR, NEW, OLD, WRK):
```

But for [PRVP] thus:

```
def Of execute (COMMON, XOUT, PAR, WRK, DOF):
```

Further you write code itself, which realizes the logic of the work of your object. Here user has complete freedom to use all means of language python.

Output from the method *Of execute* must be strictly [opredeljonnym]. For the models it appears as follows:

```
        res = of return_result (COMMON, the I, Y, the X, the V, A, NEW, OLD,
WRK)
        return of res
```

For [PRVP] thus:

```
        res = of return_result (COMMON, XOUT, WRK)
        return of res
```

3. Components of the code.

This section is dedicated to the explanation of some components of writing the code, rules and possibilities.

0 parameters of the method Of execute.

Here we will give brief explanation to the parameters of the method *Of execute*.

3.1.2 models.

The declaration of method for the models appears as follows:

```
        def Of execute (COMMON, the I, Y, the X, the V, A, PAR, NEW, OLD,
WRK):
```

Here the parameters have the following values:

- **COMMON**: the structure, which contains the values of the unnamed [komon]- region of FORTRAN.
- **The I**: the vector of forces (moments) for the element.
- **Y**: the jacobian of the model of element.
- **The X**: the vector of displacements of the units of dimensionality EXT+ENT. It is not used with ADR=2, ADR=3.
- **The V**: the velocity vector of the units of dimensionality EXT+ENT. It is not used with ADR=3.
- **A**: the G-vector of the units of dimensionality EXT+ENT.
- **PAR**: the massif of the parameters of model.
- **NEW**: the vector “new state” of model.
- **OLD**: the vector “old state” of model.
- **WRK**: working massif for the model of element.

3.1.3 [PRVP].

The declaration of method for [PRVP] appears as follows:

```
    def Of execute (COMMON, XOUT, PAR, WRK, DOF):
```

Here the parameters have the following values:

- **COMMON**: the structure, which contains the values of the unnamed [komon]- region of FORTRAN.
- **XOUT**: the calculated output variable or the vector of the calculated output variables.
- **PAR**: the massif of the parameters [PRVP].
- **WRK**: working massif for [PRVP].
- **DOF**: massif with the values of degrees of freedoms.

4 functions of library S000J.

With writing of [plagin] of objects on the python to user are accessible to function the libraries S000J, which are frequently used, for example, in the models on FORTRAN. All functions, which were accessible on FORTRAN, are now accessible from the python. In this division we in detail will describe how them to use.

It is first of all necessary to connect the necessary library to your python- file:

```
from of s000j of import *
```

After this, to create the object of the corresponding class:

```
sj = S000J ()
```

Now will be accessible to you the methods of this class, which are called exactly on the names of necessary functions (S0001, S0002,…). And here there is one crucial point, at which it is worthwhile to focus attention. It is connected with transfer of the parameters to the methods and obtaining of results of them.

In FORTRAN in the function was transferred the collection of the variables, some of which served the transmission of data to the function, and others of it. The call of these functions from the python is characterized by the fact that in them it is necessary to transfer only those parameters, which transmit data into the function. Remaining data (emerging) of function return by list. Let us give a example.

Here one of such functions on FORTRAN:

```
      C the program of the calculation of the guides of cosines and
      C of the derived guides of the cosines of the axes
      C of the local coordinate system, built according to e to the points:
A, B, C.
      C
      C the first editorial staff of program 04/27/94 12:20 am
      C the date of last correction 09/27/95 08:26 am
      C
      C
               SUBROUTINE Of s0002 (
     , DXCA, DYCA, DZCA, DXCB, DYCB, DZCB,
     , DELTA,
     , LCA,
     , COSXX, COSYX, COSZX,
     , COSXY, COSYY, COSZY,
     , COSXZ, COSYZ, COSZZ,
     , DCOS,
     , CODIN, CODOUT)
      C
      C the input parameters:  DXCA, DYCA, DZCA, DXCB, DYCB, DZCB,
      C DELTA,
      C CODIN
      C the output parameters: LCA,
      C COSXX, COSYX, COSZX,
      C COSXY, COSYY, COSZY,
      C COSXZ, COSYZ, COSZZ,
      C DCOS,
      C CODOUT
```

With its call we [peredajom] only input parameters:

```
lstout = of sj.S0002 (DXCA, DYCA, DZCA, DXCB, DYCB, DZCB, DELTA, CODIN)
```

Now *lstout* - this is the list, which contains all output parameters. In order to obtain them it is possible to make the following:

```
LCA = of lstout [0]
COSXX = of lstout [1]
COSYX = of lstout [2]
COSZX = of lstout [e]
COSXY = of lstout [4]
COSYY = of lstout [shch]
COSZY = of lstout ']
COSXZ = of lstout ["]
COSYZ = of lstout [8]
COSZZ = of lstout [9]
DCOS = of lstout [10]
CODOUT = of lstout [of 11]
```

User, of course, can devise another method to use this list.

5 conclusion of the errors of models.

With writing of [plagin] of models on the python the user can use a accessible collection of templates for the communications about the errors. The complete list of templates can be seen in the document “accessible errors of models”. Here we will describe how to use accessible templates.

Let us assume you wanted to derive error, using template 1003:

```
1003 08 E (M 004) (/T4, "the incorrect value of the parameter:",/, T8,
     "the parameter", F6.0, "must be >=", G11.5,/, T8, "the
     parameter", F6.0, "=", G11.5)
```

As you see, this template requires 4 numbers: 2 wholes even 2 real (being encountered Of f6.0 - this integers, G11.5 - real). If we assign these numbers by values 1, 2.2, 1, 1.1, then on the screen this will appear then:

```
Incorrect value of the parameter:
    parameter 1. must be >= 2.2000
    parameter 1. = 1.1000
```

In order to use a template, in the python- file of your model it is necessary to make the following.

To connect the necessary libraries:

```
from Of pradisLog of import *
from of array of import *
```

The library *Of pradisLog* is supplied with the complex PRADIS, and *array* is standard [pitonovskoj] library. It is further necessary to create the object of the necessary class:

```
pl = PradisLog ()
```

Now in that place, where you should derive error, it is necessary to write the following:

```
a = of array ("d", [1, 2.2, 1, 1.1])
pl.perr (1003, 4, a.buffer_info () [0])
```

With the fulfillment of these lines on the screen will appear precisely that was indicated above. The first line of [sozdajot] the massif of real numbers, which is filled up with those numbers, which you want to see in the template with the conclusion of communication. A quantity of these numbers must strictly coincide with a quantity of those required in the template. The second line uses a method *of perr* of the class *Of pradisLog*. In its parameters should be indicated the number of template, a quantity of numbers in the template and the address of the beginning of the massif of the parameters.

In a example we named variable "a" and "pl", but user, of course, can call them in its own way.

4. Examples.

Let us here give a example the python of model. This is the already existing in the library of complex model MD. This is how it could appear on the python:

```
# Library is necessary for writing of the model
from of pradis.ppl.model of import *

# Libraries for the conclusion of the errors
from Of pradisLog of import *
from of array of import *

# Library for using the functions S000J
from of s000j of import *

# We declare class MD and indicate that this is the model
class MD (model):

    # [Pereopredeljaem] the method Of execute (compulsorily)
    def Of execute (COMMON, the I, Y, the X, the V, A, PAR, NEW, OLD, WRK):

        # We further write the essence of the model
        # The first cut
        if COMMON.NEWINT == 1:

            # [Sozdajom] object for using the functions S000J
            sj = S000J ()

            # It is utilized the necessary function
            d = of sj.S0005 (0.1, 0.2, 0.3, 0.4, 0.5, 1)
            print d

            # [Sozdajom] object for the conclusion of the errors
            pl = PradisLog ()
            ERR = 0
```

```
                  if PAR [1] < 0.:
                        ERR = 1
                        if COMMON.SYSPRN > 0.:
                              # It is concluded error to template 1003
                              a = of array ("d", [1, 0, 1, PAR [1]])
                               pl.perr (1003, 4, a.buffer_info () [0])

                  if PAR [2] < 0.:
                        ERR = 1
                        if COMMON.SYSPRN > 0.:
                              # It is concluded error to template 1003
                              a = of array ("d", [2, 0.2, PAR [2]])
                               pl.perr (1003, 4, a.buffer_info () [0])

                  if ERR == 1:
                        if COMMON.CODE < 100.:
                              COMMON.CODE = 100.

                              # We leave from the model
                             res = of return_result (COMMON, the I, Y, the X, the V,
A, NEW, OLD, WRK)
                              return of res


            I [1] = A [1] * PAR [1]
            I [2] = A [2] * PAR [1]
            THE I [E] = A [E] * OF PAR [2]

            Y [1] = PAR [1]
            Y [2] = 0.
            Y [e] = 0.
            Y [4] = 0.
            Y [SHCH] = OF PAR [1]
            Y ['] = 0.
            Y ["] = 0.
            Y [8] = 0.
           Y [9] = PAR [2]

            # We leave from the model
           res = of return_result (COMMON, the I, Y, the X, the V, A, NEW, OLD,
WRK)
           return of res
```