PRADIS

DEVELOPMENT [PGO] TO C++

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**

• Introduction

This document is introductory document for the users of the complex Of [pradis], which decided to create their own programs of graphic means ([PGO]) in the language Of [s]++ for this complex. Besides the language Of [s]++, the complex Of [pradis] makes it possible to create BY [PGO] in the language FORTRAN. The methods of [sozdananija] [PGO] on FORTRAN are presented in the document “development [PGO] on FORTRAN”. Selection for developing the language Of [s]++ gives to developer the unquestionable advantages in comparison with the development on FORTRAN, since allows access to entire rich selection of the functions of the graphic packet Of openCASCADE, but it at the same time assigns much more high demands for the qualification of programmer and requires the presence of a much larger quantity of knowledge, especially from the work with the packet Of openCASCADE.

- **Objective model [PGO]**

All developed to C++ [PGO] are developed in the form separate classes and must inherit the class Of lVPS_GraphicModel and [implementirovat] the existing there virtual methods:

<[Konstruktor]> ();
~<[Destruktor]> ();

virtual of int Of init (const of std::vector<LVPS_Node> & of nodeList,
const Of q3Dict<QString> & of parameterList, const Of lVPS_Animator * of animator,
Handle (AIS_InteractiveContext) & of ais); - the initialization of parameters and numbers of units.

virtual of void Of calculate (LVPS_XYZNode * of nodes); - calculation according to the new layer of time, geometry, transformation.

virtual of void Of display (); - [otrisovka]
[PGO].
virtual of void Of refresh (); - renovation [PGO]. Application of transformation, the copying of geometry.

virtual of void Of reset (); - discharge [PGO]. Removal of object from the context.

virtual Of lVPS_GraphicModel * Of clone (); - the creation of the same unit [PGO].

virtual of inline Of qString Of getType () const; - type [PGO], i.e., name [PGO] into [Pradis].

virtual of inline Of qString Of getModelClass () const; - class [PGO], Mechanical of – mechanics**.**

virtual of void Of setVisibleLSK (bool); - the installation of the visibility of the local axes of object.

Since into the composition of th objective library Of lVPS.lib e supplied from The [pradis] enters the large collection of the auxiliary classes, which themselves realize all these methods except the method Of init, then new [PGO] can inherit these auxiliary classes, and then it it remains to realize only the method Of init. The following auxiliary classes enter into the composition of library:

LVPS_GM1Displacement
LVPS_GM1DisplacementExternalGeometry
LVPS_GM1Point
LVPS_GM1Point3Rotation
LVPS_GM1PointExternalGeometry
LVPS_GM1Quaternion
LVPS_GM1Rotation
LVPS_GM1RotationExternalGeometry
LVPS_GM2Displacement

LVPS_GM2Displacement1Rotation
LVPS_GM2Displacement1RotationExternalGeometry
LVPS_GM2DisplacementExternalGeometry
LVPS_GM2Point
LVPS_GM2PointExternalGeometry
LVPS_GM2PointSpringExternalGeometry
LVPS_GM2Quaternion

These classes realize different standard graphic conversions and so load from the graphic files of necessary geometry. In the general case [PGO] can realize itself all that the fact that make the classes enumerated above, but it is better to use them, if task makes it possible to make this.

3.1 methods of classes [PGO]

As it was said above, [PGO] in the general case must realize the following methods:

virtual of int Of init (const of std::vector<LVPS_Node> & of nodeList,
const Of q3Dict<QString> & of parameterList, const Of lVPS_Animator * of animator,
Handle (AIS_InteractiveContext) & of ais);

First always is carried out the method Of init; therefore in it all preparatory actions must be carried out.

virtual of void Of display ();

By the following is carried out the method Of display. In it graphic object is mapped into [vjuvere].

virtual of void Of calculate (LVPS_XYZNode * of nodes);

Further, in the process of animation is used the method Of calculate. In this method occurs the recomputation of attitude and form of shock absorber, and the object of [pererisovyvaestsja] by the method Of openCASCADE - Redisplay**.**

virtual of void Of refresh ();

The method Of refresh simply changes the attitude of object on the basis of the converted transformation.

virtual of void Of reset ();

The method Of reset moves away graphic object from [vjuvera].

virtual Of lVPS_GraphicModel * Of clone ();

The method Of clone creates the copy of this [PGO].

virtual of inline Of qString Of getType () const;

The method Of getType returns type [PGO], i.e., name [PGO] to [Pradis].

virtual of inline Of qString Of getModelClass () const;

The method Of getModelClass returns class [PGO], Mechanical of – mechanics**.**

virtual of void Of setVisibleLSK (bool);

The method Of setVisibleLSK [ustanovlivaet] the regime of the visibility of the local axes of object.

Detailed description and a example of that how they realize these methods can be read in the document “process of creation by new [PGO] to C++”.

3.2 order of the call of the methods

The methods enumerated above are caused in the process of the work of postprocessor in the following order:

- **Init**
- **Display**
- **Calculate**
- **Refresh**
- **Reset**

Remaining methods are caused in the arbitrary order.

- ## Operating principles

[PGO] are connected to the postprocessor as dynamic libraries on the technology of [plaginov]. One dynamic library can include several different [PGO]. Each [PGO] it must have in [bilioteke] the appropriate function, which returns the object, created from the class of this [PGO].

The information about each [PGO] is contained in the file of [ropozitorija] [PGO] (PGO_List.txt). [PGO] are caused from the postprocessor as needed and th parameters e necessary for creating the graphic units are transferred by them from the postprocessor.

[PGO] creates necessary graphic means using the means of the packet Of openCASCADE, and transfers this graphic means into 3d [vjuver]. Graphic means is created only one time in the method Of init, and it is reflected by the method Of display. But then it only is moved or is transformed by the method Of calculate.

- **List of the utilized commands Of openCASCADE**

Is given below the list most of frequently utilized with the creation [PGO] of functions OpenCASCADE ([OSS]). It must be noted, that the possibilities [OSS] from the work with the three-dimensional drawing are completely great and this list cannot pretend to the completeness, but it serves only as benefit for the novices to use functions [OSS]. For further study it is proposed to study the documentation, applied in [distributive] [OSS].

5.1 commands of the creation of the topology

The order of the creation of three-dimensional means is the following:

- First is created the geometry of object with the aid of the geometric library **gp**;
- Then on the basis of geometry is created the topology of object with the aid of the libraries **Of bRepBuilderAPI** and **BRepPrimAPI**;
- Then is created the topological object **Of aIS_Shape**, which [otrisovyvaetsja] in [vjuvere];
- Then to this the object **Of aIS_Shape** is applied the transformation **of gp_Trsf**.

For creating the graphic units the following classes are used:

gp_Ax2 of – compiling axis with the direction;
gp_Pnt of – compiling three-dimensional point;
gp_Circ of – compiling circle;
gp_Lin - compiling straight line;
gp_Dir of – the creation of direction;
gp_Elips of – the creation of [elipsa];
gp_Hypr of – the creation of hyperbola;
gp_Parab of – compiling parabola;
gp_Pln of – the creation of plan;
gp_Vec of – compiling vector;

In [OSS] there is a large collection of classes and methods of creating the topology of the three-dimensional objects:

BRepBuilderAPI_MakeEdge of – compiling different types of curves;
BRepBuilderAPI_MakeWire of – compiling the broken lines, which consist of different curves;
BRepBuilderAPI_MakeFace of – the creation of surfaces;
BRepPrimAPI_MakeCone of – the creation of cone;
BRepPrimAPI_MakeSphere of – the creation of sphere;
BRepPrimAPI_MakePrism of – the creation of prism;
BRepPrimAPI_MakeCylinder of – the creation of cylinder;
GeomFill_Pipe - extraction of outline along the normal or on the curve;
TopoDS_Compound of – the association of different topological objects into one component.

All created topological [primityvy] are written in the united structure of data described by class **TopoDS_Shape**.

Examples of the use of the classes enumerated above:

Compiling the surface in the form of the circle

```
gp_Pnt A (0, 0, 0);
gp_Pnt B (0, 0, 1);
double of radius = 10;
gp_Vec of vec (A, B);
gp_Dir of dir (vec);
gp_Ax2 of ax2 (A, dir);
gp_Circ of circ (ax2, radius);
BRepBuilderAPI_MakeEdge of edge (circ);
TopoDS_Wire Of wire = Of bRepBuilderAPI_MakeWire (edge.Edge ());
TopoDS_Shape s = Of bRepBuilderAPI_MakeFace (Wire);
```

Creation of the sphere:

```
gp_Pnt A (10, 0, shch);
double of diameter = shch;
BRepPrimAPI_MakeSphere sf (A, diameter/2.);
TopoDS_Shape s = of sf.Shape ();
```

Creation of the cylinder:

```
gp_Pnt A (0,0,0);
gp_Pnt B (0,0,10);
double of diameter = ";
double of length=sqrt (pow (B.X () - A.X (), 2) +pow (B.Y () - A.Y (), 2) +pow (B.Z () - A.Z
(), 2));
gp_Vec of vec (A, B);
gp_Dir of dir (vec);
gp_Ax2 of ax2 (A, dir);
BRepPrimAPI_MakeCylinder of cyl (ax2, diameter/of 2., length);
TopoDS_Shape s = of cyl.Shape ();
```

Compiling the straight line:

```
gp_Pnt A (10, 15, 20);
gp_Pnt B (50, 100, 130);
BRepBuilderAPI_MakeEdge of edge (A, B);
TopoDS_Shape s = of edge.Edge ();
```

Creation of the arc:

```
gp_Pnt A (0,0,0);
gp_Pnt B (0,0,10);
double Of ang1 = 0;
double Of ang2 = 0.5;
double of diameter = 10;
gp_Vec of vec (A, B);
gp_Dir of dir (vec);
gp_Ax2 of ax2 (A, dir);
gp_Circ of circ (ax2, diameter/2.);
BRepBuilderAPI_MakeEdge of edge (circ, Ang1, Ang2);
TopoDS_Shape s = of edge.Edge ();
```

5.2 commands of conversion in the space

The graphic objects, created with the commands, enumerated in the point the first can be moved in the space and transformed. For this is used the class **of gp_Trsf**. This class has such methods as:

SetRotation of – the task of the rotation:
SetTranslation of – the task of displacement;
SetScaleFactor of – the task of the scale factor:

These methods make it possible to produce different displacements and transformations above the three-dimensional graphic object. Furthermore, the most direct possible task of the matrix of transformation.

Example of the use of a class of gp_Trsf:

```
gp_Trsf of aTrsf;

gp_Vec Of vector (gp_Pnt (0,0,0), of gp_Pnt (0,0,5));
gp_Vec v2 (B.X () - A.X (), B.Y () - A.Y (), B.Z () - A.Z ());
gp_Pnt Of point (0,0,0);

double of phi = of acos (Vector * v2/Vector.Magnitude ()/of v2.Magnitude ());
if (fabs (phi) of >=1e-8 & &!Vector.IsParallel (v2, gp::Resolution ()))
{
     gp_Vec of vec = Of vector^v2;
     gp_Ax1 of ax1 (Point, vec);
     aTrsf.SetRotation (ax1, phi);
}
gp_Trsf of trsf;

gp_Vec n1 (A.X () - Point.X (), A.Y () - Point.Y (), A.Z () - Point.Z ());
trsf.SetTranslation (n1);
aTrsf=trsf * of aTrsf;
aTrsf.SetScaleFactor (shch.);
```

5.3 creation of the unit Of aIS_Shape

For mapping of graphic objects in 3d [vjuvere] is used the special object **Of aIS_Shape**, [kotryj] is used together with another class - **AIS_InteractiveContext**, being appeared mechanism of mapping graphic objects. For each final graphic object is created the separate object **Of aIS_Shape,** [kotryj] then will be brought in into the object **Of aIS_InteractiveContext,** which exists one for all graphic objects and it is the medium of their existence.

The class **Of aIS_Shape** is created on the basis of the object **Of topoDS_Shape** and it has the following useful methods:

SetMaterial of – the task of material;

SetColor of – the task of color;
SetTransparen[s]y of – the task to transparency;
Redisplay of – the copying of object;
SetDisplayMode of – the task of the regime of drawing;

The class **Of aIS_InteractiveContext** is created one for all the application and in [PGO] it to be created and to change must and comes in the form the input parameter. The class **Of aIS_InteractiveContext** has the following useful methods:

SetDisplayMode of – the task of the regime of the drawing of graphic object;
Display of – mapping graphic object;
Redisplay of – the copying of the already represented graphic object;
ResetLocation of – a change in the position of object;
SetLocation of – the displacement of graphic object with the use of a object of gp_Trsf;
Remove of – the removal of graphic object;
SetMaterial of – the task of material;
SetColor of – the task of color;
SetTransparen[s]y of – the task to transparency;

Example of the joint use of classes Of aIS_Shape and AIS_InteractiveContext:

Creation of the unit Of aIS_Shape and the task of the regime of its mapping.

```
Handle (AIS_Shape) of myAISShape = of new Of aIS_Shape (Shape);
myAISShape->SetMaterial (Graphic of 3d_NOM_PLASTIC);
myAISContext->SetDisplayMode (myAISShape, 1, Standard_False);

.....
```

Mapping the object Of aIS_Shape and the task of its attitude.

```
myAISContext->Display (myAISShape, 1,1, Standard_False, Standard_False);
myAISContext->SetLocation (myAISShape, aTrsf);

.....
```

Copying of the object Of aIS_Shape on the basis of new topology.

```
Handle (AIS_Shape)::DownCast (myAISShape) - >Set (Shape);
myAISContext->Redisplay (myAISShape, Standard_False);

.....
```

[Izmenennie] of the attitude of the object Of aIS_Shape.

```
myAISContext->ResetLocation (myAISShape);
myAISContext->SetLocation (myAISShape, aTrsf);

.....
```

Removal of the object Of aIS_Shape.

```
myAISContext->Remove (myAISShape);
```

5.4 methods Of solverContext

For the access to the input parameters [PGO] are used the methods of the object **Of solverContext** and some other classes, which work together with the object of the class **Of solverContext.** The object **Of solverContext** is created one for all the application and in [PGO] it to be created and to change must and comes in the form the input parameter. The detailed description of work from **SolverContext** see in the document "**work with DAT by file**".

All static input parameters [PGO] are taken from **SolverContext** and come in [PGO] as the parameter in the method **Of init** in the form of list with the keys **Of q3Dict<QString> & of parameterList**. Each form of the parameters has the appropriate key name to which it is added its ordinal number. There are following key names:

Par of – the parameters of the model, which includes this [PGO];
PARIMD of – the parameters [PGO];
PARLR2 of – the parameters of layer;

Example of obtaining the input parameters [PGO]:

```
double of par [9];
for (int of a=0;a<6;a++)
{
        QString of param=QString () of.sprintf ("Par % d", a);
        QString Of dr=* (parameterList [of param]);
        par [a] of =LVPS_Utility::ToDouble (Dr);
}

for (int of a=0;a<2;a++)
{
        QString of param=QString () of.sprintf ("PARIMD % d", a);
        QString Of dr=* (parameterList [of param]);
        par [a + '] of =LVPS_Utility::ToDouble (Dr);
}
```

Values on the units come as the input parameter of the method **Of calculate - LVPS_XYZNode * of nodes,** and the descriptions of units themselves come as the parameter of the method **Of init - std::vector<LVPS_Node> & of nodeList.** This vector [sodeorzhit] the objects of the class Of lVPS_Node, which contains ID of the unit, which is used in this [PGO]. On this ID it is possible to obtain the value of concrete unit from the vector of nodes in the method Of calculate.

Example of obtaining the values of the units:

```
int LVPS_AMORT::Init (const of std::vector<LVPS_Node> & of nodeList,
    const Of q3Dict<QString> & of parameterList, const Of lVPS_Animator * of animator,
    Handle (AIS_InteractiveContext) & of ais)
    {
        LVPS_Node Of nodeX1= of nodeList [0];
        LVPS_Node Of nodeY1= of nodeList [1];
```

```
        LVPS_Node Of nodeZ1= of nodeList [2];
        LVPS_Node Of nodeX2= of nodeList [e];
        LVPS_Node Of nodeY2= of nodeList [4];
        LVPS_Node Of nodeZ2= of nodeList [shch];
    .........

void LVPS_AMORT::Calculate (LVPS_XYZNode * of nodes)
{
    double of leng=sqrt (pow (nodes [Of nodeX2.ID] of.S+B.X () - nodes [Of nodeX1.ID] of.S-
    A.X (), 2) +pow (nodes [Of nodeY2.ID] of.S+B.Y () - nodes [Of nodeY1.ID] of.S-A.Y (), 2)
    +pow (nodes [Of nodeZ2.ID] of.S+B.Z () - nodes [Of nodeZ1.ID] of.S-A.Z (), 2));

    .........
```

All additional parameters are taken from the appropriate contexts, which work on the basis of the object **Of solverContext.**

Example of obtaining the values of working massif from ModelContext:

For obtaining the values of massifs for the concrete model it is necessary to at first establish ModelContext to the necessary model according to its number by using a method Of setModelNumber.

```
ModelContext MC (SolverCont);
MC.SetModelNumber (2);
double Of w14 = Of mC.GetWRK () [14];
double Of w15 = Of mC.GetWRK () [15];
double W8 = MC.GetWRK () [8];
```

Example of obtaining the values of the old and new state vector Of modelContext:

```
ModelContext MC (SolverCont);
MC.SetModelNumber (2);
double NEW1 = Of mC.GetNew () [1];
double OLD3 = Of mC.GetOld () [e];
```

- **System environment for [razarabotki] [PGO] to C++**

For the development [PGO] to [S]++ it is necessary, that in the system were established following software:

OpenCASCADE 5.2 or is above.
Qt 4
PRADIS Of postprocessor
MS Of visual Of studio 6.0

So in the project of [razarabatyvaemoj] [PGO] must be included the libraries **Of aReader.lib** and **LVPS.lib**, entering the complete set PRADIS. Furthermore, must be used the following of include files, so entering the complete set PRADIS:

- For the work with the library Of aReader it is necessary to include the directories Of aReader;
- For the work with the library LVPS it is necessary to include directories LVPS.

The project of th library e developed BY DLL must include the following tuning:

For the compilation: /nologo/Of mTd/W3/Gm/GX/ZI/Od/of the I "**the c:\dinama\include\LVPS**"/of the I "**the c:\dinama\include\AReader**"/of the I "i (QTDIR) \ include"/the I "i (QTDIR)/include/Of qtGui"/I "i (QTDIR)/include/Qt3Support"/I "Qwt/of include"/I "i (QTDIR)/include/QtXml"/the I "i (QTDIR)/include/Of qtOpenGL"/I "i (QTDIR)/include/QtCore"/the I "i (QTDIR)/include"/I "i (QTDIR)/include/ActiveQt"/I "tmp \ of moc \ of release_shared"/the I". " /I "i (QTDIR) \ mkspecs \ of win32-msvc"/the I "i (CASROOT) \ inc"/D" of _DEBUG "/D "OF WIN32"/D" of _WINDOWS "/D "of _MBCS"/D "OF WNT"/D "OF CSFDB"/D "OF QT_DLL"/D "OF QT3_SUPPORT"/FR " tmp \ of obj \ of release_shared/"/Fp " win32 \ of objd/IESample.pch "/YX/Fo " win32 \ of objd/"/Fd " win32 \ of objd/"/FD/GZ/c

For the assembling: "c:\dinama\lib\LVPS.lib" "c:\dinama\lib\AReader.lib" "i (QTDIR) \ lib \ of qtmain.lib" "i (QTDIR) \ lib \ Of qtCore4.lib" "i (QTDIR) \ lib \ Of qt3Support4.lib" "i (QTDIR) \ lib \ Of qtOpenGL4.lib" "i (QTDIR) \ lib \ Of qtXml4.lib" "i (QTDIR) \ lib \ Of qt3Supportd4.lib" "i (QTDIR) \ lib \ Of qtGui4.lib" kernel32.lib of user32.lib of gdi32.lib of winspool.lib of comdlg32.lib of advapi32.lib of shell32.lib of ole32.lib of oleaut32.lib of uuid.lib of odbc32.lib of odbccp32.lib of qtmain.lib Of tKernel.lib Of tKMath.lib Of tKService.lib TKV of 3d.lib Of tKBrep.lib Of tKIGES.lib Of pTKernel.lib Of tKSTL.lib Of tKVRML.lib Of tKSTEP.lib Of tKShapeSchema.lib TKG of 3d.lib TKG of 2d.lib Of tKXSBase.lib Of tKPShape.lib Of tKShHealing.lib Of tKTopAlgo.lib Of tKBool.lib Of tKBO.lib Of tKFillet.lib Of tKOffset.lib Of tKPrim.lib Of tKGeomBase.lib Of tKGeomAlgo.lib Of tKMeshVS.lib Of tKFeat.lib Of tKCAF.lib/nologo/dll/of incremental:yes/of pdb:"Debug/Of pGO_NAME.pdb"/debug/of machine:I386/of out:"bin \ Of pGO_NAME.dll"/implib:"Debug/Of pGO_NAME.lib"/pdbtype:sept/of libpath:"i (QTDIR) \ lib"/libpath:"i (CASROOT) \ win32 \ of lib"

The name [PGO] **PGO_ NAME** must be respectively changed to the necessary.

In the way **c:\dinama** must be respectively written the disk, on which is established [Pradis].

Project must have a access to the following libraries PRADIS:

LVPS.lib
AReader.lib

Project must have a access to the following libraries Qt:

LVPS.lib
AReader.lib
qtmain.lib
QtCore4.lib
Qt3Support4.lib
Qt3Supportd4.lib
QtGui4.lib
qtmain.lib

Project must have a access to the following libraries Of openCASCADE:

TKernel.lib
TKMath.lib
TKService.lib
TKV of 3d.lib
TKBrep.lib
TKIGES.lib
PTKernel.lib
TKSTL.lib
TKVRML.lib
TKSTEP.lib
TKShapeSchema.lib
TKG of 3d.lib
TKG of 2d.lib
TKXSBase.lib
TKPShape.lib
TKShHealing.lib
TKTopAlgo.lib
TKBool.lib
TKBO.lib
TKFillet.lib
TKOffset.lib
TKPrim.lib
TKGeomBase.lib
TKGeomAlgo.lib
TKMeshVS.lib
TKFeat.lib
TKCAF.lib

- ## Procedure of the addition of [plagin] [PGO] to [S]++ in PRADIS

For adding new [PGO] in PRADIS it is necessary to produce the following actions:

1. To create DLL the library, which contains new [PGO].

2. To place th library into the catalog **% OF DINSYS % \ of dinama \ of Post** \ e created WITH DLL **plugins \ of gip.**

3. To place information about [PGO] into the file of [ropozitorija] [PGO] (**PGO_List.txt**). In detail this procedure is described in the document “development [PGO] on FORTRAN”.

4. To create the text of empty [PGO], the only title and description [PGO] according to the requirements, presented to this by the system Of [pradis], and that containing no calculations. The size of description [PGO] on FORTRAN is contained in the documents “dynamic incorporation into the solver PRADIS of the libraries of the models of elements, [PGO], [PRVP]” and “THE START OF THE PROGRAMS OF THE REALIZATION OF GRAPHIC MEANS IN THE LIBRARIES OF COMPLEX”.

   For example, the text of empty [PGO] AMORT on FORTRAN would appear as follows:

```
C IMAGE AMORT:EXT=6, PAR=3, WRK=1
C
C HELP the graphic means of shock absorber.
C HELP THE NAME:        Graphic means of shock absorber.
C
C
C HELP OF DEGREE OF FREEDOM:
C HELP 1- is progressive of point A in the direction of axis OX;
C HELP is 2nd progressive of point A in the direction of axis OY;
C HELP 3- is progressive of point A in the direction of axis OZ;
C HELP 4- is progressive of point B in the direction of axis OX;
C HELP is 5th progressive of point B in the direction of axis OY;
C HELP ' is progressive of point B in the direction of axis OZ.
C
C HELP THE PARAMETERS:
C HELP 1- diameter of shock absorber;
C HELP is 2nd the ratio of compression stroke to the initial length of shock
absorber;
C HELP 3- the ratio of the motion of tension to the initial length of shock absorber.
C
C
      include “of init.inc”

      SUBROUTINE AMORT (
     , NAMEX,
     , The I,
```

```
     , X_, V_, A_,
     , INNER, EXT,
     , PARX, WRKX,
     , PAR, WRK,
     , PARLR2)

    !DEC$ ATTRIBUTES DLLEXPORT:: AMORT

        include "of common.inc"

    C the formal parameters
          CHARACTER * OF 8 NAMEX
          REAL * OF 8 I (1)
          REAL * 8 X_ ('), V_ ('), A_ (')
          REAL * OF 8 INNER (1), OF PARX (1), OF WRKX (1), OF PAR (1), OF
    WRK (1), OF PARLR2 (1)
          INTEGER * OF 4 EXTC
    C
         RETURN
         END
```

5. To add this [PGO] into the solver with the aid of the utility ARM, after giving the command:

ARM + OF <[IMJA] [PGO]>
(For example: **ARM + OF AMORT)**

Work with the utility ARM is described in the document "dynamic incorporation into the solver PRADIS of the libraries of the models of elements, [PGO], [PRVP]".

6. After the fulfillment of all procedures enumerated above, new [PGO] is ready to use.

• Process of creation by new [PGO] to C++

It is assumed that the reader of this document is familiar with the bases of programming in the medium Of openCASCADE. For the more detailed acquaintance from OpenCASCADE should be studied the documentation, applied to OpenCASCADE.

For the fact that to create new [PGO] to C++, it is necessary to, first of all, organize the appropriate environment.

After this, it is necessary to create in VC 6.0 empty project of dynamic library, to connect to it all necessary libraries and include files, to include in project the files of name.cpp and name.h containing text [PGO] as this is shown below.

One DLL library can contain both one and how [PGO]. Each [PGO] is described as separate class and for each [PGO] in DLL to library must be provided the function of its call as this shown in example given below for [PGO] AMORT:

```
# ifdef Q_WS_WIN
# define MY_EXPORT of __declspec (dllexport)
# else
# define MY_EXPORT
# endif

extern "C" OF MY_EXPORT Of lVPS_GraphicModel * OF AMORT ()
{
        LVPS_GraphicModel * of model = of new LVPS_AMORT ();
        return of model;
}
```

Further we will examine a example of development by concrete actually utilized [PGO] with the name AMORT. Let us first give the complete texts of its code, which is contained in two files Of lVPS_AMORT.cpp and LVPS_AMORT.h:

LVPS_AMORT.cpp

```
# include of <LVPS_AMORT.h>
# include of <LVPS_XYZNode.hxx>
# include of <LVPS_Utility.hxx>
# include of <Geom_TrimmedCurve.hxx>
# include of <TopoDS_Edge.hxx>
# include of <BRepBuilderAPI_MakeWire.hxx>
# include of <GC_MakeArcOfCircle.hxx>
# include of <BRepBuilderAPI_MakeEdge.hxx>
# include of <TopoDS_Wire.hxx>
# include of <gp_Circ.hxx>
# include of <gp_Pln.hxx>
# include of <TopoDS_Face.hxx>
# include of <BRepBuilderAPI_MakeFace.hxx>
# include of <BRepOffsetAPI_MakePipe.hxx>
```

```
# include of <Geom_CartesianPoint.hxx>
# include of <AIS_Point.hxx>
# include of <BRepPrimAPI_MakeCylinder.hxx>
# include of <TopoDS_Compound.hxx>

# ifdef Q_WS_WIN
# define MY_EXPORT of __declspec (dllexport)
# else
# define MY_EXPORT
# endif

extern “C” OF MY_EXPORT Of lVPS_GraphicModel * OF AMORT ()
{
        LVPS_GraphicModel * of model = of new LVPS_AMORT ();
        return of model;
}


LVPS_AMORT::LVPS_AMORT ()
{
        myAISShape.Nullify ();
};

LVPS_AMORT::~LVPS_AMORT ()
{

};

int LVPS_AMORT::Init (const of std::vector<LVPS_Node> & of nodeList,
        const Of q3Dict<QString> & of parameterList, const Of lVPS_Animator * of
animator,
        Handle (AIS_InteractiveContext) & of ais)
{
        NodeX1= of nodeList [0];
        NodeY1= of nodeList [1];
        NodeZ1= of nodeList [2];
        NodeX2= of nodeList [e];
        NodeY2= of nodeList [4];
        NodeZ2= of nodeList [shch];
        myAISContext = of ais;

        double of par [9];
        for (int of a=0;a<6;a++)
        {
                QString of param=QString () of.sprintf (“Par % d”, a);
                QString Of dr= * (parameterList [of param]);
                par [a] of =LVPS_Utility::ToDouble (Dr);
        }

        for (int of a=0;a<2;a++)
        {
                QString of param=QString () of.sprintf (“PARIMD % d”, a);
```

```
                QString Of dr= * (parameterList [of param]);
                par [a + '] of =LVPS_Utility::ToDouble (Dr);
        }

        A=gp_Pnt (par [0], par [1], par [2]);
        B=gp_Pnt (par [e], par [4], par [shch]);
        diameter=par ['];
        length=sqrt (pow (B.X () - A.X (), 2) +pow (B.Y () - A.Y (), 2) +pow (B.Z () -
A.Z (), 2));
        minlength = of length * of par [''];
        maxlength = of length + of length * of par [8];

        Shape = Of amort (diameter, length, minlength, maxlength);
        myAISShape = of new Of aIS_Shape (Shape);
        myAISShape->SetMaterial (Graphic of 3d_NOM_PLASTIC);
        SetColor (myAISShape);

        myAISContext->SetDisplayMode (myAISShape, 1, Standard_False);

        return 0;
};

void LVPS_AMORT::Display ()
{
        myAISContext->Display (myAISShape, 1,1, Standard_False, Standard_False);
        myAISContext->SetLocation (myAISShape, aTrsf);
};

void LVPS_AMORT::Calculate (LVPS_XYZNode * of nodes)
{
        double of leng=sqrt (pow (nodes [Of nodeX2.ID] of.S+B.X () - nodes [Of
nodeX1.ID] of.S-A.X (), 2) +pow (nodes [Of nodeY2.ID] of.S+B.Y () - nodes [Of nodeY1.ID]
of.S-A.Y (), 2) +pow (nodes [Of nodeZ2.ID] of.S+B.Z () - nodes [Of nodeZ1.ID] of.S-A.Z (),
2));
        Shape = Of amort (diameter, leng, minlength, maxlength, nodes);

                if (myAISShape.IsNull ())
                {
                        myAISShape = of new Of aIS_Shape (Shape);
                } else
                {
                        Handle (AIS_Shape)::DownCast (myAISShape) - >Set (Shape);
                        myAISContext->Redisplay (myAISShape, Standard_False);
                }
};


void LVPS_AMORT::Refresh ()
{
        myAISContext->ResetLocation (myAISShape);
        myAISContext->SetLocation (myAISShape, aTrsf);
};
```

```
void LVPS_AMORT::Reset ()
{
        myAISContext->Remove (myAISShape);
};

LVPS_GraphicModel * OF LVPS_AMORT::Clone ()
{
        return (LVPS_GraphicModel *) (new LVPS_AMORT ());
};

TopoDS_Shape LVPS_AMORT::Amort (const Of standard_Real of diameter1,
                const Of standard_Real of length, const Of standard_Real of minlength,
const Of standard_Real of maxlength, LVPS_XYZNode * of nodes)
{

        TopoDS_Compound of comp;
        BRep_Builder of builder;
        builder.MakeCompound (comp);

        BRepPrimAPI_MakeCylinder of cyl1 (diameter 1/2 ; minlength);
        builder.Add (comp, cyl1.Shape ());

        BRepPrimAPI_MakeCylinder of cyl2 (diameter1/of 10., length);

        builder.Add (comp, cyl2.Shape ());

        S = of comp;

        if (nodes==NULL)
        {
                gp_Vec Of vector (gp_Pnt (0,0,0), of gp_Pnt (0,0,5));
                gp_Vec v2 (B.X () - A.X (), B.Y () - A.Y (), B.Z () - A.Z ());
                gp_Pnt Of point (0,0,0);

                double of phi = of acos (Vector * v2/Vector.Magnitude ()/of
v2.Magnitude ());
                if (fabs (phi) of >=1e-8 & &!Vector.IsParallel (v2, gp::Resolution ()))
                {
                        gp_Vec of vec = Of vector^v2;
                        gp_Ax1 of ax1 (Point, vec);
                        aTrsf.SetRotation (ax1, phi);
                }
                gp_Trsf of trsf;

                gp_Vec n1 (A.X () - Point.X (), A.Y () - Point.Y (), A.Z () - Point.Z ());
                trsf.SetTranslation (n1);
                aTrsf=trsf * of aTrsf;
        }
        else
        {
                gp_Vec Of vector (gp_Pnt (0,0,0), of gp_Pnt (0,0,5));
```

```
                        gp_Vec v2 (nodes [Of nodeX2.ID] of.S+B.X () - nodes [Of nodeX1.ID]
of.S-A.X (),
                                nodes [Of nodeY2.ID] of.S+B.Y () - nodes [Of nodeY1.ID] of.S-
A.Y (),
                                nodes [Of nodeZ2.ID] of.S+B.Z () - nodes [Of nodeZ1.ID] of.S-
A.Z ());
                        gp_Pnt Of point (0,0,0);

                        double of phi = of acos (Vector * v2/Vector.Magnitude ()/of
v2.Magnitude ());
                        if (fabs (phi) of >=1e-8 & &!Vector.IsParallel (v2, gp::Resolution ()))
                        {
                                gp_Vec of vec = Of vector^v2;
                                gp_Ax1 of ax1 (Point, vec);
                                aTrsf.SetRotation (ax1, phi);
                        }
                        gp_Trsf of trsf;

                        gp_Vec n1 (nodes [Of nodeX1.ID] of.S+A.X () - Point.X (),
                                nodes [Of nodeY1.ID] of.S+A.Y () - Point.Y (), nodes [Of
nodeZ1.ID] of.S+A.Z () - Point.Z ());
                        trsf.SetTranslation (n1);
                        aTrsf=trsf * of aTrsf;
                }

                return S;
        }
```

LVPS_AMORT.h

```
# ifndef LVPS_AMORT_H
# define LVPS_AMORT_H

# include “Of lVPS_GraphicModel.hxx”
# include “Of lVPS_Node.hxx”
# include of <TopoDS_Face.hxx>
# include of <TopoDS_Wire.hxx>
# include “Of lVPS.h”



class LVPS_AMORT:public Of lVPS_GraphicModel
{
public:
  LVPS_AMORT ();
  ~LVPS_AMORT ();

  virtual of inline Of qString Of getType () const
  {
          return “OF AMORT”;
  };
```

```
  virtual of inline Of qString Of getModelClass () const
  {
        return “Of mechanical”;
  };
  virtual of int Of init (
  const of std::vector<LVPS_Node> & of nodeList,
  const Of q3Dict<QString> & of parameterList, const Of lVPS_Animator * of animator,
  Handle (AIS_InteractiveContext) & of ais);

  virtual of void Of calculate (LVPS_XYZNode * of nodes);
  virtual of void Of display ();
  virtual of void Of refresh ();
  virtual of void Of reset ();
  virtual Of lVPS_GraphicModel * Of clone ();
  virtual of void Of setVisibleLSK (bool) {};
  TopoDS_Shape Of amort (const Of standard_Real of diameter1,
                  const Of standard_Real of length, const Of standard_Real of minlength,
           const Of standard_Real of maxlength, LVPS_XYZNode * of nodes=NULL);

protected:
        TopoDS_Shape Of shape;
        Handle (AIS_Shape) of myAISShape, myshape;
        Handle (AIS_InteractiveContext) of myAISContext;
        LVPS_Node Of nodeX1, NodeY1, NodeZ1, NodeX2, NodeY2, NodeZ2;
        double of diameter, quantity, length, maxlength, minlength;
        gp_Trsf of aTrsf;
        gp_Pnt A, B;
        TopoDS_Face F;
        TopoDS_Wire W;
        TopoDS_Shape S;
};
# endif
```

Each [PGO] it must contain the specific collection of the realized required methods:

```
  virtual of int Of init (const of std::vector<LVPS_Node> & of nodeList,
         const Of q3Dict<QString> & of parameterList, const Of lVPS_Animator * of animator,
         Handle (AIS_InteractiveContext) & of ais);

  virtual of void Of calculate (LVPS_XYZNode * of nodes);
  virtual of void Of display ();
  virtual of void Of refresh ();
  virtual of void Of reset ();
  virtual Of lVPS_GraphicModel * Of clone ();
  virtual of inline Of qString Of getType () const;
  virtual of inline Of qString Of getModelClass () const;
```

And it must be inherited from the class Of lVPS_GraphicModel or from another more general common class, which forms part of library LVPS and which inherits in turn the class Of lVPS_GraphicModel. In that case [PGO] itself must realize only the methods Of init and Clone, since remaining methods are already realized in the inherited classes.

[PGO] AMORT, which we examine as a example, itself realizes all necessary methods, since she is inherited directly from the class Of lVPS_GraphicModel.

First always is carried out the method Of init; therefore in it all preparatory actions must be carried out. At first the variables of the units initialize from the input parameters:

```
NodeX1= of nodeList [0];
NodeY1= of nodeList [1];
NodeZ1= of nodeList [2];
NodeX2= of nodeList [e];
NodeY2= of nodeList [4];
NodeZ2= of nodeList [shch];
```

Then is filled up field for AIS_InteractiveContext:

```
myAISContext = of ais;
```

After this, [zapominaejutsja] all necessary input parameters [PGO]:

```
double of par [9];
for (int of a=0;a<6;a++)
{
        QString of param=QString () of.sprintf ("Par % d", a);
        QString Of dr= * (parameterList [of param]);
        par [a] of =LVPS_Utility::ToDouble (Dr);
}

for (int of a=0;a<2;a++)
{
        QString of param=QString () of.sprintf ("PARIMD % d", a);
        QString Of dr= * (parameterList [of param]);
        par [a + '] of =LVPS_Utility::ToDouble (Dr);
}
```

Further, from the values of the input parameters two three-dimensional points and variable, which contains the diameter of the shock absorber are created:

```
A=gp_Pnt (par [0], par [1], par [2]);
B=gp_Pnt (par [e], par [4], par [shch]);
diameter=par ['];
```

It is calculated long shock absorber and they are long the components of its parts:

```
length=sqrt (pow (B.X () - A.X (), 2) +pow (B.Y () - A.Y (), 2) +pow (B.Z () - A.Z (),
2));
minlength = of length * of par ["];
maxlength = of length + of length * of par [8];
```

Is created the object Of topoDS_Shape containing the geometry of the shock absorber (work of the function Of amort we let us examine more lately):

```
Shape = Of amort (diameter, length, minlength, maxlength);
```

Is created AIS_InteracliveObject (AIS_Shape):

myAISShape = of new Of aIS_Shape (Shape);

The material of the plastics is assigned:

myAISShape->SetMaterial (Graphic of 3d_NOM_PLASTIC);

Is assigned color from the operator of the task Of layer:

SetColor (myAISShape);

To object is assigned the regime of the mapping:

myAISContext->SetDisplayMode (myAISShape, 1, Standard_False);

By the following is carried out the method Of display. In it graphic object is mapped into [vjuvere]:

myAISContext->Display (myAISShape, 1,1, Standard_False, Standard_False);

and is established its [polozheneie] in the space:

myAISContext->SetLocation (myAISShape, aTrsf);

Further, in the process of animation is used the method Of calculate. In this method occurs the recomputation of attitude and form of shock absorber, and the object of [pererisovyvaestsja] by the method **Of redisplay (myAISShape, Standard_False).**

double of leng=sqrt (pow (nodes [Of nodeX2.ID] of.S+B.X () - nodes [Of nodeX1.ID] of.S-A.X (), 2) +pow (nodes [Of nodeY2.ID] of.S+B.Y () - nodes [Of nodeY1.ID] of.S-A.Y (), 2) +pow (nodes [Of nodeZ2.ID] of.S+B.Z () - nodes [Of nodeZ1.ID] of.S-A.Z (), 2));
Shape = Of amort (diameter, leng, minlength, maxlength, nodes);

```
if (myAISShape.IsNull ())
{
        myAISShape = of new Of aIS_Shape (Shape);
} else
{
        Handle (AIS_Shape)::DownCast (myAISShape) - >Set (Shape);
        myAISContext->Redisplay (myAISShape, Standard_False);
}
```

It must be noted, that in all methods of reflection, the regime **of updateviewer** must be established into the value **Of standard_False,** which means that [vjuver] in this case does not draw again. Copying will be made later in the postprocessor itself one time for all [PGO], which substantially accelerates the process of animation.

The method Of refresh simply changes the attitude of object on the basis of the converted transformation **of aTrsf.**

```
myAISContext->ResetLocation (myAISShape);
myAISContext->SetLocation (myAISShape, aTrsf);
```

Now let us examine the function Of amort, in which it occurs it [rasschet] the geometry of shock absorber.

In the beginning is created the object **Of topoDS_Compound,** which makes it possible to create TopoDS_Shape with the complex construction, which consists of any quantity different, not connected with each other, graphic objects. So is created the object **Of bRep_Builder,** which makes it possible to work with the object **Of topoDS_Compound.**

```
TopoDS_Compound of comp;
BRep_Builder of builder;
builder.MakeCompound (comp);
```

We further create the first cylinder, of which will consist the means of our shock absorber:

```
BRepPrimAPI_MakeCylinder of cyl1 (diameter 1/2 ; minlength);
```

And we place it into **TopoDS_Compound:**

```
builder.Add (comp, cyl1.Shape ());
```

The same we make also with the second cylinder:

```
BRepPrimAPI_MakeCylinder of cyl2 (diameter1/of 10., length);
builder.Add (comp, cyl2.Shape ());

S = of comp;
```

Further, depending on whether is sketched object at the very beginning **(nodes==NULL)**, or it is sketched in the process of animation, it is produced the calculation of its attitude:

```
if (nodes==NULL)
{
        gp_Vec Of vector (gp_Pnt (0,0,0), of gp_Pnt (0,0,5));
        gp_Vec v2 (B.X () - A.X (), B.Y () - A.Y (), B.Z () - A.Z ());
        gp_Pnt Of point (0,0,0);

        double of phi = of acos (Vector * v2/Vector.Magnitude ()/of
v2.Magnitude ());
        if (fabs (phi) of >=1e-8 & &!Vector.IsParallel (v2, gp::Resolution ()))
        {
                gp_Vec of vec = Of vector^v2;
                gp_Ax1 of ax1 (Point, vec);
                aTrsf.SetRotation (ax1, phi);
        }
        gp_Trsf of trsf;
```

```
                gp_Vec n1 (A.X () - Point.X (), A.Y () - Point.Y (), A.Z () - Point.Z ());
                trsf.SetTranslation (n1);
                aTrsf=trsf * of aTrsf;
        }
        else
        {
                gp_Vec Of vector (gp_Pnt (0,0,0), of gp_Pnt (0,0,5));
                gp_Vec v2 (nodes [Of nodeX2.ID] of.S+B.X () - nodes [Of nodeX1.ID]
of.S-A.X (),
                        nodes [Of nodeY2.ID] of.S+B.Y () - nodes [Of nodeY1.ID] of.S-
A.Y (),
                        nodes [Of nodeZ2.ID] of.S+B.Z () - nodes [Of nodeZ1.ID] of.S-
A.Z ());
                gp_Pnt Of point (0,0,0);

                double of phi = of acos (Vector * v2/Vector.Magnitude ()/of
v2.Magnitude ());
                if (fabs (phi) of >=1e-8 & &!Vector.IsParallel (v2, gp::Resolution ()))
                {
                        gp_Vec of vec = Of vector^v2;
                        gp_Ax1 of ax1 (Point, vec);
                        aTrsf.SetRotation (ax1, phi);
                }
                gp_Trsf of trsf;

                gp_Vec n1 (nodes [Of nodeX1.ID] of.S+A.X () - Point.X (),
                        nodes [Of nodeY1.ID] of.S+A.Y () - Point.Y (), nodes [Of
                        nodeZ1.ID] of.S+A.Z () - Point.Z ());
                trsf.SetTranslation (n1);
                aTrsf=trsf * of aTrsf;
        }
```

the [rasschitanaja] form of object returns in the object **S**,

```
return S;
```

But the calculated transformation [sokhranajaetsja] in the object **of aTrsf,** which then is used with the copying of shock absorber in the method Of refresh.

The method Of getType returns the line, which contains name [PGO]:

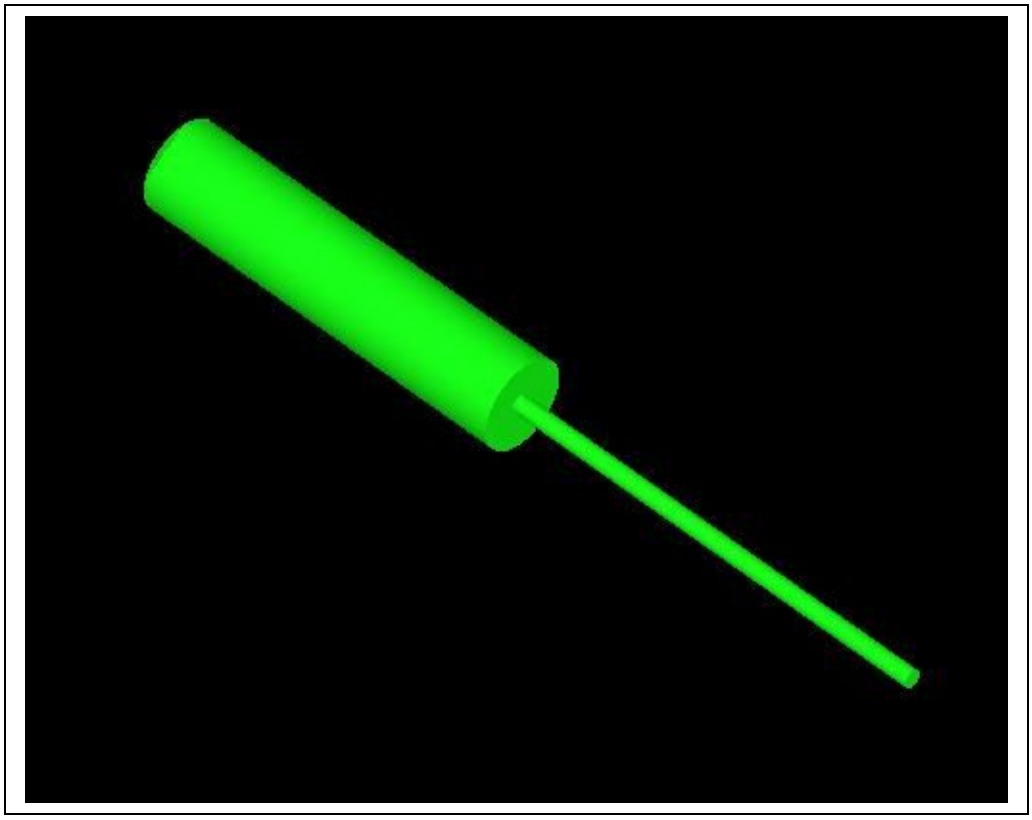
```
virtual of inline Of qString Of getType () const
{
        return "OF AMORT";
};
```

The method Of getModelClass returns the line, which contains the name of class. For mechanical [PGO] it always consists of the word **"Of mechanical"**:

```
virtual of inline Of qString Of getModelClass () const
```

```
{
        return “Of mechanical”;
};
```

The obtained means of shock absorber takes the following form:



Received as a result of the compilation of this [PGO] DLL library must be placed into the catalog **% OF DINSYS % \ of dinama \ of Post,** and the information about it must be [zanesny] into the file of [ropozitorija] [PGO] (PGO_List.txt).