```
# -------------------------------------------------------
# This is a scrtipt to check the geom.CreateBSSE command
# ------------------------------------------------------

*xyz 0 1
o:      -1.69296787   -0.05579265    0.00556629
h:      -2.01296504    0.84704339   -0.01586469
h:      -0.73325076    0.04238910    0.00084302
o        1.23009925    0.02698440   -0.00375550
h        1.60672086   -0.41139567    0.76236888
h        1.60236356   -0.44922858   -0.74915800
*

%Compound
  Geometry monomerA;
  variable myFilename     = "BSSE";
  Variable method         = "BP86";

  # --------------------------------------
  # Calculation for Fragment A
  # --------------------------------------
  New_Step
    !&{method}
  Step_End

  # -------------------------------------
  # Read the geometry of Fragment A
  # -------------------------------------
  monomerA.Read();

  # -------------------------------------
  # Create the missing xyz files
  # -------------------------------------
  monomerA.CreateBSSE(filename=myFilename);

End
```

NOTE The files will contain XYZ geometries in **BOHRS**.

G.FollowNormalMode

FollowNormalMode command acts on a geometry object (see see *Geometry*). It will displace the loaded geometry following a chosen normal mode of vibtation

Syntax:
myGeom.FollowNormalMode(vibrationSN=myVibration, [ScalingFactor=myScalingFactor]);

Where:

- *myGeom* is a geometry object that already contains a geometry
- *vibrationSN* is the serial number of the vibration. **NOTE** Please remember that counting starts with 1.
- *scalingFactor* is the scaling of the normal mode of vibration. This argument is optional.

Example:

```
# ----------------------------------------------------
# This is a script to check the followNormalMode function
# -----------------------------------------------------
*xyz 0 1
  O      -1.69296787   -0.05579265    0.00556629
  H      -2.01296504    0.84704339   -0.01586469
```

(continues on next page)

(continued from previous page)

```
  H      -0.73325076    0.04238910    0.00084302
*

%Compound
  Geometry myGeom;
  Variable CC, normalModes;
  Variable res = -1;
  New_Step
    !BP86 Freq
  Step_End
  myGeom.Read();
  myGeom.FollowNormalMode(vibrationSN=7, scalingFactor=0.8);
  CC = myGeom.GetCartesians();
  CC.PrintMatrix();
End
```

G.DisplaceAtom

Function *DisplaceAtom* acts on geometry objects and moves the cartesian coordintates of a given atom of the current geometry. The displacement is in Bohrs.

Syntax:

res = geom.DisplaceAtom(atomID, dx, dy, dz)

Where:

geom A geometry object previously loaded.

atomID The index of the atom we want to move. Please note that counting starts with zero.

dx The displacement in the x axis in Bohrs.

dy The displacement in the y axis in Bohrs.

dz The displacement in the z axis in Bohrs.

Example:

```
# ------------------------------------
# This is to test Geometry function
#         DisplaceAtom
# ------------------------------------
%Compound
  Variable dist=0.1;
  Geometry myGeom;
  NewStep
    !BP86
    *xyz 0 1
       O  0.000  0.000   0.394
       H -0.755  0.000  -0.197
       H  0.755  0.000  -0.197
    *
  StepEnd
  myGeom.Read();
  myGeom.DisplaceAtom(0, 0.0, 0.0, -dist);  #Move the z coordinate of Oxygen
  myGeom.WriteXYZFile(filename="displaced.xyz");

End
```

G.GetAngle

Function *GetAngle* acts on geometry objects and returns the angle between three atoms in **Degrees**.

Syntax:

res = geom.GetAngle(atomA, atomB, atomC)

Where:

res The angle between atoms atomA, atomB and atomC.

geom A geometry object previously loaded.

atomA The index of atomA in the geometry.

atomB The index of atomB in the geometry.

atomC The index of atomC in the geometry.

NOTE Please not that the ordering of atoms is important

Example:

```
# ------------------------------------
# This is to test Geometry function
#          GetAngle
# ------------------------------------
%Compound
  Variable dist;
  Geometry myGeom;
  New_Step
    !BP86
    *xyz 0 1
       O  0.00000000  0.00000000   0.39393904
       H -0.75503878  0.00000000  -0.19696952
       H  0.75503878  0.00000000  -0.19696952
    *
  Step_End
  myGeom.Read();  #Reads teh geometry of the previous step
  print( " ---------------------------------------------------------\n");
  print( "  Compound Geometry functions test  (GetAngle) \n");
  print( "  It should print 103.9051 Degrees\n");
  print( " ---------------------------------------------------------\n");
  print( " The angle between atom %d, atom %d and atom %d  is: %.4lf Degreess\n",
           1, 0, 2, myGeom.GetAngle(1,0,2));

End
```

G.GetAtomicNumbers

Function *GetAtomicNumbers* acts on geometry objects and returns and array with the atomic numbers of the elements in the working geometry.

Syntax: *atomNumbers = geom.GetAtomicNumbers()*

atomNumbers A variable that will be filled with the values of the atomic numbers

geom A geometry object that should already be loaded.

Example:

```
*xyz 0 1
  O      -1.69296787   -0.05579265    0.00556629
  H      -2.01296504    0.84704339   -0.01586469
  H      -0.73325076    0.04238910    0.00084302
```

(continues on next page)

(continued from previous page)

```
*

%Compound
  Geometry myGeom;
  Variable atomicNumbers;

  New_Step
    !BP86
  Step_End
  myGeom.Read();
  atomicNumbers = myGeom.GetAtomicNumbers();
  print("\nCompound \n");
  for i from 0 to atomicNumbers.GetSize()-1 Do
    print("Atom '%d': atomic number: %d\n", i, atomicNumbers[i]);
  EndFor
End
```

G.GetBondDistance

Function *GetBondDistance* acts on geometry objects and returns the distance between two atoms **in Bohrs**.

Syntax:

res = geom.GetBondDistance(atomA, atomB)

Where:

res The distance between atoms atomA and atomB.

geom A geometry object previously loaded.

atomA The index of atomA in the geometry.

atomB The index of atomB in the geometry.

NOTE indices start counting from 0

Example:

```
# -------------------------------------
# This is to test Geometry function
#    GetBondDistance
# -------------------------------------
%Compound
  Variable dist;
  Geometry myGeom;
  New_Step
    !BP86
    *xyz 0 1
      H 0.0 0.0 0.0
      H 0.0 0.0 0.8
    *
  Step_End
  myGeom.Read();  #Reads teh geometry of the previous step
  print( " ------------------------------------------------------\n");
  print( "  Compound Geometry functions test (GetBondDistance) \n");
  print( "  It should print 1.5118\n");
  print( " ------------------------------------------------------\n");
  print( " The distance between atom %d and atom %d  is: %.4lf Bohr\n",
           0, 1, myGeom.GetBondDistance(0,1));

End
```

G.GetCartesians

Function *GetCartesians* acts on geometry objects (see *Geometry*) and returns the distance xyz cartesian coordinates. Please remember that it alsways returns the cooridnates in **BOHRS**.

Syntax:

coords = geom.GetCartesians()

Where:

coords: A (*nAtoms*,3) array with the cartesian coordinates in **BOHRS**.

geom: A geometry object previously loaded.

Example:

```
# ----------------------------------------------------
# This is a script to check the GetCartesians function
# NOTE: It always return it in Bohrs!
# -----------------------------------------------------
*xyz 0 1
  O      -1.69296787   -0.05579265    0.00556629
  H      -2.01296504    0.84704339   -0.01586469
  H      -0.73325076    0.04238910    0.00084302
*

%Compound
  Geometry myGeom;
  Variable CC;
  New_Step
    !BP86
  Step_End
  myGeom.Read();
  CC = myGeom.GetCartesians();
  for i from 0 to CC.GetDim1()-1 Do
    print("%12.9lf  %12.9lf  %12.9lf\n",
    CC[i][0], CC[i][1], CC[i][2]);
  endFor
End
```

G.GetGhostAtoms

Function *GetGhostAtoms* acts on geometry objects (see *Geometry*). It returns a vector of size nAtoms where for each atom the value will be -1 if it is a ghost atom, otherwise the atomic number of the element

Syntax:

ghostAtoms = geom.GetGhostAtoms()

Where:

ghostAtoms: A (*nAtoms*,1) integer vector with values -1 or the atomic number of the atom, in case it is not a ghost atom.

geom: A geometry object previously loaded.

Example:

```
# ----------------------------------------------------
# This is a script to check the getGhostAtoms function
# -----------------------------------------------------
*xyz 0 1
o:      -1.69296787   -0.05579265    0.00556629
h:      -2.01296504    0.84704339   -0.01586469
```

(continues on next page)

(continued from previous page)

```
h:      -0.73325076    0.04238910    0.00084302
o        1.23009925    0.02698440   -0.00375550
h        1.60672086   -0.41139567    0.76236888
h        1.60236356   -0.44922858   -0.74915800
*

%Compound
  Geometry myGeom;
  Variable ghostAtoms;
  New_Step
    !BP86
  Step_End
  myGeom.Read();
  ghostAtoms = myGeom.GetGhostAtoms();
  ghostAtoms.PrintMatrix();
End
```

G.GetNumOfAtoms

GetNumOfAtoms returns an integer with the number of atoms of the working geometry.

Syntax:

res = geom.GetNumOfAtoms();

Where:

res is the resulting number of atoms

geom is the name of a geometry variable (see *Geometry*) we are using.

Example:

```
*xyz 0 1
  O      -1.69296787   -0.05579265    0.00556629
  H      -2.01296504    0.84704339   -0.01586469
  H      -0.73325076    0.04238910    0.00084302
*

%Compound
  Geometry myGeom;
  Variable numOfAtoms = 0;

  New_Step
    !BP86
  Step_End
  Alias currStep;
  myGeom.Read(currStep);
  numOfAtoms = myGeom.GetNumOfAtoms();
  print("\nCompound \n");
  print("Number of atoms: %d (it should print 3)\n", numOfAtoms);

End
```

G.MoveAtomToCenter

Function *MoveAtomToCenter* acts on geometry objects (see *Geometry*). It will adjust the cartesian coordinates so that the chosen atom will rest at (0.0 0.0 0.0).

Syntax:

geom.MoveAtomToCenter(atom serial nubmer);

Where:

geom: A geometry object previously loaded.

atom serial number: The serial number of the atom in the geometry.

Example:

```
# ----------------------------------------------------
# This is a script to check the moveAtomToCenter function
# -----------------------------------------------------
*xyz 0 1
  O      -1.69296787   -0.05579265    0.00556629
  H      -2.01296504    0.84704339   -0.01586469
  H      -0.73325076    0.04238910    0.00084302
*

%Compound
  Geometry myGeom;
  Variable CC;
  New_Step
    !BP86
  Step_End
  myGeom.Read();
  myGeom.MoveAtomToCenter(0);
  myGeom.BohrToAngs();
  CC = myGeom.GetCartesians();
  CC.PrintMatrix();
End
```

NOTE Please remember that counting **starts with 0**, meaning that the first atom is 0 and not 1!

G.Read

Function *Read* acts on geometry objects (see *Geometry*) and reads a geometry from a property file related to a previous step.

Syntax:

geom.Read([stepID=myStepID], propertySN=myPropertySN)

Where:

geom: A geometry object that will be updated.

stepID: The step from which we are going to read the geometry. If not given the previous step will be used.

propertySN: The serial number the geometry in the property file. If not given the last available geometry will be used.

Example:

```
# ----------------------------------------------------
# This is a script to check the Read function for
#   geometries
# -----------------------------------------------------
*xyz 0 1
  O      -1.69296787   -0.05579265    0.00556629
```

(continues on next page)

(continued from previous page)

```
  H      -2.01296504    0.84704339   -0.01586469
  H      -0.73325076    0.04238910    0.00084302
*

%Compound
  Geometry myGeom;
  Variable CC;
  New_Step
    !BP86 opt
  Step_End
  myGeom.Read(propertySN=2);
End
```

G.RemoveAtoms

Function *RemoveAtoms* acts on geometry objects (see *Geometry*). It accepts a list of atoms and removes them from the loaded geometry. In the end the geometry object will be updated.

Syntax:

geom.RemoveAtoms(atom1, atom2, ...);

Where:

geom: A geometry object previously loaded.

atom1, atom2, ...: The serial number of the atoms in the geometry.

Example:

```
# -----------------------------------------------------
# This is a script to check the RemoveAtoms function
# ------------------------------------------------------
*xyz 0 1
  O      -1.69296787   -0.05579265    0.00556629
  H      -2.01296504    0.84704339   -0.01586469
  H      -0.73325076    0.04238910    0.00084302
*

%Compound
  Geometry myGeom;
  Variable numOfAtoms;
  New_Step
    !BP86
  Step_End
  myGeom.Read();
  numOfAtoms = myGeom.GetNumOfAtoms();
  print("Number of atoms before: %d (It should print 3)\n", numOfAtoms);
  myGeom.RemoveAtoms(0);  #Remove the first atom
  numOfAtoms = myGeom.GetNumOfAtoms();
  print("Number of atoms after : %d (It should print 2)\n", numOfAtoms);
End
```

NOTE Please remember that counting **starts with 0**, meaning that the first atom is 0 and not 1!

G.RemoveElements

Function *RemoveElements* acts on geometry objects (see *Geometry*). It will remove from the loaded geometry all atoms with an atomic number given in the list.

Syntax:

geom.RemoveElements(atomNumber1, atomicNumber2, …);

Where:

geom: A geometry object previously loaded.

atomicNumber1, atomicNumber2, …: The atomic number of elements to be removed from the current geometry.

Example:

```
# ----------------------------------------------------
# This is a script to check the RemoveElements function
# -----------------------------------------------------
*xyz 0 1
  O      -1.69296787   -0.05579265    0.00556629
  H      -2.01296504    0.84704339   -0.01586469
  H      -0.73325076    0.04238910    0.00084302
*

%Compound
  Geometry myGeom;
  Variable numOfAtoms;
  Variable CC;
  New_Step
    !BP86
  Step_End
  myGeom.Read();
  numOfAtoms = myGeom.GetNumOfAtoms();
  CC = myGeom.GetCartesians();
  CC.PrintMatrix();
  print("Number of atoms before: %d (It should print 3)\n", numOfAtoms);
  myGeom.RemoveElements(8);  #Remove the oxygen
  numOfAtoms = myGeom.GetNumOfAtoms();
  print("Number of atoms after : %d (It should print 2)\n", numOfAtoms);
  CC = myGeom.GetCartesians();
  CC.PrintMatrix();
End
```

G.WriteXYZFile

Function *WriteXYZFile* acts on geometry objects (see *Geometry*) and writes on disc an xyz file with the coordinates of the current goemetry object. Please remember that by default it writes the coordinates in **Angstroems**.

Syntax:

res = geom.WriteXYZFile(filename=myFilename, [useBohr=True/False])

Where:

res: An integer that returns '0' if everything worked smoothly.

myFilename The name of the file that will contain the coordinates.

useBohr: A boolean that can take the value of "True" or "False". This is an optional argument and the default value is "False".

geom: A geometry object previously loaded.

NOTE: Please note that we changed the default units. Now by default we write the xyz file in Angstroems. We added the parameter UseBohr that is set by default to "False" but if it is used with the value "True" then the geometry is saved in Bohrs.

Example:

```
# -------------------------------------------------------
# This is a script to check the WriteXYZ function
# NOTE: By default it write the coordinates in Angstroems!
# -------------------------------------------------------
*xyz 0 1
  O      -1.69296787   -0.05579265    0.00556629
  H      -2.01296504    0.84704339   -0.01586469
  H      -0.73325076    0.04238910    0.00084302
*

%Compound
  Geometry myGeom;
  Variable res=-1;
  New_Step
    !BP86
  Step_End
  myGeom.Read();
  res = myGeom.WriteXYZFile(filename="myGeom.xyz",useBohr=false);
End
```

GOAT

In *Compound* we have *GOAT* objects. These objects can be treated like normal variables of type '*compGOAT*'. An important difference between normal variables and *GOAT* variables is the declaration. Instead of the normal:

```
Variable myGoat;
```

we excplicitly have to declare that this is a GOAT object. So the syntax for a goat declaration is:

Syntax:

```
GOAT myGoat;
```

Below is a list of functions that work on *Geometry* objects.

- Get_Energy (*Goat.Get_Energy*)
- Get_Num_Of_Geometries (*Goat.Get_Num_Of_Geometries*)
- ParseEensembleFile (*Goat.ParseEnsembleFile*)
- Set_Basename (*Goat.Set_Basename*)
- Print (*Goat.Print*)
- WriteXYZFile (*Goat.WriteXYZFile*)

Example:

```
# Example for the usage of GOAT objects in compound
#
# NOTE:  The file 'test.finalensemble.xyz'
#        should be availablie in the directory
%Compound
  goat myGoat;
  variable myFilename;
  myGoat.Set_Basename("test");
  myGoat.ParseEnsemblefile();
```

(continues on next page)

(continued from previous page)

```
  for myGeomID from 1 to myGoat.Get_Num_of_Geometries() Do
    write2String(myFilename, "goatGeom_%d.xyz", myGeomID);
    myGoat.WriteXYZFile(geomID=myGeomID, filename=myFilename);
    print("geomID: %d Energy: %lf\n", myGeomID, myGoat.Get_
↪Energy(geomID=myGeomID));
  EndFor
  myGoat.Print();
EndRun
```

Goat.Get_Energy

Function *Get_Energy* acts on goat objects (see *GOAT*) and returns a double with the energy of the requested structure.

Syntax:

res = myGoat.Get_Energy(geomID=myGeomID)

Where:

res: A double with the energy of the requested structure.

myGoat A GOAT object that is already loaded (meaning it has already parse an ensemble file, see (see *Goat.ParseEnsembleFile*).

myGeomID: The geometry ID of the requested structure. Counting starts from 1.

Example:

see example in *GOAT*

Goat.Get_Num_Of_Geometries

Function *Get_Num_Of_Geometries* acts on goat objects (see *GOAT*) and returns an integer with the number of structures available in the current *GOAT* object.

Syntax:

res = myGoat.Get_Num_Of_Geometries()

Where:

res: An integer with the number of geometries in the current *GOAT* object.

myGoat A GOAT object that is already loaded (meaning it has already parse an ensemble file, see (see *Goat.ParseEnsembleFile*).

Example:

see example in *GOAT*

Goat.ParseEnsembleFile

Function *ParseEnsembleFile* acts on goat objects (see *GOAT*). It will read an ensemble file created by a GOAT run and then update the object with the geometries and energies of each geometry available in the current *GOAT* object.

Syntax:

res = geom.ParseEnesembleFile(filename=myFilename)

Where:

res: An integer that returns '0' if everything worked smoothly.

myFilename The name of the file that will contain the coordinates.

myGoat: A GOAT object previously loaded.

Example:

see example in *GOAT*

NOTE: The old syntax (Parse_Ensemble_File) is still compatible but please do not use it because in the future it will be deprecated.

Goat.Set_Basename

Function *Set_Basename* acts on goat objects (see *GOAT*). It set the basename of the ensemble file created by a GOAT run. It will automatically add the extension: `'.finalensemble.xyz'` in the end of the basename.

Syntax:

myGoat.Set_Basename(myFilename)

Where:

myFilename The basename of the GOAT ensemble file (**without the extension 'finalensemble.xyz'**).

myGOAT: A GOAT object previously loaded.

Example:

see example in *GOAT*

Goat.Print

Function *Print* acts on goat objects (see *GOAT*). It will print in the output the geometries of current GOAT object.

Syntax:

myGoat.Print()

Where:

myGoat: A GOAT object previously loaded.

Example:

see example in *GOAT*

Goat.WriteXYZFile

Function *WriteXYZFile* acts on goat objects (see *GOAT*). It will write the requested geometry to an XYZ file on disk. It expects two arguments:

1. The geometry ID of the geometry and
2. The filename that it will use to store the geometry

Syntax:

myGoat.WriteXYZFile(geomID=myGeomID, filename=myFilename)

Where:

myFilename: The name that it will be used to store the geometry on the disk.

myGeomID: The geometry ID of the requested structure. Counting starts from 1.

Example:

see example in *GOAT*

GetNumOfInstances

The *GetNumOfInstances* returns the number of instances of a specific object in a propertyfile.

Syntax:

[res=] GetNumOfInstances(propertyName=myName, [step=myStep], [filename=myFilename], [baseProperty=true/false])

Where:

res: An integer that returns the number of instances of the required property in the property file.

propertyName: A string alias that defines the variable the user wants to read.

step: The step from which we want to read the property. If not given the property file from the last step will be read.

filename: A filename of a property file. If a filename and at the same time a step are provided the program will ignore the step and try to read the property file with the given filename.

NOTE please note that in the end of filename the extension *.property.txt* will be added.

baseProperty: A true/false boolean. The default value is set to false. If the value is set to true then a generic property of the type asked will be read. This means if dipole moment is asked, it will return the last dipole moment, irrelevant if and MP2 or SCF one wad defined.

Example

```
# -----------------------------------------------------
# This is an example script for readNumOfInstances
# ------------------------------------------------------
%Compound
  Variable res = 0;
  Variable myProperty="MP2_DIPOLE_TOTAL";
  Variable myBaseProperty="DIPOLE_MOMENT_TOTAL";
  New_Step
    !MP2
    #%mp2
    #  density relaxed
    #end
    *xyz 0 1
      H 0.0 0.0 0.0
      H 0.0 0.0 0.8
    *
  Step_End
  # First read the MP2 dipole moment
  res = GetNumOfInstances(propertyName=myProperty);
  print("Num of MP2 dipole moments   : %d\n", res);
  res = GetNumOfInstances(propertyName=myBaseProperty, Property_Base=true);
  print("Num of total dipole moments : %d\n", res);
End
```

GoTo

The *GoTo* command allows the 'jump' inside the normal flow of a *compound* script. The syntax of the command can be best presented through an example.

Example:

```
# -----------------------------------------------------
# This is an example script for GoTo
# (It should print only 0,1,2,3)
# ------------------------------------------------------
%Compound
```

(continues on next page)

(continued from previous page)

```
  Variable TCut=3;
  Variable Done;
  for i from 0 to 6 Do
    print("Index: %d\n", i);
    if (i >= TCut) then
      GoTo Done;
    EndIf
  EndFor
  Done:
    print("Done\n");
End
```

Please note that the variable we use as a label for the *GoTo* command should be previously defined like a normal variable.

If

The *if* block allows the user to make decisions. The syntax in *Compound* is the following:
Syntax:

If *(expression)* **Then**

actions

Else if *(expression)* **Then**

actions

Else

actions

Endif

Below is an example of the usage of *if block* in *compound*.

```
# -------------------------------------------------------------
# This is to check all available ways of 'if blocks'
# -------------------------------------------------------------
Variable x1 = 10.0;
Variable y1 = 20.0;
Variable b1 = False;
Variable b2 = True;
Variable s1 = "alpha";
Variable s2 = "beta";
Variable s3 = "alpha";
print( " ---------------------------------------------------------- \n");
print( " ----------      SUMMARY OF IF CASES        -------------- \n");
print( " ---------------------------------------------------------- \n");

print(" x1: %.1lf\n", x1);
print(" y1: %.1lf\n", y1);
print(" b1: %s\n", b1.GetString());
print(" b2: %s\n", b2.GetString());
print(" s1: %s\n", s1);
print(" s2: %s\n", s2);
print(" s3: %s\n", s3);
# *****************************************************************
#                          DOUBLES
# *****************************************************************
print(" -------------------     Doubles     ----------------------- \n");
print("      Variable/constant / One operator  / if (x1>5)          \n");
print("                    No else if/No else                       \n");
```

(continues on next page)

(continued from previous page)

```
if (x1>5) then
  print(" %.2lf > 5 \n", x1);
endif
# ----------------------------------------------------------------
print("      function / Variable / One operator  / if (3*x1>y1)    \n");
print("                   no else if /   else                      \n");
if (3*x1>y1) then
  print(" 3*%.1lf > %.1lf\n", x1, y1);
else
  print(" 3*%.1lf < %.1lf\n", x1, y1);
endif
# ----------------------------------------------------------------
print("     function / function / One operator  / if (x1-y1>-10.0) \n");
print("                    else if/else                      \n");
if (x1-y1>-10.0) then
  print(" %.2lf - %.2lf > -10.0\n", x1, y1);
else if (x1-y1 < -10.0) then
  print(" %.2lf - %.2lf < -10.0\n", x1, y1);
else
  print(" %.2lf - %.2lf = -10.0\n", x1, y1);
endif
# ****************************************************************
#                          BOOLEANS
# ****************************************************************
print(" ---------------      Booleans      ---------------------------\n");
print("       Variable  /  No operator  /  if (b1)                   \n");
if (b1) then
  print("b1 is True\n");
else
  print("b1 is False\n");
endIf
# --------------------------------------------------------------------
# ---------------------------------------------------------------------
print("       Constant  / No operator  / if (true)                   \n");
if (True) then
  print( "True\n");
else
  print( "False");
endIf
# ---------------------------------------------------------------------
# ---------------------------------------------------------------------
print("       Variable/Variable / AND operator / if (b1 and b2)                 \n
↪");
if (b1 and b2) then
  print("(%s and %s) is true\n", b1.GetString(), b2.GetString());
else
  print("(%s and %s) is not true\n", b1.GetString(), b2.GetString());
endIf
# ---------------------------------------------------------------------
# ---------------------------------------------------------------------
print("       Variable/Variable / OR operator / if (b1 or b2)                 \n");
if (b1 OR b2) then
  print("(%s or %s) is true\n", b1.GetString(), b2.GetString());
else
  print("(%s or %s) is not true\n", b1.GetString(), b2.GetString());
endIf
# ---------------------------------------------------------------------
# ---------------------------------------------------------------------
print("       Bool /Doubles Function / AND operator / if (b1 and x1>y1 )\n");
if (b1 and y1>x1) then
  print("(%s and %.1lf>%.1lf) is True\n", b1.GetString(), x1, y1);
```

(continues on next page)

(continued from previous page)

```
else
  print("(%s and %.1lf>%.1lf) is False\n", b1.GetString(), x1, y1);
endIf
# -----------------------------------------------------------------------
# -----------------------------------------------------------------------
print("       Nested if / if (b2) then if (y1>x1)\n");
if (b2) then
  if (y1 > x1) then
    print ( "(%s is True) and (%.1lf>%.1lf)\n", b2.GetString(), y1, x1);
  else
    print ( "(%s is True) and (%.1lf<%.1lf)\n", b2.GetString(), y1, x1);
  endIf
else
    print ( "%s is False\n", b2.GetString());
endIf
# -----------------------------------------------------------------------
# -----------------------------------------------------------------------
print(" ----------------      Strings     ---------------------------\n");
print(" ------------------------------------------------------------\n");
print(" ------------------------------------------------------------\n");
print("       Variable/Variable / if s1=s2\n");
if (s1=s2) then
  print("%s is same as %s \n", s1, s2);
else
  print("%s is not same as %s \n", s1, s2);
EndIf

# -----------------------------------------------------------------------
# -----------------------------------------------------------------------
print(" ----------------      Strings     ---------------------------\n");
print(" ------------------------------------------------------------\n");
print(" ------------------------------------------------------------\n");
print("       Variable/constant / if s1=\"alpha\"\n");
if (s1="alpha") then
  print("%s is same as %s \n", s1, "alpha");
else
  print("%s is not same as %s \n", s1, "alpha");
EndIf
End
```

Some comments about the syntax:

The *Else if* or *Else* blocks are not obligatory.

The numerical operators that can be used are: '>', '<', '>=', '<=', '='.

The available logical operators are: *'and' and 'or'*.

Unfortunately in the current version multi-parentheses are not allowed.

There is now the possibility to compare strings.

InvertMatrix

Compound can peform matrix algebraic operations, one of the available algebraic operation is the inversion of a matrix. Be carefull that the matrix, whose the invert we are looking for, **must be** a real, square matrix.

Syntax:
AInvert = A.InvertMatrix();

Where:

- *A:* The matrix to be inverted.
- *AInvert:* The invert of A. It can be A itself and then A will just be updated.

Example:

```
# ----------------------------------------------------
# This is an example script for matrix inversion
# -----------------------------------------------------
%Compound
  Variable Dim=3;
  Variable A[Dim][Dim];
  Variable invertA, C;
  Variable res=-1;
  for i from 0 to Dim-1 Do
    for j from 0 to Dim-1 Do
      if (i=j) then
        A[i][j] = i+1;
      else
        A[i][j] =  0.0;
      EndIf
    EndFor
  EndFor
  invertA = A.invertMatrix();
  A.PrintMatrix();
  invertA.PrintMatrix();
  C = Mat_x_Mat(A,invertA,false, false, 1.0, 1.0);
  C.PrintMatrix();
End
```

Mat_p_Mat

Compound can peform matrix algebraic operations, one of the available algebraic operation is matrix addittion. In order to add two matrices they **must** have the same dimensions.

Syntax:
C=Mat_p_Mat(alpha, A, beta, B);

Where:

- *C:* The resulting matrix.
- *alpha:* The coefficient for matrix A.
- *A:* The left matrix of the addition.
- *beta:* The coefficient for matrix B.
- *B:* The right matrix of the addition.

Example:

```
# ----------------------------------------------------
# This is an example script for matrix addition
# ----------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
%Compound
  Variable Dim=3;
  Variable A[Dim][Dim];
  Variable B[Dim][Dim];
  Variable C;
  Variable res=-1;
  for i from 0 to Dim-1 Do
    for j from 0 to Dim-1 Do
      A[i][j] = 1.0;
      B[i][j] = 2.0;
    EndFor
  EndFor
  A.PrintMatrix();
  B.PrintMatrix();
  C = Mat_p_Mat(2.0, A, 3.0, B);
  C.PrintMatrix();
End
```

Mat_x_Mat

Compound can peform matrix algebraic operations, one of the available algebraic operation is matrix multiplication. In general we can multiply each matrix with constants *alpha* and *beta* so that the general multiplication is:

*C=(alpha*A)*(beta*B)

In addition each of matrices A and B are allowed to be transposed.

Syntax:
C=Mat_x_Mat(A, B, [transposeA], [transposeB], [alpha], [beta]);

Where:

- *C:* The resulting matrix.
- *A:* The left matrix of the multiplication.
- *B:* The right matrix of the multiplication.
- *transposeA:* A boolean to state if matrix A should be transposed before the mutliplication (default: False).
- *transposeB:* A boolean to state if matrix B should be transposed before the multiplication (default: False).
- *alpha:* A scalar to multiply matrix A before the mutliplication (default 1.0).
- *beta:* A scalar to mutliply matrix B before the multiplication (default 1.0).

Example:

```
# ----------------------------------------------------
# This is an example script for matrix multiplication
# -----------------------------------------------------
%Compound
  Variable Dim=3;
  Variable A[Dim][Dim];
  Variable invertA;
  Variable C;
  Variable D;
  Variable res=-1;
  for i from 0 to Dim-1 Do
    for j from 0 to Dim-1 Do
      if (i=j) then
        A[i][j] = i+1;
      else
        A[i][j] =  0.0;
```

(continues on next page)

(continued from previous page)

```
      EndIf
    EndFor
  EndFor
  A.invertMatrix(invertA);
  A.PrintMatrix();
  invertA.PrintMatrix();
  C = Mat_x_Mat(A,invertA,false, false, 1.0, 1.0);
  C.PrintMatrix();
  C = Mat_x_Mat(A, invertA, true, true, 1.0, 1.0);
  C.PrintMatrix();
  C = Mat_x_Mat(A, invertA, false, false, 2.0);
  C.PrintMatrix();
  C = Mat_x_Mat(A, invertA, false, false, 2.0, 3.0);
  C.PrintMatrix();
End
```

Mat_x_Scal

Compound can peform matrix algebraic operations, one of the available algebraic operation is multiplication of the elements of a matrix with a scalar. The function returns the multiplied matrix that can be the one that we use as an argument in the parenthesis, meaning it is updated, or a different one.

Syntax:
C=Mat_x_Scal(alpha, A);

Where:

- *C:* The resulting matrix.
- *alpha:* A scalar to multiply the elements of matrix A.
- *A:* The matrix to be mutliplied.

Example:

```
# ----------------------------------------------------
# This is an example script for matrix times scalar
# -----------------------------------------------------
%Compound
  Variable Dim=3;
  Variable A[Dim][Dim];
  Variable alpha=2.0;
  Variable C;
  for i from 0 to Dim-1 Do
    for j from 0 to Dim-1 Do
        A[i][j] = i+j;
    EndFor
  EndFor
  A.PrintMatrix();
  C = Mat_x_Scal(alpha,A);
  A = Mat_x_Scal(alpha,A);
  A.PrintMatrix();
  C.PrintMatrix();
End
```

NewGeom

NewGeom is a platform for geometry manipulation. The basic idea is to have functions that can read a geometry and then produce one or more new geometries with some characteristics that we need. For the moment under the umbrella of *NewGeom* fall 4 different functions, and these are: Displace, Remove_Atom, Remove_Element.

NOTE: The old syntax (New_Geom) is still compatible but please do not use it because in the future it will be deprecated.

Displace

The idea behind "*Displace*" is to have a structure, perform an analytical frequncy calculation on it (currently we do not store numerical frequencies in the property file) and then read the Hessian from this calculation to adjust the geometry based on a normal mode of vibration that we choose. The syntax of this command is:

syntax: NewGeom = (Displace, step, hessian, frequency, scaling)

where:
step is the step from which we choose the original geometry
hessian is the hessian read from a property file
frequency defines which normal mode we will use and
scaling is a factor of how severe we want the displacement to be.

We should note that *Displace* is the only command of the *new_geom* family of commands that will not store a geometry on disk but only internally pass the new geometry to the next calculation. An example of the usage of this command can be found in the script 'iterativeOptimization' that is given with ORCA. The relevant part is as follows: **example:**

```
# Define variables
                Variable MaxNTries    =  25;
                Variable CutOff       = -50;
                Variable displacement = 0.6;
                Variable NNegative =   0;
                Variable freqs[];
                Variable modes[];
                Variable NFreq;
                Variable limit;
                Variable done;
                Variable FinalEnergy;

                # ==============================================================
                # Start a for loop over number of tries
                # ==============================================================
                For itry From 1 To maxNTries Do

                # ----------------------------------
                # Run a geometry optimization
                # ----------------------------------
                New_Step
                ! tightopt freq verytightscf nopop def2-TZVP xyzfile
                Step_End
                Read freqs  = THERMO_FREQS[itry];
                Read modes  = HESSIAN_MODES[itry];
                Read NFreq  = THERMO_NUM_OF_FREQS[itry];
                limit = NFreq - 1;
                # ----------------------------------
                #  check for sufficeintly negative
                #  frequencies
                # ----------------------------------
                NNegative = 0;
                For ifreq From 0 to limit Do
                if ( freqs[ifreq] < CutOff )  then
                New_Geom = (Displace, itry,  modes,  ifreq, displacement);
```