V.GetDouble()

This function works on a variable. It will return a double value in case the variable is integer or double. In all other cases the program will crash providing a relevant message.

Syntax:

myVar.GetDouble();

where:

myVar is an already initialized variable.

Example

```
# -----------------------------------------------------
# This is an example script for
#  Variable functions
# ------------------------------------------------------
%Compound
  Variable double=1.0;
  Variable integer=2;
  Variable iToBool = integer.GetBool();
  Variable boolean=false;

  print("----------------------------------------\n");
  print("     Results for translation functions \n");
  print("Double  to integer : %d    (it should print 1)\n", double.GetInteger());
  print("Integer to double  : %.2lf (it should print 2.00)\n", integer.
↪GetDouble());
  print("Boolean to string  : %s    (it should print FALSE)\n", boolean.
↪GetString());
  print("Integer to boolean : %s    (it should print TRUE)\n", iToBool.
↪GetString());
  print("Double  to string  : %s    (it should print 1.00000000000000000000e+00)\n
↪", double.GetString());
  print("Integer to string  : %s    (it should print 2)\n", integer.GetString());
End
```

V.GetInteger()

This function works on a variable. It will return an integer value in case the variable is integer or double. In all other cases the program will crash providing a relevant message.

Syntax:

myVar.GetInteger();

where:

myVar is an already initialized variable.

Example

```
# -----------------------------------------------------
# This is an example script for
#  Variable functions
# ------------------------------------------------------
%Compound
  Variable double=1.0;
  Variable integer=2;
  Variable iToBool = integer.GetBool();
  Variable boolean=false;
```

(continues on next page)

(continued from previous page)

```
  print("----------------------------------------\n");
  print("     Results for translation functions \n");
  print("Double  to integer : %d    (it should print 1)\n", double.GetInteger());
  print("Integer to double  : %.2lf (it should print 2.00)\n", integer.
↪GetDouble());
  print("Boolean to string  : %s    (it should print FALSE)\n", boolean.
↪GetString());
  print("Integer to boolean : %s    (it should print TRUE)\n", iToBool.
↪GetString());
  print("Double  to string  : %s    (it should print 1.00000000000000000000e+00)\n
↪", double.GetString());
  print("Integer to string  : %s    (it should print 2)\n", integer.GetString());
End
```

V.GetSize()

This function works on a variable. If the variable is a scalar it will return 1. If the variable is a 1-Dimensional array it will return the size of the array which is the same with the *GetDim1()* . If the variable is a 2-Dimensional array it will return the results Dim1*Dim2.

Syntax:

myVar.GetSize();

where:

myVar is an already initialized variable.

Example

```
# ----------------------------------------------------
# This is an example script for
#  Variable functions
# -----------------------------------------------------
%Compound
  Variable dim1, dim2, size;
  Variable A;
  Variable B[3];
  Variable C[3][2];

  print("----------------------------------------\n");
  print("          Results for scalar  \n");
  print("Dim1 : %d (it should print 1)\n", A.GetDim1());
  print("Dim2 : %d (it should print 1)\n", A.GetDim2());
  print("Size : %d (it should print 1)\n", A.GetSize());
  print("----------------------------------------\n");
  print("          Results for 1D-Array  \n");
  print("Dim1 : %d (it should print 3)\n", B.GetDim1());
  print("Dim2 : %d (it should print 1)\n", B.GetDim2());
  print("Size : %d (it should print 3)\n", B.GetSize());
  print("----------------------------------------\n");
  print("          Results for 2D-Array  \n");
  print("Dim1 : %d (it should print 3)\n", C.GetDim1());
  print("Dim2 : %d (it should print 2)\n", C.GetDim2());
  print("Size : %d (it should print 6)\n", C.GetSize());
End
```

V.GetString()

This function works on a variable. It will return a string of the value of the variable. It works for doubles, integers and booleans.

Syntax:

myVar.GetString();

where:

myVar is an already initialized variable.

Example

```
# -----------------------------------------------------
# This is an example script for
#  Variable functions
# ------------------------------------------------------
%Compound
  Variable double=1.0;
  Variable integer=2;
  Variable iToBool = integer.GetBool();
  Variable boolean=false;

  print("----------------------------------------\n");
  print("     Results for translation functions \n");
  print("Double  to integer : %d    (it should print 1)\n", double.GetInteger());
  print("Integer to double  : %.2lf (it should print 2.00)\n", integer.
↪GetDouble());
  print("Boolean to string  : %s    (it should print FALSE)\n", boolean.
↪GetString());
  print("Integer to boolean : %s    (it should print TRUE)\n", iToBool.
↪GetString());
  print("Double  to string  : %s    (it should print 1.00000000000000000000e+00)\n
↪", double.GetString());
  print("Integer to string  : %s    (it should print 2)\n", integer.GetString());
End
```

V.PrintMatrix()

This function works on variables. It will print print an array on a format with 8 columns.

Syntax: *myVar.PrintMatrix([NCols=numOfColumns]);*

where:

myVar is an already initialized variable.

numOfColumns is the desired number of columns for the printing. This is not obligatory and if not used then by default ORCA will print using **4** columns.

Example

Example:

```
# ----------------------------------------------
#  A script to check PrintMatrix
# ----------------------------------------------
%Compound
  Variable Dim1 = 5;
  Variable Dim2 = 16;
  Variable x[Dim1][Dim2];
  for i from 0 to Dim1-1 Do
```

(continues on next page)

(continued from previous page)

```
    for j from 0 to Dim2-1 Do
      x[i][j] = i+j;
    EndFor;
  EndFor;

  x.PrintMatrix();        # This should print with 4 columns
  x.PrintMatrix(NCols=8); # This should print with 8 columns
EndRun
```

NOTE In case of scalars it will only print the header without any values.

NOTE It only works for arrays of type *'double'* or type *'integer'*. With all variables of other types the program will exit providing an error message.

With

The purpose of the "with" command is to add the ability to call compound while adjusting some of the variables that are already defined in the compound file. This means that if there is a variable defined in the compound file and a value is assigned to it, we can during the call change the assigned value of this variable.

One can pass numbers, string or boolean variables.

It should be noted that it is not possible to call array variables this way. Beside this restriction, the syntax of the variable assignment in the case of with is the same with the variable assignment in a normal *Compound* script.

An important note here is that in case we use the *With* command the *%Compound* block should end with an *'End'* even if we call a *Compound* script file.
Syntax:
% compound *"filename"*
With
var1 = *val1*;
var2 = *val2*;
End
Example:

```
# ------------------------------------------------------------
# This is to check all available ways of variable assignement
#   in combination with the 'with' calls.
# ------------------------------------------------------------

# ---------------------------------
# Some necessary initial definitions
# ----------------------------------
Variable x1, x2, x3, x4;

# -----------------------------------
# Now the assignments
# -----------------------------------
#Scalars doubles
x1 = 1.0;
#Scalars integers
x2 = 1;
#Scalars strings
x3 = "test";
#Scalars bools
x4 = True;
print( " --------------------------------------------- \n");
print( " ------- SUMMARY OF WITH ASSIGNMENTS --------- \n");
print( " --------------------------------------------- \n");
print( " The calling input:\n");
```

(continues on next page)

(continued from previous page)

```
print("%Compound \"0975.cmp\"\n");
print("  with\n");
print("    x1       = 3.0;\n");
print("    x2       = 2;\n");
print("    x3       = \"with\";\n");
print("    x4       = False;\n");
print("end\n");
print( " ---------------- Scalars -------------------- \n");
print( " x1 (1.0)      :  %.2lf\n", x1);
print( " x2 (1)        :  %d\n", x2);
print( " x3 (\"test\")   :  %s\n", x3);
print( " x4 (True)     :  %s\n", x4.GetString());
#print( " x6      :  %s\n",    x6);
#if (x4) then
#  print(" x4      : TRUE\n");
#else
#  print(" x4   : FALSE\n");
#endIfo
End
```

Write2File

With the *Print* command (see *Print*) one can write in the ORCA output. Nevertheless it might be that one would prefer to write to a different file. In *Compound* one can achieve this using the *write2File* command. The syntax follows closely the syntax of ‘fprintf’ command of the programming language C. The arguments definition and the syntax is identical with the syntax of the *Compound ‘Print’* command with the addition that one should define a file object to send the printing.

Syntax:
Write2File(file variable, format string, [variables]);

Where:

file variable: is a predefined variable corresponding to an already open, through the *OpenFile* command, file.

format string and *variables* follow exactly the syntax of the *Print* command, so for more details please refere to section *Print*.

NOTE Please remember once everything is writen to the file to close the file, using the *CloseFile* command (see *CloseFile*).

Example:

```
%Compound
  # -------------------------------------------------------------
  #      This is to check all available write2String and
  #                 write2File options
  # -------------------------------------------------------------
  Variable xS    = "test_";
  Variable xI    = 1;
  Variable final;
  Variable fp;
  Variable myFilename = "0955.txt";


  #Create also a file object
  fp = OpenFile(myFilename, "w");
  write2String(final, "   ----- Test ----- \n");
  write2File(fp, "%s", final);
  CloseFile(fp);
```

(continues on next page)

(continued from previous page)

```
  print( " ---------------------------------------------- \n");
  print( " ------ SUMMARY OF WRITE2STRING AND  --------- \n");
  print( " ------          WRITE2FILE          --------- \n");
  print( " ---------------------------------------------- \n");
  write2String(final, "%s", "constant" );
  print( " Final      : %s\n", final);
  write2String(final,"%s", "constant" );      #No space before the quotation marks
  print( " Final      : %s\n", final);
  write2String(final,  "%s", "constant" );    #More than one spaces before
  print( " Final      : %s\n", final);
  write2String(final,"    %s", "constant" );  #No spaces before but  more␣
↪afterwards
  print( " Final      : %s\n", final);
  write2String(final,   "  %s", "constant" ); #More spaces before and more␣
↪afterwards
  print( " Final      : %s\n", final);
  write2String(final, "%s", xS);
  print( " Final      : %s\n", final);
  write2String(final, "%s_%d", xS, xI);
  print( " Final      : %s_%d\n", final, xI);
  write2String(final, "%s_%d", xS,2*xI+1);
  print( " Final      : %s\n", final);
End
```

Write2String

In case one needs to construct a string using some variables, *Compound* provides the *Write2String* command. The syntax of the command is identical with the *Write2File* (see *Write2File*) command with the only exception that instead of a file we should provide the name of a variable that is already declared in the file. The syntax of the format and the variables used is identical with the *Print* command (please refer to *Print*.)

Syntax:

Write2String(variable, format string, [variables]);

where:

variable: is the name of a variable that should already be declared.

format string and *variables* follow exactly the syntax of the *Print* command, so for more details please refer to section *Print*.

Example:

```
%Compound
  # --------------------------------------------------------------
  #      This is to check all available write2String and
  #                 write2File options
  # --------------------------------------------------------------
  Variable xS    = "test_";
  Variable xI    = 1;
  Variable final;
  Variable fp;
  Variable myFilename = "0955.txt";


  #Create also a file object
  fp = OpenFile(myFilename, "w");
  write2String(final, "   ----- Test ----- \n");
  write2File(fp, "%s", final);
  CloseFile(fp);
```

(continues on next page)

(continued from previous page)

```
  print( " --------------------------------------------- \n");
  print( " ------ SUMMARY OF WRITE2STRING AND  --------- \n");
  print( " ------          WRITE2FILE          --------- \n");
  print( " --------------------------------------------- \n");
  write2String(final, "%s", "constant" );
  print( " Final      : %s\n", final);
  write2String(final,"%s", "constant" );      #No space before the quotation marks
  print( " Final      : %s\n", final);
  write2String(final,  "%s", "constant" );    #More than one spaces before
  print( " Final      : %s\n", final);
  write2String(final,"    %s", "constant" );  #No spaces before but  more␣
↪afterwards
  print( " Final      : %s\n", final);
  write2String(final,   "   %s", "constant" ); #More spaces before and more␣
↪afterwards
  print( " Final      : %s\n", final);
  write2String(final, "%s", xS);
  print( " Final      : %s\n", final);
  write2String(final, "%s_%d", xS, xI);
  print( " Final      : %s_%d\n", final, xI);
  write2String(final, "%s_%d", xS,2*xI+1);
  print( " Final      : %s\n", final);
End
```

8.3.2 List of known Properties

The name and a sort explanation of all the known variables that can be automatically recovered, from the property file, are given in the next table

Table 8.2: Variables, known to the compound block, with short ex

==	==
==	==AUTOC
==	==
++	++Energies++++
AUTOCI_REF_ENERGY	AutoCI Reference Energy
AUTOCI_CORR_ENERGY	AutoCI Correlatioin Energy
AUTOCI_TOTAL_ENERGY	AutoCI Total Energy
++	+++++++++++++++++++++++++++++++++++++ ENERGY Gradie
AUTOCI_NUCLEAR_GRADIENT	AutoCI Energy nuclear gradient
AUTOCI_NUCLEAR_GRADIENT_NORM	AutoCI Norm of the nuclear gradient
AUTOCI_NUCLEAR_GRADIENT_ATOM_NUMBERS	AutoCI The atomic numbers of the atoms in the gradient
++	+++++++++++++++++++++++++ Electric Properties (Dipole mom
AUTOCI_DIPOLE_MAGNITUDE	AutoCI The value of the dipole moment
AUTOCI_DIPOLE_ELEC_CONTRIB	AutoCI The electronic contribution to the dipole moment
AUTOCI_DIPOLE_NUC_CONTRIB	AutoCI The nuclear contribution to the dipole moment
AUTOCI_DIPOLE_TOTAL	AutoCI The total dipole moment
SCF_ENERGY	SCF Energy
++	++++++++++++++++++++++++ Electric Properties (Polarizability
AUTOCI_POLAR_ISOTROPIC	AutoCI The polarizability isotropic value
AUTOCI_POLAR_RAW	AutoCI The raw polarizability tensor
AUTOCI_POLAR_DIAG_TENSOR	AutoCI The polarizability diagonalized tensor
AUTOCI_POLAR_ORIENTATION	AutoCI The polarizability orientation (eigenvectors)
++	+++++++++++++++++++++ Electric Properties (Quadrupole mom

Table 8.2 – continued from previous page

AUTOCI_QUADRUPOLE_MOMENT_ISOTROPIC	AutoCI The quadrupole moment isotropic value
AUTOCI_QUADRUPOLE_MOMENT_DIAG_TENSOR	AutoCI The quadrupole moment diagonalized tensor
AUTOCI_QUADRUPOLE_MOMENT_ELEC_CONTRIB	AutoCI The elctronic contribution to the quadrupole moment tensor
AUTOCI_QUADRUPOLE_MOMENT_NUC_CONTRIB	AutoCI The nuclear contribution to the quadrupole moment tensor
AUTOCI_QUADRUPOLE_MOMENT_TOTAL	AutoCI The total quadrupole moment
+++++++++++++++++++++++++++++++++++++++	+++++++++++++++++++++++++++ Magnetic Properties (D Tensor
AUTOCI_D_TENSOR_EIGENVALUES	AutoCI The D Tensor eigenvalues
AUTOCI_D_TENSOR_EIGENVECTORS	AutoCI The D Tensor eigenvectors
AUTOCI_D_TENSOR_RAW	AutoCI The Raw D Tensor
AUTOCI_D_TENSOR_D	AutoCI The final D value for the D Tensor
AUTOCI_D_TENSOR_E	AutoCI The final E value for the D Tensor
AUTOCI_D_TENSOR_MULTIPLICITY	AutoCI The spin-multiplicity used for the D Tensor calculation
+++++++++++++++++++++++++++++++++++++++	+++++++++++++++++++++++++++ Magnetic Properties (G Tensor
AUTOCI_G_TENSOR_RAW	AutoCI The Raw G Tensor
AUTOCI_G_TENSOR_ELEC	AutoCI The Electronic part of the G Tensor
AUTOCI_G_TENSOR_TOT	AutoCI The Total G Tensor
AUTOCI_G_TENSOR_ISO	AutoCI The isotropic g value
AUTOCI_G_TENSOR_ORIENTATION	AutoCI The G Tensor orientation (eigenvectors)
SCF_ENERGY	SCF Energy
VDW_CORRECTION	van der Waals correction
SCF Electric properties	
SCF_DIPOLE_MAGNITUDE.	SCF dipole moment (debye)
SCF_DIPOLE_ELEC_CONTRIB	SCF Electronic contribution to dipole moment
SCF_DIPOLE_NUC_CONTRIB	SCF Nuclear contribution to dipole moment
SCF_DIPOLE_TOTAL	SCF Total dipole moment
SCF_QUADRUPOLE_ISOTROPIC	SCF isotropic quadrupole moment
SCF_QUADRUPOLE_DIAG_TENSOR	SCF quadrupole moment diagonalised tensor
SCF_QUADRUPOLE_ELEC_CONTRIB	SCF electronic contribution to the quadrupole moment
SCF_QUADRUPOLE_NUC_CONTRIB	SCF nuclear contribution to the quadrupole moment
SCF_QUADRUPOLE_TOTAL	SCF total quadrupole moment
SCF_POLAR_ISOTROPIC	SCF isotropic polarizability
SCF_POLAR_RAW	SCF polarizability raw tensor
SCF_POLAR_DIAG_TENSOR	SCF diagonaised polarizability tensor
DBOC Energy Correction	
DBOC_ENERGY	The Diagonal Born-Oppenheimer energy correction
DFT	
DFT_NUM_OF_ALPHA_EL	Number of alpha electrons
DFT_NUM_OF_BETA_EL	Number of beta electrons
DFT_NUM_OF_TOTAL_EL	Total number of electrons
DFT_TOTAL_EN	DFT Total energy
DFT_EXCHANGE_EN	DFT Exchange energy
DFT_CORR_EN	DFT Correlation Energy
DFT_XC_EN	DFT Exchange-Correlation Energy
DFT_NON_LOC_EN	DFT Non-Local correlation
DFT_EMBED_CORR	DFT Embedding correction
gCP correction	
GCP_CORRECTION	gCP energy correction

Table 8.2 – continued from previous page

MP2	
MP2_REF_ENERGY	Reference SCF Energy
MP2_CORR_ENERGY	MP2 Correlation energy
MP2_TOTAL_ENERGY	Total Energy (SCF + MP2)
MP2 Electric properties	
MP2_DIPOLE_MAGNITUDE.	MP2 dipole moment (debye)
MP2_DIPOLE_ELEC_CONTRIB	MP2 Electronic contribution to dipole moment
MP2_DIPOLE_NUC_CONTRIB	MP2 Nuclear contribution to dipole moment
MP2_DIPOLE_TOTAL	MP2 Total dipole moment
MP2_QUADRUPOLE_ISOTROPIC	MP2 isotropic quadrupole moment
MP2_QUADRUPOLE_DIAG_TENSOR	MP2 quadrupole moment diagonalised tensor
MP2_QUADRUPOLE_ELEC_CONTRIB	MP2 electronic contribution to the quadrupole moment
MP2_QUADRUPOLE_NUC_CONTRIB	MP2 nuclear contribution to the quadrupole moment
MP2_QUADRUPOLE_TOTAL	MP2 total quadrupole moment
MP2_POLAR_ISOTROPIC	MP2 isotropic polarizability
MP2_POLAR_RAW	MP2 polarizability raw tensor
MP2_POLAR_DIAG_TENSOR	MP2 diagonaised polarizability tensor
MDCI	
MDCI_REF_ENERGY	Reference SCF Energy
MDCI_CORR_ENERGY	Total Correlation Energy
MDCI_TOTAL_ENERGY	Total Energy (SCF + Correlation)
MDCI_ALPHA_ALPHA_CORR_ENERGY	Correlation energy from $\alpha\alpha$ electron pairs
MDCI_BETA_BETA_CORR_ENERGY	Correlation energy from $\beta\beta$ electron pairs
MDCI_ALPHA_BETA_CORR_ENERGY	Correlation energy from $\alpha\beta$ electron pairs
MDCI_DSINGLET_CORR_ENERGY	Correlation energy from singlet electron pairs (only for closed-shell)(
MDCI_DTRIPLET_CORR_ENERGY	Correlation energy from triplet electron pairs (only for closed-shell) (
MDCI_SSINGLET_CORR_ENERGY	Correlation energy from singlet electron pairs (only for closed-shell) (
MDCI_STRIPLET_CORR_ENERGY	Correlation energy from triplet electron pairs (only for closed-shell) (
MDCI_TRIPLES_ENERGY	Perturbative triples correlation energy
MDCI_ALL_ELECTRONS	Total number of electrons
MDCI_CORR_ELECTRONS	Number of correlated electrons
MDCI_CORR_ALPHA_ELECTRONS	Number of correlated α electrons
MDCI_CORR_BETA_ELECTRONS	Number of correlated β electrons
MDCI Electric properties	
MDCI_DIPOLE_MAGNITUDE	MDCI dipole moment (debye)
MDCI_DIPOLE_ELEC_CONTRIB	MDCI Electronic contribution to dipole moment
MDCI_DIPOLE_NUC_CONTRIB	MDCI Nuclear contribution to dipole moment
MDCI_DIPOLE_TOTAL	MDCI Total dipole moment
MDCI_QUADRUPOLE_ISOTROPIC	MDCI isotropic quadrupole moment
MDCI_QUADRUPOLE_DIAG_TENSOR	MDCI quadrupole moment diagonalised tensor
MDCI_QUADRUPOLE_ELEC_CONTRIB	MDCI electronic contribution to the quadrupole moment
MDCI_QUADRUPOLE_NUC_CONTRIB	MDCI nuclear contribution to the quadrupole moment
MDCI_QUADRUPOLE_TOTAL	MDCI total quadrupole moment
MDCI_POLAR_ISOTROPIC	MDCI isotropic polarizability
MDCI_POLAR_RAW	MDCI polarizability raw tensor
MDCI_POLAR_DIAG_TENSOR	MDCI diagonaised polarizability tensor

Table 8.2 – continued from previous page

CASSCF	
CASSCF_NUM_OF_MULTS	The number of CASSCF spin multiplicities
CASSCF_NUM_OF_IRREPS	The number of CASSCF irreps
CASSCF_FINAL_ENERGY	The CASSCF final energy
PT2_NUM_OF_MULTS	The CASPT2 spin multiplicities
PT2_NUM_OF_IRREPS	The number of CASPT2 irreps
PT2_FINAL_ENERGY	The CASPT2 Energy
DCDCAS_NUM_OF_MULTS	The number of DCDCAS spin multiplicities
DCDCAS_NUM_OF_IRREPS	The number of DCDCAS irreps
DCDCAS_FINAL_ENERGY	The DCDCAS Energy
CASSCF_ABS_SPECTRUM	The CASSCF Absorption spectrum
CASSCF_ABS_SPECTRUM_INFO	Information about the excitations of the CASSCF spectrum
CASSCF_ABS_SPECTRUM_NROOTS	The number of Roots
CASSCF_CD_SPECTRUM	The CASSCF CD spectrum
CASSCF_CD_SPECTRUM_INFO	Information about the excitations of the CASSCF CD spectrum
CASSCF_CD_SPECTRUM_NROOTS	The number or roots
CASPT2_ABS_SPECTRUM	The CASPT2 Absorption spectrum
CASPT2_ABS_SPECTRUM_INFO	Information about the excitations of the CASPT2 spectrum
CASPT2_ABS_SPECTRUM_NROOTS	The number of roots
CASPT2_CD_SPECTRUM	The CASPT2 CD spectrum
CASPT2_CD_SPECTRUM_INFO	Information about the excitations of the CASPT2 CD spectrum
CASPT2_CD_SPECTRUM_NROOTS	The number of roots
CAS_CUSTOM_ABS_SPECTRUM	The Custom CASSCF Absorption spectrum
CAS_CUSTOM_ABS_SPECTRUM_INFO	Information about the excitations of the custom CASSCF absorption
CAS_CUSTOM_ABS_SPECTRUM_NROOTS	The number of roots
CAS_CUSTOM_CD_SPECTRUM	The Custom CASSCF CD spectrum
CAS_CUSTOM_CD_SPECTRUM_INFO	Information about the excitations of the custom CASSCF CD spectru
CAS_CUSTOM_CD_SPECTRUM_NROOTS	The number of roots
DCDCAS_ABS_SPECTRUM	The DCDCAS Absorption spectrum
DCDCAS_ABS_SPECTRUM_INFO	Information about the excitations of the DCDCAS absorption spectru
DCDCAS_ABS_SPECTRUM_NROOTS	The number of roots
CASSCF_DTENSOR_EIGENVALUES	CASSCF D Tensor eigenvalues
CASSCF_DTENSOR_RAW_EIGENVECTORS	CASSCF D Tensor Raw eigenvectors
CASSCF_DTENSOR_D	D value of CASSCF ZFS
CASSCF_DTENSOR_E	E value of CASSCF ZFS
CASSCF_DTENSOR_MULTIPLICITY	Spin multiplicity
CASPT2_DTENSOR_EIGENVALUES	CASPT2 D Tensor eigenvalues
CASPT2_DTENSOR_RAW_EIGENVECTORS	CASPT2 D Tensor raw eigenvectors
CASPT2_DTENSOR_D	D value of CASPT2 ZFS
CASPT2_DTENSOR_E	E value of CASPT2 ZFS
CASPT2_DTENSOR_MULTIPLICITY	Spin multiplicity
CAS_CUSTOM_DTENSOR_EIGENVALUES	custom CASSCF D Tensor eigenvalues
CAS_CUSTOM_DTENSOR_RAW_EIGENVECTORS	custom CASSCF D Tensor Raw eigenvectors
CAS_CUSTOM_DTENSOR_D	D value of custom CASSCF ZFS
CAS_CUSTOM_DTENSOR_E	E value of custom CASSCF ZFS
CAS_CUSTOM_DTENSOR_MULTIPLICITY	Spin multiplicity
CIPSI	
CIPSI_SPIN_MULTIPLICITY	The CIPSI spin multiplicity
CIPSI_NUM_OF_ROOTS	The CIPSI number of roots
CIPSI_FINAL_ENERGY	The CIPSI Final energy
CIPSI_ENERGIES	The CIPSI Energies

Table 8.2 – continued from previous page

CIS	
CIS_FINAL_ENERGY	The final total energy
CIS_ESCF	The SCF Energy
CIS_E0	The Energy of the ground state
CIS_ENERGIES	The singlet energies
CIS_ENERGIESP1	The triplet energies
CIS_MODE	One of the CIS modes
CIS_NUM_OF_ROOTS	The number of roots
CIS_ROOT	State to be optimized
CIS_ABS_SPECTRUM_NROOTS	The number of roots
CIS_ABS_SPECTRUM	The CIS absorption spectrum
CIS_ABS_SPECTRUM_VELOCITY	The CIS absorptioin spectrum in velocity representation
CIS_ABS_SOC_SPECTRUM_NROOTS	The number or roots
CIS_ABS_SOC_SPECTRUM	The CIS absorption spectrum including SOC
CIS_CD_SPECTRUM_NROOTS	The number of roots
CIS_CD_SPECTRUM	The CIS CD spectrum
CIS_CD_SOC_SPECTRUM_NROOTS	The number of roots
CIS_CD_SOC_SPECTRUM	The CIS CD spectrum including SOC
ROCIS	
ROCIS_STATE	ROCIS State
ROCIS_REF_ENERGY	ROCIS Reference energy
ROCIS_CORR_ENERGY	ROCIS correlation energy
ROCIS_TOTAL_ENERGY	ROCIS total energy
ROCIS_ABS_SPECTRUM_NROOTS	Number of roots
ROCIS_ABS_SPECTRUM	ROCIS Absorption spectrum
ROCIS_ABS_SOC_SPECTRUM_NROOTS	Number of roots
ROCIS_ABS_SOC_SPECTRUM	ROCIS absorption spectrum including SOC
ROCIS_CD_SPECTRUM_NROOTS	Number of roots
ROCIS_CD_SPECTRUM	ROCIS CD spectrum
ROCIS_CD_SOC_SPECTRUM_NROOTS	Number of roots
ROCIS_CD_SOC_SPECTRUM	ROCIS CD spectrum including SOC
MRCI	
MRCI_ABS_SPECTRUM	The MRCI absorption spectrum
MRCI_ABS_SPECTRUM_INFO	Information about the absorption spectrum
MRCI_ABS_SPECTRUM_NROOTS	The number of roots
MRCI_CD_SPECTRUM	The MRCI CD spectrum
MRCI_CD_SPECTRUM_INFO	Information about the MRCI CD spectrum
MRCI_CD_SPECTRUM_NROOTS	The number of roots
MRCI_DIPOLE_MOMENTS	The MRCI dipole moments
MRCI_DIPOLE_MOMENTS_INFO	Information about the MRCI dipole moments
MRCI_DTENSOR_EIGENVECTORS	The eigenvectors of the MRCI D tensor
MRCI_DTENSOR_EIGENVALUES	The eigenvalues of the MRCI D tensor
MRCI_DTENSOR_RAW_EIGENVECTORS	The raw eigenvectors of the MRCI D tensor
MRCI_DTENSOR_D	The MRCI D value for the ZFS
MRCI_DTENSOR_E	The MRCI E value for the ZFS
MRCI_DTENSOR_MULTIPLICITY	The MRCI spin multiplicity
EXTRAPOLATION	
EXTRAP_SCF_ENERGIES	The SCF energies with the different basis sets

Table 8.2 – continued from previous page

EXTRAP_CBS_SCF	The extrapolated SCF energy
EXTRAP_CORR_ENERGIES	The correlation energies with the different basis sets
EXTRAP_CBS_CORR	The extrapolated correlatioin energy
EXTRAP_CBS_TOTAL	The extrapolated total energy
EXTRAP_CCSDT_X	The (T) contribution to the energy
EXTRAP_NUM_OF_ENERGIES	The number of energies (basis sets) used for the extrapolation
THERMOCHEMISTRY	
THERMO_TEMPERATURE	Temperature (^oK)
THERMO_PRESSURE	Pressure (Atm)
THERMO_TOTAL_MASS	Total Mass of the molecule (AMU)
THERMO_SPIN_DEGENERACY	Electronic degeneracy
THERMO_ELEC_ENERGY	Electronic energy (Eh)
THERMO_TRANS_ENERGY	Translational energy (Eh)
THERMO_ROT_ENERGY	Rotational energy (Eh)
THERMO_VIB_ENERGY	Vibrational energy (Eh)
THERMO_NUM_OF_FREQS	The number of vibrational frequencies
THERMO_FREQS	Frequencies
THERMO_ZPE	Zero point energy (Eh)
THERMO_INNER_ENERGY_U	Inner Energy (Eh)
THERMO_ENTHALPY_H	Enthalpy (Eh)
THERMO_ELEC_ENTROPY	(Electronic Entropy)*T (Eh)
THERMO_ROR_ENTROPY	(Rotational Entropy)*T (Eh)
THERMO_VIB_ENTROPY	(Vibrational Entropy)*T (Eh)
THERMO_TRANS_ENTROPY	(Translational Entropy)*T (Eh)
THERMO_ENTROPY_S	(Total Entropy)*T (Eh)
THERMO_FREE_ENERGY_G	Free Energy (Eh)
EPR-NPR Spin-Spin coupling	
EPRNMR_SSC_NUM_OF_NUC_PAIRS	Number of nuclear pairs to calculate something
EPRNMR_SSC_NUM_OF_NUC_PAIRS_DSO	Number of nuclear pairs to calculate DSO terms
EPRNMR_SSC_NUM_OF_NUC_PAIRS_PSO	Number of nuclear pairs to calculate PSO terms
EPRNMR_SSC_NUM_OF_NUC_PAIRS_FC	Number of nuclear pairs to calculate FC terms
EPRNMR_SSC_NUM_OF_NUC_PAIRS_SD	Number of nuclear pairs to calculate SD terms
EPRNMR_SSC_NUM_OF_NUC_PAIRS_SD_FC	Number of nuclear pairs to calculate SD/FC terms
EPRNMR_SSC_NUM_OF_NUCLEI_PSO	Number of nuclei to calculate PSO perturbations
EPRNMR_SSC_NUM_OF_NUCLEI_FC	Number of nuclei to calculate SD/FC perturbations
SOC Energy Correction	
SOC_NUCLEAR_ENERGY	The nuclear energy
SOC_2C_ENERGY	The total 2-component energy
SOC_NON_SOC_ENERGY	The non-SOC total energy
SOC_ENERGY_CORRECTION	The SOC energy correction
Solvation	
SOLVATION_EPSILON	Dielectric constant
SOLVATION_REFRAC	Refractive index
SOLVATION_RSOLV	Solvent probe radius
SOLVATION_SURFACE_TYPE	Cavity surface
SOLVATION_CPCM_DIEL_ENERGY	Total energy including the CPCM dielectric correction
SOLVATION_NPOINTS	Number of points for the Gaussian surface

Table 8.2 – continued from previous page

SOLVATION_SURFACE_AREA	Surface area
General Job Information	
JOB_INFO_MULT	Job Multiplicity
JOB_INFO_CHARGE	Job Charge
JOB_INFO_NUM_OF_ATOMS	Total number of atoms
JOB_INFO_NUM_OF_EL	Total number of electrons
JOB_INFO_NUM_OF_FC_EL	Number of frozen core electrons
JOB_INFO_NUM_OF_CORR_ELC	Number of correlated electrons
JOB_INFO_NUM_OF_BASIS_FUNCS	Number of basis functions
JOB_INFO_NUM_OF_AUXC_BASIS_FUNCS	Number of auxilliary C basis functions
JOB_INFO_NUM_OF_AUXJK_BASIS_FUNCS	Number of auxilliary J basis functions
JOB_INFO_NUM_OF_AUX_CABS_BASIS_FUNCS	Number of auxilliary JK basis functions
JOB_INFO_NUM_OF_AUX_CABS_BASIS_FUNCS	Number of auxilliary CABS basis functions
JOB_INFO_TOTAL_EN	Final energy
HESSIAN	
HESSIAN_MODES	The hessian
Math Functions	
ABS	Absolute value
COS	Cosine
SIN	Sine
TAN	Tangent
ACOS	Inverse cosine
ASIN	Inverse sine
ATAN	Inverse tangent
COSH	Hyperbolic cosine
SINH	Hyperbolic sine
TANH	Hyperbolic tangent
EXP	Exponential
LOG	Common logarithm
LN	Natural logarithm
SQRT	Square root
ROUND	Round down to nearest integer

8.4 Compound Examples

8.4.1 Introduction

A library of compound scripts exist in page https://github.com/ORCAQuantumChemistry/CompoundScripts .

8.4.2 Hello World

Introduction

This is the simplest script that nevertheless points to an important feature of *Compound*. That is the fact that *Compound* does not have to run an actual 'normal' ORCA calculation but it can also be used as a driver for various tasks, in this case to just print a message.

Filename

helloWorld.inp

SCRIPT

```
%Compound
  print("Hellow World!\n");
EndRun
```

8.4.3 New Job

Introduction

One of the features of ORCA that will be deprecated in the future and should not be used any more is the *'New_Job'* feature. The current script is a simple example how *Compound* can be used to just run a series of calculations.

Filename

replaceNewJob.inp

SCRIPT

```
# This is a small script thas shows how
# 'Compound' can replace the previous
# ORCA '$New_Job' feature
%Compound
  # ------------------------------------
  # First job
  # ------------------------------------
  New_Step
    !BP86
    *xyz 0 1
      H 0.0 0.0 0.0
      H 0.0 0.0 0.8
    *
  Step_End
  # ------------------------------------
  # Second job with same goemetry
  # but different functional
  # ------------------------------------
  New_Step
    !B3LYP
    *xyz 0 1
      H 0.0 0.0 0.0
      H 0.0 0.0 0.8
    *
  Step_End
EndRun
```

Comments

From the *Compound* point of view the syntax in this script is not the most efficient one. It can be rewritten in more compact, cleaner, general way. Neverteless this is meant only as an exmample of how *Compound* can replace older ORCA calculations that used the, to be deprecated, *'New_Job'* feature.

8.4.4 High Accuracy

Introduction

This is a script that utilizes the scheme by N. J. DeYonker, T. R. Cundari, and A. K. Wilson published on: J. Chem. Phys. 124, 114104 (2006). The script calculates accurate total energies of molecules.

Filename

ccCA_CBS_2.cmp

SCRIPT

```
# This is a small script thas shows how
# 'Compound' can replace the previous
# ORCA '$New_Job' feature
%Compound
  # ------------------------------------
  # First job
  # ------------------------------------
  New_Step
    !BP86
    *xyz 0 1
      H 0.0 0.0 0.0
      H 0.0 0.0 0.8
    *
  Step_End
  # ------------------------------------
  # Second job with same goemetry
  # but different functional
  # ------------------------------------
  New_Step
    !B3LYP
    *xyz 0 1
      H 0.0 0.0 0.0
      H 0.0 0.0 0.8
    *
  Step_End
EndRun
```

Comments

It is interesting that in this scheme the total energy is treated and there is not separation in extrapolation between HF energy and correlation energy.

8.4.5 Scan

Introduction

This is an example script for a 1-Dimensional geometry scan. It is set up for the Ne-Ne bond distance but can be modified to suit the user's specific needs.

Filename

scan_1D_1M_1P.cmp

SCRIPT

```
# Author : Dimitrios G. Liakos
# Date   : May of 2024
#
# This is a script that will calculate and potentially
#   plot ONE property(1P) along a scan in ONE dimesion (1D)
#   using only ONE method (1M)
#
#   It is part of a series of scripts for different
#     combinations of scans for dimensions, methods,
#     and properties
#
# Here as an example we use for:
#    - dimension: the Ne-Ne bond (dist)
#    - method   : "HF" (method)
#    - property : the SCF energy (propName)
#
# The script creates a csv file with the absolute energies
#   and an additional one with the potential energies  in
#   kcal/mol. Both will be saved on disk.
#
# If 'DoPython' is set to true it will also create a python
#   script that plots the generated values and then run
#   it. The python script will be saved on disk and thus one
#   can afterwards manipulate it.
#
# NOTE The boolean option plotPotential will choose between
#      plotting absolute values or potential.
#
# NOTE The boolean obtion doKcal if set to true multiplies
#      the potential values with the HartreeToKcal factor.
#
# NOTE In case the doPython is set to true the script expects
#   that python3 is avaiable and also the following libraries:
#   - pandas
#   - seaborn
#   - matplotlib.pyplot
#
# -------------------------------------------------------------
#
# ----------------    Variables to change (e.g. through 'with')   -----------------
↪---------
Variable method       = "HF";           # The methods of the calculation
Variable basis        = "cc-pVDZ";      # The basis set of the calculation
Variable restOfInput  = "TightSCF";     # Maybe something common for the simple␣
↪input
Variable charge       = 0;              # Charge
Variable mult         = 1;              # Spin multiplicity
Variable myPropName   = "SCF_Energy";   # The properties we want to read
#
Variable lowerLimit   = 2.5;            # Lower limit value
Variable UpperLimit   = 5.0;            # Upper limit value
```

(continues on next page)

(continued from previous page)

```
Variable NSteps       = 13;              # Number of steps for the grid
Variable baseFilename = "myPotential";   # The basename for the created files
Variable plotPotential= true;            # Plot the potential instead of absolute␣
↪values
Variable DoKcal       = true;            # Multiply the potential values with the␣
↪HartreeToKcal factor
Variable removeFiles  = true;            # Remove *_Compound_*, *bas* files
# ------------------  python plot relevant variables ------------------------------
↪---------
Variable DoPython     = true;            # if we want python or not
Variable lw           = 4;               # The line width in case we plot with␣
↪python
Variable marker       = "o";             # The type of markers
Variable markerSize   = 10;              # The size of the markers in case we plot
Variable fontSize     = 18;
#
# -----------------------      Rest of the variables       ------------------------
↪---------
#
Variable HartreeToKcal = 627.5096080305927;                    # Hartree to kcal/
↪mol conversion factor
Variable stepSize      = (UpperLimit-LowerLimit)/(NSteps-1);  # The stepsize of␣
↪the grid
Variable calcValues[NSteps];                                  # An array to store␣
↪the calculated values
Variable  res, dist, calcValue;
Variable myFilename, csvFilename;
Variable fPtr;                                                # A file to write

# ---------------------------------------------------
#  Open and Write file header for the absolute values
# ---------------------------------------------------
write2String(csvFilename, "%s_absValues.csv", baseFilename);
fPtr = OpenFile(csvFilename, "w");
write2File(fptr, "distance,method,property,calcValue\n");

# ---------------------------------------------------
#  Perform the calculations and update the file
# ---------------------------------------------------
for iStep from 0 to NSteps-1 Do
  dist = lowerLimit + (iStep)*stepSize;
  New_Step
    !&{method} &{basis} &{restOfInput}
    *xyz &{charge} &{mult}
      Ne 0.0 0.0 0.0
      Ne 0.0 0.0 &{dist}
    *
    Step_end
    res = calcValue.readProperty(propertyName=myPropName);
    write2File(fPtr, "%.4lf,%20s,%20s,%20.10lf\n", dist, method,myPropName,␣
↪calcValue);
    calcValues[iStep]=calcValue;
EndFor
CloseFile(fPtr);   # Close the file

# ---------------------------------------------------
# Evaluate and write the relative values
# ---------------------------------------------------
write2String(csvFilename, "%s_relValues.csv", baseFilename);
fPtr = OpenFile(csvFilename, "w");
write2File(fPtr, "distance,method,property,calcValue\n");
```

(continues on next page)

(continued from previous page)

```
for iStep from 0 to NSteps-1 Do
  dist = lowerLimit + (iStep)*stepSize;
  if (DoKcal) then
    calcValue = (calcValues[iStep]-calcValues[NSteps-1])*HartreeToKcal;
  else
    calcValue = calcValues[iStep]-calcValues[NSteps-1];
  EndIf
  write2File(fPtr, "%.4lf,%20s,%20s,%20.10lf\n", dist, method,myPropName,␣
↪calcValue);
EndFor
CloseFile(fPtr);   # Close the file

if (removeFiles) then
  sys_cmd("rm *_Compound_* *.bas*");
EndIf

# ---------------------------------------------------
#  Create a python file and run it
# ---------------------------------------------------
if (DoPython) then
  if (plotPotential) then
    write2String(csvFilename, "%s_relValues.csv", baseFilename);
  else
    write2String(csvFilename, "%s_absValues.csv", baseFilename);
  endIf

  write2String(myFilename, "%s.py", baseFilename);
  fPtr = openFile(myFilename, "w");
  # Import necessary libraries
  write2File(fPtr, "import pandas as pd\n");
  write2File(fPtr, "import seaborn as sns\n");
  write2File(fPtr, "import matplotlib.pyplot as plt\n");
  # Read the csv file
  write2File(fPtr, "df = pd.read_csv('%s')\n", csvFilename);
  #Make a lineplot
  write2File(fPtr, "sns.lineplot(data=df, x=\"distance\", y=\"calcValue\", hue=\
↪"property\", \n
                     lw=%d, markers=True, marker='%s', markersize=%d,␣
↪dashes=False)\n", lw, marker, markersize);
  write2File(fPtr, "plt.axhline(y=0, color='black', linestyle='-', linewidth=1)\n
↪");
  write2File(fPtr, "plt.title(\"Energy Potential\", fontsize=%d)\n", fontsize+4);
  write2File(fPtr, "plt.xlabel(\"Ne-Ne Distance\", fontsize=%d)\n", fontsize);
  write2File(fPtr, "plt.ylabel(\"Energy (kcal/mol)\", fontsize=%d)\n", fontsize);
  write2File(fPtr, "plt.xticks(fontsize=%d)\n", fontSize);
  write2File(fPtr, "plt.yticks(fontsize=%d)\n", fontSize);
  write2File(fPtr, "plt.show()\n");
  closeFile(fPtr);
  sys_cmd("python3 %s", myFilename);
EndIf

End
```

Comments

This script has some interesing features. It contains two variables *removeFiles* and *DoPython*. If the first of them is set to *true* then the script will use a system command to remove files that are not needed anymore after the end of the calculation. The latter, *DoPython*, if set to *true* will read the *.csv* file that is created and write a *python* file to make a plot of the results. Then it will run the python script to actually make the plot.

8.4.6 Numerical polarizabilities

Introduction

This script calculates numerically the polarizability of the molecule using single point calculations with an electric field.

Filename

numericalPolarizability.cmp

SCRIPT

```
# Authors: Dimitrios G. Liakos / Frank Neese / Zikuan Wang
# Date   : May of 2024
#
# This is a compound script that calculates the
#  dipole-dipole polarizability tensor numerically
#  using the double derivative of energy.
#
# The idea is the following:
#
# 1 Perform a field free calculation
#
# 2 Loop over directions I=X,Y,Z
#
# 3 Loop over directions J=X,Y,Z
#
#    - put a small Q-field in directions I and J
#    - Solve equations to get the energy for each combination
#    - Polarizability alpha(I,J) =- ( E(+I,+J) - E(+I,-J)-E(-I,+J)+ E(-i,-j)/
↪(4*Field^2)
# 4 Print polarisability
#
# ----------------------------------------------------------------------
# ----------------------    Variables    --------------------------------
# --- Variables to be adjusted (e.g. using 'with' ----------------------
Variable molecule    = "h2o.xyz";
Variable charge      = 0;
Variable mult        = 1;
Variable method      = "HF";
Variable basis       = " ";
Variable restOfInput = "VeryTightSCF";
Variable blocksInput = " ";
Variable E_Field     = 0.0001;
Variable enPropName  = "JOB_Info_Total_En";
Variable removeFiles = true;
# -------------- Rest of the variables ---------------------------------
Variable FField[3];
Variable EFree, EPlusPlus, EPlusMinus, EMinusPlus, EMinusMinus, a[3][3];
Variable FFieldStringPlusPlus, FFieldStringPlusMinus;
Variable FFieldStringMinusPlus, FFieldStringMinusMinus;
Variable aEigenValues, aEigenVectors;

# -----------------------------------------
# Calculation without field
# -----------------------------------------
New_Step
  !&{method} &{basis} &{restOfInput}
  &{blocksInput}
  *xyzfile &{charge} &{mult} &{molecule}
Step_End
EFree.ReadProperty(propertyName=enPropName);
```

(continues on next page)

(continued from previous page)

```
# -----------------------------------------------------------------
# Loop over the x, y, z directions
# -----------------------------------------------------------------
for i from 0 to 2 Do
  for j from 0 to 2 Do
    # -----------------------------------------------------------
    # Create the appropriate direction oriented field string
    # -----------------------------------------------------------
    # ---------------------- (++) -----------------------------
    for k from 0 to 2 Do
      FField[k] = 0.0;
    EndFor
    FField[i] = FField[i] + E_Field;
    FField[j] = FField[j] + E_Field;
    write2String(FFieldStringPlusPlus,   " %lf,  %lf,  %lf",
    FField[0], FField[1], FField[2]);
    #
    # --------------------- (+-) -----------------------------
    for k from 0 to 2 Do
      FField[k] = 0.0;
    EndFor
    FField[i] = FField[i] + E_Field;
    FField[j] = FField[j] - E_Field;
    write2String(FFieldStringPlusMinus,   " %lf,  %lf,  %lf",
    FField[0], FField[1], FField[2]);
    #
    # --------------------- (-+) -----------------------------
    for k from 0 to 2 Do
      FField[k] = 0.0;
    EndFor
    FField[i] = FField[i] - E_Field;
    FField[j] = FField[j] + E_Field;
    write2String(FFieldStringMinusPlus,   " %lf,  %lf,  %lf",
    FField[0], FField[1], FField[2]);
    #
    # --------------------- (--) -----------------------------
    for k from 0 to 2 Do
      FField[k] = 0.0;
    EndFor
    FField[i] = FField[i] - E_Field;
    FField[j] = FField[j] - E_Field;
    write2String(FFieldStringMinusMinus,   " %lf,  %lf,  %lf",
    FField[0], FField[1], FField[2]);

    # ----------------------------------------
    # Perform the calculations.
    # The plus_plus (++) one
    # ----------------------------------------
    ReadMOs(1);
    New_Step
      !&{method} &{basis} &{restOfInput}
      %SCF
        EField = &{FFieldStringPlusPlus}
      End
      &{blocksInput}
    Step_End
    EPlusPlus.readProperty(propertyName=enPropName);
    # ----------------------------------------
    # The plus_minus (+-) one
    # ----------------------------------------
    ReadMOs(1);
```

(continues on next page)