PRADIS

THE DESCRIPTION OF UTILITIES

THE SOFTWARE FOR SIMULATION OF NON-STATIONARY PROCESSES IN MECHANICAL SYSTEMS AND SYSTEMS OF OTHER PHYSICAL NATURE

VERSION 4.3

Contents

1. Use of the console solver's utility SLANG.EXE

a. Format of start

Format of console solver start: slang [options] name_of_file1 [name_of_file2]. The files corresponding to names name_of_file1 and name_of_file2 should be specified with absolute ways or with ways concerning current directory. If two names are specified, all corresponding files should settle down in the same directory.

Name_of_file1 should be always there , it is the name of PRADISlang text with the task for calculation, and if there is only one task there can be the description of model.

Name_of_file2 can be absent. If it is present, it is perceived as a name of a text PRADISlang which was calculated earlier(there should be files name_of_file2.VAR, name_of_file2.TRN, name_of_file2.IID, name_of_file2.OID, name_of_file2.MID).

b. The list and the destination of the keys

The list of options (keys) of start, any of them can be absent:

- -pgoN - the record of graphical 3D information in a file (ΠΓΟ file), N means the counter of shown points (it shows each N-th point) if N is not set then N=1
- -s or-e (by default). A format of data output during the calculation (see below).–s means a short format,-e – the expanded format. If any of options is not specified, the expanded format (-e) is used.
- -r (default), or-m. The option concerns to frequency of output during the calculation. If -r is specified, the frequency is calculated on real time, if it is -m - on time of model. The data output is made when time after the last conclusion exceeds frequency of a conclusion in seconds. Value of frequency of a conclusion is taken from parameter of the program of integration PRTTIME in text PRADISLANG.

c. The review of information during the calculation

Slang can print a data in a short format (the -s option, only the time of model is shown), or in the expanded format (an -e option, by default).

- Current time of model
- Real time of calculation that had passed
- Expected real time before the end of calculation
- Current step of integration by time
- Quantity of successful and unsuccessful Euler's iterations
- Quantity of successful and unsuccessful Newton's iterations
- Current values of variables displayed on-the-fly which are specified in parameters of the program of integration (PRADISLang)

d. Output files

During the calculation the solver creates following output files (names coincide with a name of an initial file of the description of a model):

- RSL и PNM contain the data of output bidimentional curves and can be looked through by means of the postprocessor.
- PGO contains a 3D scene visualizing the behaviour of model during calculation. Can be looked through by means of a player of the postprocessor.
- TRN, VAR, IID, MID, OID intermediate binary files of a solver. They are necessary for the subsequent calculation of already generated model.

e. Interruption of calculation

During iterations on time, it is possible to finish calculation correctly by means of a combination of keys Ctrl+C. Output files RSL and PGO are closed and can be opened for viewing. After interruption (as well as after usual finish) it is possible to cause continuation of calculation (PRADISLang with the task with presence $RESTORE). Files will continue to work from the interrupted place.

2. Use of the DELCOMMENT.EXE utility

By means of utility DelComment it is possible to delete automatically comments from job files in language PradisLang.

Use:

At start of the utility without parametres, on a console the short instruction on its use is inferred:

```
C:\DINAMA\pradis32> delcomment.exe
Use: delcomment <имя1> [<имя2> [имя3... [имяN]]]]
```

Procedure of removal of comments from files of jobs.

```
<имя1... N> - names of files of jobs.
```

The utility can set in the capacity of parametres some names of files. Thus for each of them the following becomes:

1. All comments in a job file are deleted.
2. The new job is recorded in a file del _ <a name of a file of the job>
3. The same becomes recursively for all $INCLUDE files in the job
4. The file title in $INCLUDE also is renamed

Instance:

In folder DINAMA\TEST\KN3EF the test job contains. «00-hello» - a file of the job for обсчёта utility Slang.exe. In it you can discover a line:

```
$INCLUDE:sq1.txt
```

And in a file «sq1.txt» you can see in the end a line:

```
$INCLUDE: sq1f.txt
```

In this folder set in a command line a command «delcomment 00-hello».
After that you will have 3 new files:

del_00-hello
del_sq1.txt
del_sq1f.txt

In a file del_00-hello the line will contain:

```
$INCLUDE:del_sq1.txt
```

In a file del_sq1.txt the line will contain:

```
$INCLUDE:del_sq1f.txt
```

Without comments del_00-hello it is possible as to give the gained job on calculation to the utility slang.exe.

Possible errors:

Utility Delcomment does not pay attention to syntactical errors in the job, it only deletes comments. Therefore one error is possible only – «the file of the job with the set name does not exist». If you wrongly write the title of a file of the job delcomment will give out this error in parametres. However the utility will continue operation if one file of the job has been set not.

As there can be two informative messages on aberrant comments:

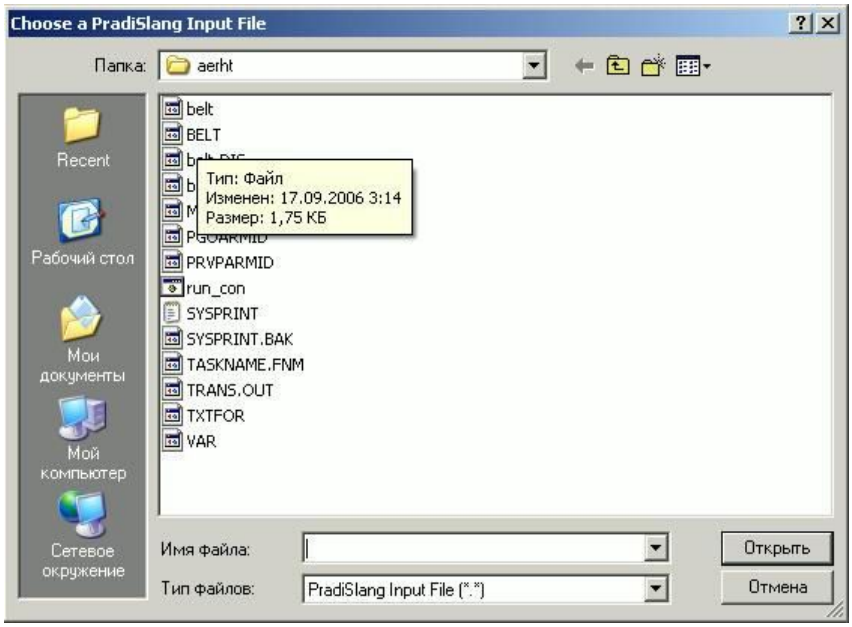
```
«In a file <taskname> неокончен the comment»
«In a file <taskname> a superfluous bracket}»
```

However these messages do not discontinue operation delcomment, and files with a prefix “del _” all the same will appear. But their contents can mismatch your expectations.

3. Use of the PRADISW.EXE utility

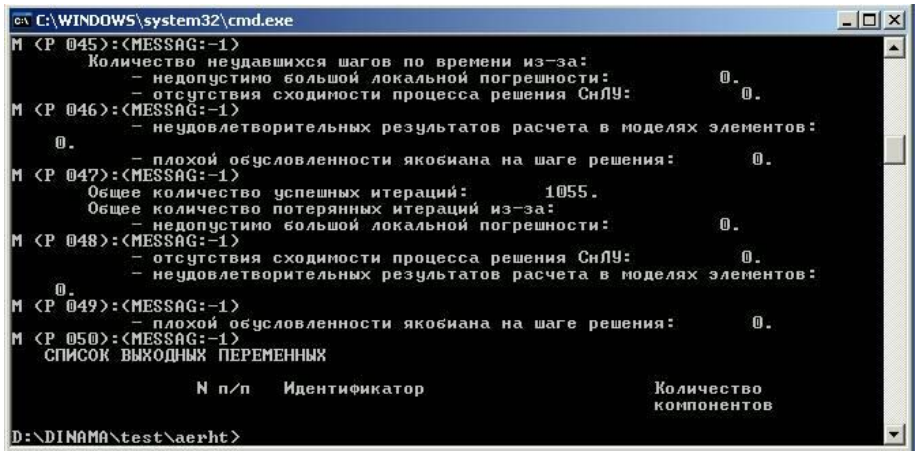
The utility pradisw is intended for start of tasks for calculation in a window mode.

For start of the application it is necessary to start a file pradisw.exe by means of a corresponding icon on a desktop. Further - to specify a way to a file with the description of the task and to press «Open»:



For start of the task for calculation press «Run» in the opened window:

Performance of calculation:



4. Use of utility OUTFILE

The utility outfile is intended for a leading-out in a text file of values of output variables an OVP from DAT a file.
The first parametre of the utility is the path to DAT to a file, the second parametre is the path to an output text file.
For example, if to start a command:

outfile.exe C:\dinama\pradis32\swing.dat out.txt

That as a result of its operation in leaking directories the text file with a name out.txt, containing output datas of job SWING in the table shape will be created.

File fragment out.txt:

```
Calculation of a spring pendulum

Time (s) Migration т.A on a X-axis
                        3.001927-1.123834
                        3.004446-1.129830
                        3.007756-1.137682
                        3.012654-1.149243
                        3.020344-1.167239
                        3.030344-1.190343
                        3.040344-1.213088
                        3.050344-1.235454
                        3.060344-1.257418
                        3.070344-1.278961
                        3.080344-1.300064
                        3.090344-1.320711
                        3.100344-1.340883
                        3.110344-1.360565
                        3.120344-1.379744
                        3.130344-1.398406
                        3.140344-1.416538
                        3.150344-1.434130
                        3.160344-1.451171
                        3.170344-1.467652
                        3.180344-1.483565
                        3.190344-1.498903
                        3.200344-1.513659
                        3.210344-1.527827
                        3.220344-1.541403
                        3.230344-1.554383
                        3.240344-1.566763
                        3.250344-1.578541
                        3.260344-1.589714
                        3.270344-1.600281
                        3.280344-1.610240
                        3.290344-1.619591
                        3.300344-1.628333
                        3.310344-1.636466
                        3.320344-1.643991
```

5. Utility use armdoc

a. Introduction.

Utility ARMDOC is intended for support of the contained system catalogue in the form of XML structures (structure of folders with xml-files), and as documentation shapings in HTML a format on all installations of the system catalogue. It allows to get convenient access to any information on any installation.

Further in the deed will be more detailed it is told about XML structure of the system catalogue and formats xml files.

b. Structure of the system catalogue.

All information in XML a format about the system catalogue is stored in catalogue *DINAMA/sysarm/XML*. There the master file *sysarm.xml*, containing the information on existing modules is had. For each of the modules enumerated there it is created the catalogue named a name of the module, the information about which contains. The module catalogue contains xml a file named a name of the module, presenting the elements containing in the module, and as six catalogues: *Model*, *OVP*, *Image*, *Node*, *Parameter*, *Object*. These catalogues accordingly contain xml files with the description of models, првп, пго, nodes, parametres and the pythons-installations which are available in the module.

Thus, we have the following structure: the system catalogue contains modules, modules contain installations which share on six types.

More in detail about structure XML of files it will be told more low.

c. Utility possibilities

At start without arguments the instruction on the use, is short speaking about utility possibilities is given out.

```
C:\DINAMA\pradis32> armdoc
Use: armdoc <key> <name>
    <Key>:
        -a to Add installation or the module.
        -r to Delete installation or the module.
        -g to Inferred the module or installation description.
        -d to Generate the documentation on installation or the module.
        -h to Inferred this inquiry.
    Name a name of a xml-file of the module or installation with expansion or
without.
```

From this inquiry it is visible, that, using the utility, we have a possibility:

1. To add new installation or even the new module in system XML the catalogue.
2. To delete from it available installation or all module bodily.
3. To gain the short description of installation or the module and lists of those elements which it contains.
4. To generate HTML the documentation on installation or the module.
5. And to inferred this inquiry.

By the way, at the direction of an aberrant key, you as gain this inquiry on utility use.

Except a key in utility parametres there is a line parametre <*name*>. It is necessary to be shut down more in detail on a format of this parametre.

At its use it is possible to gate out three cases:

1. It is absolutely not necessary at the key direction "-h".
2. At the key direction "-a" it is a name of a file with expansion ".xml" or without it. If to set a name of a file without expansion the utility will add it when will search for the necessary file. Thus the file should present installation of the system catalogue in XML a format, but the file name can be not linked in any way with a name of the most presented installation. For example, the model *mymdl* from the module *mymodule* can be presented in a file *123.xml*
3. At the direction of remaining keys this parametre should be a name of installation of the system catalogue. How this name looks, we will tell explicitly.

Installation of the system catalogue can be either the module, or installation of the module. If we wish to specify the module simply we specify its title. If we wish to specify in installation of any module we specify a name of the module and a name of the installation necessary to us through a point ("."). In the end of this parametre as it is possible to assign expansion ".xml", the utility correctly will treat it. Here some instances of names of installations of the system catalogue:

- *abcd.xml* - The module *abcd*
- *qwe.rty* - Installation *rty* the module *qwe*
- *module_12.node_34.XmL* - Installation *node_34* the module *module_12*

Now we will tell about each of utility possibilities hardly more in detail.

d. Adding.

To add in system XML the catalogue it is possible installations of various types:

- The module
- Model
- OVP
- GIP
- Node
- Parametre
- Python-object

For each of them the structure xml a file transmitted to the utility in the capacity of of the second parametre is defined. At adding of installations in system XML the catalogue it is necessary to observe one condition: names of installations of the module should **be unique** among other installations **of the same type** in **the given module**.

Let's observe more in detail formats XML of files.

i. The module.

```
<module name = "ModuleName">
     <description>
          <russian>
               <! - the module description in Russian->
          </russian>
          <english>
               <! - the module description in English->
```

```
          </english>
     </description>
</module>
```

As you can see, the module file should contain only a name (attribute *name* a tag *module*) both the description in Russian and English languages in an arbitrary aspect.

ii. Python-object.

```
<object name = "ObjectName" module = "ModuleName">
     <description>
          <russian>
               <! - the python-installation description in Russian->
          </russian>
          <english>
               <! - the python-installation description in English->
          </english>
     </description>
     <fieldlist>
          <! - Перечесление elements->
          <field name = "FieldName" type = "FieldType">
               <description>
                    <russian>
                         <! - the element description in Russian->
                    </russian>
                    <english>
                         <! - the element description in English->
                    </english>
               </description>
          </field>
          ...
     </fieldlist>
</object>
```

The python-installation file should contain an installation name, a module name (attributes *name* and *module* a tag *object*) both the description in Russian and English languages. Also it contains the list of elements going into in a python-installation (tags *field*) with names, types and descriptions in two languages.

iii. Parametre.

```
<parametre name = "ParameterName" module = "ModuleName">
     <description>
          <russian>
               <! - the parametre description in Russian->
          </russian>
          <english>
               <! - the parametre description in English->
          </english>
     </description>
     <fieldlist>
          <! - Перечесление elements->
          <field name = "FieldName" type = "FieldType">
               <description>
                    <russian>
                         <! - the element description in Russian->
                    </russian>
                    <english>
```

```
                    <! - the element description in English->
                </english>
            </description>
        </field>
        ...
    </fieldlist>
</parameter>
```

The parametre file is almost identical to a python-installation file except that begins with a tag *parametre*. Instances of files of parametre will be resulted further in the deed.

iv. Node.

```
<node name = "NodeName" module = "ModuleName">
    <description>
        <russian>
            <! - the node description in Russian->
        </russian>
        <english>
            <! - the node description in English->
        </english>
    </description>
    <fieldlist>
        <! - Перечесление elements->
        <field name = "FieldName" type = "FieldType">
            <description>
                <russian>
                    <! - the element description in Russian->
                </russian>
                <english>
                    <! - the element description in English->
                </english>
            </description>
        </field>
        ...
    </fieldlist>
</node>
```

The node file is almost identical to python-installation and parametre files except that begins with a tag *node*. Instances of files of a node will be resulted further in the deed.

v. Model.

```
<model name = "ModelName" module = "ModuleName" ext = "1" ent = "1" par =
"1" vpr = "1" str = "1" stp = "1" wrk = "1" wrp = "1" ign = "1" adr =
"1">
    <description>
        <russian>
            <! - the model description in Russian->
        </russian>
        <english>
            <! - the model description in English->
        </english>
    </description>
    <nodelist>
        <! - Перечесление nodes->
        <node name = "NodeName" type = "NodeType">
            <description>
```

```
                    <russian>
                        <! - the node description in Russian->
                    </russian>
                    <english>
                        <! - the node description in English->
                    </english>
                </description>
            </node>
            ...
        </nodelist>
        <parameterlist>
            <! - Перечесление parametres->
            <parametre name = "ParameterName" type = "ParameterType">
                <description>
                    <russian>
                        <! - the parametre description in Russian->
                    </russian>
                    <english>
                        <! - the parametre description in English->
                    </english>
                </description>
            </parameter>
            ...
        </parameterlist>
        <worklist>
            <! - Перечесление elements of a working vector->
            <parametre name = "ParameterName" type = "ParameterType">
                <description>
                    <russian>
                        <! - the parametre description in Russian->
                    </russian>
                    <english>
                        <! - the parametre description in English->
                    </english>
                </description>
            </parameter>
            ...
        </worklist>
        <statelist>
            <! - Перечесление state vector elements->
            <parametre name = "ParameterName" type = "ParameterType">
                <description>
                    <russian>
                        <! - the parametre description in Russian->
                    </russian>
                    <english>
                        <! - the parametre description in English->
                    </english>
                </description>
            </parameter>
            ...
        </statelist>
</model>
```

The model file contains:

- Installation name - a tag *model*, attribute *name*
- Module name - a tag *model*, attribute *module*
- Parametres of the certificate of model - attributes of a tag *model*
- The model description - tag contents *description*
- The list of nodes of model (a tag *nodelist*) with names, types and descriptions - tags *node* with attributes *name* and *type* and the inserted tags *description*.

- Model argument list (a tag *parameterlist*) with names, types and descriptions - tags *parametre* with attributes *name* and *type* and the inserted tags *description*.
- The list of elements of a working vector of model (a tag *worklist*) with names, types and descriptions - tags *parametre* with attributes *name* and *type* and the inserted tags *description*.
- The list of elements of state vector of model (a tag *statelist*) with names, types and descriptions - tags *parametre* with attributes *name* and *type* and the inserted tags *description*.

Parametres and model nodes have names and types. It is necessary to illustrate their value. The name is a title of a node or parametre in model, it is arbitrary and is not unique. The type is an index on what is a node or parametre, i.e. one of titles of nodes or parametres of the given module.

It is necessary to pay attention especially that at model adding (and also првп or пго) there is a check of types of nodes and parametres. For example, if the module contains two parametres with names *param1* and *param2*, and you try to add model with type of one of parametres *param3* there will be a diagnostic message for an error.

vi. OVP.

```
<ovp name = "OVPName" module = "ModuleName" out = "1" par = "1" vps = "1"
vpr = "1" wrk = "1" wrs = "1" wrp = "1" sys = "1">
     <description>
          <russian>
               <! - the description првп in Russian->
          </russian>
          <english>
               <! - the description првп in English->
          </english>
     </description>
     <nodelist>
          <! - Перечесление nodes->
          <node name = "NodeName" type = "NodeType">
               <description>
                    <russian>
                         <! - the node description in Russian->
                    </russian>
                    <english>
                         <! - the node description in English->
                    </english>
               </description>
          </node>
          ...
     </nodelist>
     <parameterlist>
          <! - Перечесление parametres->
          <parametre name = "ParameterName" type = "ParameterType">
               <description>
                    <russian>
                         <! - the parametre description in Russian->
                    </russian>
                    <english>
                         <! - the parametre description in English->
                    </english>
               </description>
          </parameter>
          ...
     </parameterlist>
</ovp>
```

The file првп is almost identical to a model file except that begins with a tag *ovp* and has inherent првп certificate parametres (attributes of a tag *ovp*) and as has no tags *worklist* and *statelist*.

vii. GIP

```
<image name = "ImageName" module = "ModuleName" ext = "0" par = "0" wrk =
"0">
     <description>
          <russian>
               <! - the description пго in Russian->
          </russian>
          <english>
               <! - the description пго in English->
          </english>
     </description>
     <nodelist>
          <! - Перечесление nodes->
          <node name = "NodeName" type = "NodeType">
               <description>
                    <russian>
                         <! - the node description in Russian->
                    </russian>
                    <english>
                         <! - the node description in English->
                    </english>
               </description>
          </node>
          ...
     </nodelist>
     <parameterlist>
          <! - Перечесление parametres->
          <parametre name = "ParameterName" type = "ParameterType">
               <description>
                    <russian>
                         <! - the parametre description in Russian->
                    </russian>
                    <english>
                         <! - the parametre description in English->
                    </english>
               </description>
          </parameter>
          ...
     </parameterlist>
     <worklist>
          <! - Перечесление elements of a working vector->
          <parametre name = "ParameterName" type = "ParameterType">
               <description>
                    <russian>
                         <! - the parametre description in Russian->
                    </russian>
                    <english>
                         <! - the parametre description in English->
                    </english>
               </description>
          </parameter>
          ...
     </worklist>
     <statelist>
          <! - Перечесление state vector elements->
```

```
            <parametre name = "ParameterName" type = "ParameterType">
                <description>
                    <russian>
                        <! - the parametre description in Russian->
                    </russian>
                    <english>
                        <! - the parametre description in English->
                    </english>
                </description>
            </parameter>
            ...
        </statelist>
    </image>
```

The file пго is almost identical to a model file except that begins with a tag *image* and has inherent пго certificate parametres (attributes of a tag *image*).

e. Removal.

To delete from system XML the catalogue it is possible any installation, including the whole modules with all contents. Already it was said above, that after a key "-r" it is necessary to specify a name in a format of installation of the system catalogue (the format is presented above).

Here it is necessary to note only one moment. It is impossible to delete installation of the module if other installations of this module refer to it. For example. Let the module *module1* contains a node *node1* and model *model1* which has a type node *node1*. In this case, if you give a command:

```
armdoc-r module1.node1
```

That gain a diagnostic message for an error since the model *model1* refers to this node.

f. The inquiry on installation of the system catalogue.

Here only it is necessary to mention, that after a key it is possible not to set a name of installation of the system catalogue. In this case the list of all modules will be inferred. If the second parametre is set, the brief information on installation will be inferred. For each type of installation the.

- For the module - its description and lists of models, првп, пго, nodes, parametres and poton-installations.
- For model, првп and пго - the description and lists of nodes and parametres (names and types).
- For a node, parametre and python-installation - the description and the list of fields (a name, type, the description).

g. Generation HTML of the documentation.

Principal page HTML of the documentation on PRADIS is in a file: DINAMA/docs/HTML/index.html. From this page any documentation which is referring to to PRADIS is accessible. Including the information on the system catalogue. Having followed the link "Modules", you will hit in the list of all modules of the system catalogue. Further you can come into the concrete module, its any installation etc. These HTML pages represent contents system XML the catalogue in a user-friendly aspect.

You can add in the system catalogue a new plug-in installation. Utility ARMDOC to add its description in system XML the catalogue. After that, that the description of this installation has appeared in HTML documentation, it is necessary to generate it, having specified after a key "-d" a name of your installation in a format of installation of the system catalogue.

For example. You wish to add new module *MyModule*. You write xml file *MyModule.xml* (or any other name, but expansion is obligatory) for this module (the file format is presented above) and give a command:

```
armdoc-a MyModule
```

After that oscillate the documentation:

```
armdoc-d MyModule
```

Now you wish to add parametre *MyParameter* in the new module. You write xml file *NewParameter.xml* (or any other name, but expansion is obligatory) for this parametre (the file format is presented above) and give a command:

```
armdoc-a NewParameter.xMl
```

Then oscillate the documentation:

```
armdoc-d MyModule. MyParameter
```

Now, having come on page HTML of the documentation, you will see there all information on the new module and new parametre.

However, if you do not remember title of installations which added, or them was much, or you simply would not like to gather their names, you can not specify anything after a key "-d". Then the utility will generate the documentation on all system XML to the catalogue bodily. But it, of course, will occupy more time.

h. Instances.

In this point we will display instances xml the files intended for adding in system XML the catalogue by utility ARMDOC.

i. The module.

```
<module name = "pneumatics">
     <description>
          <russian> the Module containing installations, linked with
pneumatics.
          </russian>
          <english> The module about pneumatics.
          </english>
     </description>
</module>
```

The pneumatics module.

ii. Python-object.

```
<object name = "something" module = "pneumatics">
```

```
     <description>
          <russian> Any installation.
          </russian>
          <english> Some object.
          </english>
     </description>
     <fieldlist>
          <field name = "Field1" type = "main">
               <description>
                    <russian> the First field.
                    </russian>
                    <english> First field.
                    </english>
               </description>
          </field>
          <field name = "Field2" type = "main">
               <description>
                    <russian> the Second field.
                    </russian>
                    <english> Second field.
                    </english>
               </description>
          </field>
     </fieldlist>
</object>
```

iii. Node.

```
<node name = "trans_point3d" module = "pneumatics">
     <description>
          <russian>  the  Three-dimensional  point  of  a  translational
motion.
          </russian>
          <english> 3D point of translation.
          </english>
     </description>
     <fieldlist>
          <field name = "x" type = "translation">
               <description>
                    <russian> the Degree of freedom of postth driving on a
X-axis.
                    </russian>
                    <english> Degree of freedom of X translation.
                    </english>
               </description>
          </field>
          <field name = "y" type = "translation">
               <description>
                    <russian> the Degree of freedom of postth driving on a
Y-axis.
                    </russian>
                    <english> Degree of freedom of Y translation.
                    </english>
               </description>
          </field>
          <field name = "z" type = "translation">
               <description>
                    <russian> the Degree of freedom of postth driving on a
Z-axis.
                    </russian>
                    <english> Degree of freedom of Z translation.
                    </english>
```

```
            </description>
        </field>
    </fieldlist>
</node>
```

iv. Parametre.

```
<parametre name = "material" module = "pneumatics">
    <description>
        <russian> Properties of a material.
        </russian>
        <english> Material properties.
        </english>
    </description>
    <fieldlist>
        <field name = "density" type = "real">
            <description>
                <russian> a material Denseness.
                </russian>
                <english> Material density.
                </english>
            </description>
        </field>
        <field name = "elasticity" type = "real">
            <description>
                <russian> a material Modulus.
                </russian>
                <english> Material elasticity.
                </english>
            </description>
        </field>
        <field name = "yung" type = "real">
            <description>
                <russian> material Modulus of elongation.
                </russian>
                <english> Yung module of material.
                </english>
            </description>
        </field>
    </fieldlist>
</parameter>
```

v. Model.

```
<model name = "thread" module = "pneumatics" ext = "1" ent = "1" par =
"1" vpr = "1" str = "1" stp = "1" wrk = "1" wrp = "1" ign = "1" adr =
"1">
    <description>
        <russian> a Filament, linking two point.
        </russian>
        <english> Thread connecting two points.
        </english>
    </description>
    <nodelist>
        <node name = "end1" type = "trans_point3d"> </node>
        <node name = "end2" type = "trans_point3d"> </node>
    </nodelist>
    <parameterlist>
        <parametre name = "material" type = "material"> </parameter>
```

```
        </parameterlist>
    </model>
```

This model does not claim for were real, it simplis instance xml a model file. Before its adding in system XML the catalogue you should add a node *trans_point3d* and parametre *material*.

vi. OVP.

```
<ovp name = "x" module = "pneumatics" out = "1" par = "1" vps = "1" vpr =
"1" wrk = "1" wrs = "1" wrp = "1" sys = "1">
    <description>
        <russian> Value of a degree of freedom.
        </russian>
        <english> The value of degree of freedom.
        </english>
    </description>
    <nodelist>
        <node name = "dof" type = "dof"> </node>
    </nodelist>
    <parameterlist>
        <parametre name = "scale" type = "real"> </parameter>
    </parameterlist>
</ovp>
```

Adding of it assumes an OVP, that the module pneumatics already contains a node in the system catalogue *dof* and parametre *real*.

vii. GIP

```
<image name = "thread" module = "pneumatics" ext = "0" par = "0" wrk =
"0">
    <description>
        <russian> the Image for a filament.
        </russian>
        <english> The image of thread.
        </english>
    </description>
    <nodelist>
        <node name = "end1" type = "trans_point3d"> </node>
        <node name = "end2" type = "trans_point3d"> </node>
    </nodelist>
    <parameterlist>
        <parametre name = "start1" type = "StartCoord3d"> </parameter>
        <parametre name = "start2" type = "StartCoord3d"> </parameter>
    </parameterlist>
</image>
```

Adding of this assumes the GIP, that the module pneumatics already contains parametre *StartCoord3d* (co-ordinates of the origin of 3 measured points) in the system catalogue. This the GIP as does not claim for were real.

6. Utility use parm

a. Introduction.

Utility PARM is intended for turning on in system a plug-in of the installations written on the Python. Models and првп can be installations. The user has a possibility to write models not only on a Fortran, but also on a python. For adding in system of the installations written on a Fortran, utility ARM, and written on a python - utility PARM presented in the given deed is used.

Added in system a python-plug-in installations the user can use just as any others in the jobs written on PSL or PPL. How to write a plug-in installations on a python, the Spelling a plug-in of installations in language Python "is spoken in the deed".

b. Python-repository.

For added about system the python-plug-in of installations is carried on a special python-repository. It is in catalogue DINAMA/plugin/python and has the following structure.

Each of models or првп belongs to one of modules. Therefore the python-repository begins with the catalogue *pradis* which contains the catalogues named on names of all modules available in a python-repository. Each of them, in turn, contains 2 catalogues: *model* and *ovp*. At installation adding in a python-repository there are patterned a file with the text of a code and a file with откомпилированным a code.

For example if you add model "aaa" in the module "bbb", in a python-repository there will be 2 files:

DINAMA/plugin/python/pradis/bbb/model/aaa.py

DINAMA/plugin/python/pradis/bbb/model/aaa.pyc

Also for repository support file DINAMA/sysarm/python_plugin.xml which the python-plug-in of installations contains titles of all available in system is used.

c. Utility possibilities

At start without arguments the instruction on the use, is short speaking about utility possibilities is given out.

```
C:\DINAMA\pradis32> parm
Use: parm <key> <name>
Procedure of operation with the Python components PRADIS.
    <Key>:
        + Switches on components in the catalogue and compiles a component
        # Only compiles a component
        ! Switches on components in the catalogue without compilation
        - Expels components from the catalogue
        ? Infers the inquiry on the components containing in the binary
catalogue
    Name a name the Python of a component
```

From this inquiry it is visible, that, using the utility, we have a possibility:

6. To add a new python-plug-in installation in system.

7. Откомпилировать a code written for a python-plug-in of installation.
8. To add a new python-plug-in installation, not compiling it.
9. To delete from system an available python-plug-in installation.
10. To gain the description on available in system a python-plug-in to installation.

By the way, at the direction of an aberrant key, you as gain this inquiry on utility use.

Except a key in utility parametres there is a line parametre <*name*>. This title of installation (component). At use of keys "+", "#" and "!" The name can be added by expansion ".py" as in these cases it is a name of a file with a code of the program written on a python.

Now we will tell about each of utility possibilities hardly more in detail.

i. Adding.

Key "+".

At adding a python-plug-in of installations in system it is necessary to observe some rules about which it will be spoken in this section.

The file with a python-code should consist of three parts.

The first part. This direction of the coding used in a file. It should go in the first line of a file. For example:

```
# coding=Windows-1251
```

Pay attention, that the line is preceded by a numeral "#", the marking out comment in language Python as following it is not the python a command but only specifies to utility PARM, what coding to use for reading of a remaining file.

The second part. It is text XML of the file presenting installation which you add in system. As it was already spoken, the model or првп can be installation. About XML formats for models and првп explicitly it is written in the deed "Use of utility ARMDOC". Here we only will note, that each line of the text should follow numerals "#HELP" which specify that these lines are the inquiry on added installation. For example:

```
#HELP <model name = "ModelName" module = "d3" ext = "3" par = "2"
adr = "3">
...
#HELP </model>
```

This text XML of a file will be used for information adding in system XML the catalogue about which also it is spoken in the aforementioned deed.

Third. It immediately code of the program on a python, making an installation essence. How to write a code on a python, in the deed "the Spelling a plug-in of installations in language Python" explicitly is written. Here it is necessary to mention only pair of the moments.

The class-room name on a python should coincide match case with attribute *name* in title tag XML of the text. A title tag can be either *model*, or *ovp*. And this tag should coincide with a name of the parent class-room for your python-class-room.

For example, if you have written a title tag:

```
<model name = "nAmE"...>
```

That announcement of the class-room on a python should look so:

```
class nAmE (model):
```

And if a title tag such:

```
<ovp name = "NAme"...>
```

That announcement of the class-room:

```
class NAme (ovp):
```

Also at adding of installations the name of the file gived to the utility in the capacity of of the second parametre, should coincide with an installation name match case. I.e. the model *mDl* can be presented only in a file *mDl.py*.

ii. Compilation.

Key "!".

Compilation changes nothing in the system catalogue. At its execution it only substitutes files already existing in a python-repository on new if they were, or simply patterns them in a python-repository (beforehand откомпилировав).

This option is intended for those cases when the user already has a python-plug-in installation, but wishes to change something in its operation. Then the user needs to correct a code in language a python and to use utility PARM with a key "!".

iii. Adding without compilation.

Key "#".

This option is absolutely identical to the option of adding presented in point 2.1 except that the code is not compiled, and, accordingly, the file.pyc is not patterned in a python-repository.

The user can use this option in case he needs to add installation in the system catalogue, but its operation to it is not important, or does not wish to adjust a code.

iv. Removal.

Key "-".

Here almost about what to speak. The option serves for removal from the system catalogue of installations available a python-plug-in.

v. The help

Key "?".

This option allows to inferred the description of installation available a python-plug-in. The description undertakes from the text xml a file about which it was spoken in point 2.1.

d. Instances.

Here we will instance the text of a file which can be used for adding of model *NewModel* new a python-plug-in.

```
# coding=Windows-1251
#HELP <model name = "NewModel" module = "d3" ext = "3" par = "2" adr = "3">
#HELP <description>
#HELP <russian> Model NewModel </russian>
#HELP <english> Description NewModel </english>
#HELP </description>
#HELP <nodelist>
#HELP </nodelist>
#HELP <parameterlist>
#HELP </parameterlist>
#HELP </model>

from pradis.ppl.model import *

class NewModel (model):


      def Execute (COMMON, I, Y, X, V, A, PAR, NEW, OLD, WRK):
            if COMMON.NEWINT == 1:
                  ERR = 0

                  if PAR [1] <0.:
                        ERR = 1
                              if COMMON.SYSPRN <0.:
                                          print "Error =", 1003

                        if PAR [2] <0.:
                        ERR = 1
                              if COMMON.SYSPRN <0.:
                                    print "Error =", 1003

                  if ERR == 1:
                        if COMMON.CODE <100:
                                    COMMON.CODE = 100
                                    COMMON.NAME = "NewModel"

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


            I [1] = A [1] * PAR [1]
            I [2] = A [2] * PAR [1]
            I [3] = A [3] * PAR [2]

            Y [1] = PAR [1]
            Y [2] = 0.
            Y [3] = 0.
            Y [4] = 0.
            Y [5] = PAR [1]
            Y [6] = 0.
            Y [7] = 0.
            Y [8] = 0.
            Y [9] = PAR [2]

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