CHAPTER

TWO

ESSENTIAL CALCULATION ELEMENTS

2.1 General Structure of the Input File

In general, the input file is a free format ASCII file and can contain:

- one or more *"simple" keyword lines* that start with a " ! " character,
- one or more *input blocks* enclosed between an "%" sign and "end" that provide finer control over specific aspects of the calculation,
- the *specification of the coordinates*, total charge, and spin multiplicity for the system, either with a %coords block, or more usually enclosed within two "*" symbols.

Here is an example of a simple input file that contains all three input elements:

```
! HF def2-TZVP

%scf
   convergence tight
end

* xyz 0 1
C   0.0   0.0   0.0
O   0.0   0.0   1.13
*
```

Comments in the file start by a "#". For example:

```
# This is a comment. Continues until the end of the line
```

Comments can also be closed by a second "#", as the example below where TolE and TolMaxP are two variables that can be user specified:

```
TolE=1e-5;    #Energy conv.#  TolMaxP=1e-6; #Density conv.#
```

The input may contain several blocks, which consist of logically related data that can be user controlled. The program tries to choose sensible default values for all of these variables. However, it is impossible to give defaults that are equally sensible for all systems. In general the defaults are slightly on the conservative side and more aggressive cutoffs etc. can be chosen by the user and may help to speed things up for actual systems or give higher accuracy if desired.

Note

- The ORCA input is **NOT** case sensitive. UPPER CASE, lower case, or aNy cOMmBINAtiON are allowed. An exception is file names (e.g. for %MOInp or *XYZName), which are case-sensitive on Unix-like OSs.
- In general, the order of the simple keywords and input blocks is not important – see *Input Priority and Processing Order* for the finer details.

2.1.1 Input Blocks

Input blocks start with "`%`", followed by the block name and end with "`end`". For example:

```
%method
  method HF
END
```

A list of available input blocks is given in Section 2.1.6. No blocks *need* to be present in an input file but they *can* be present if detailed control over the behavior of the program is desired. Otherwise, most jobs can be defined via the simple keywords described in Section 2.1.7. Variable assignments within blocks have the following general structure:

```
VariableName Value
# or with an optional "=" sign
OtherVariableName = OtherValue
```

Values can be either numeric, quote-delimited strings (see below), or predefined aliases (such as `HF` in the example above), which are internally converted to some numeric representation.

Some variables are actually arrays. In this case several possible assignments are useful:

```
Array[1]  Value1
Array[1]  Value1,Value2,Value3
Array  Value1,Value2
```

Note

Arrays always start with index 0 in ORCA (this is because ORCA is a C++ program). The first line in the example gives the value "`Value1`" to `Array[1]`, which is the *second* member of this array. The second line assigns `Value1` to `Array[1]`, `Value2` to `Array[2]` and `Value3` to `Array[3]`. The third line assigns `Value1` to `Array[0]` and `Value2` to `Array[1]`.

Strings (such as filenames) must be enclosed in quotes. For example:

```
%scf
  MOInp  "Myfile.gbw"
end
```

Note that file names on Unix-like systems are case-sensitive (i.e., `MYFILE.GBW` and `MyFile.gbw` are different files). Under Windows the file names are not case sensitive.

Some input block keywords either open a nested sub-block, which must be closed with an additional `end`, or have a specific syntax, different from the simple variable assignment described above. For example:

```
%scf
  Guess PModel  # variable assignment
  SOSCF  # nested sub-block
    start 0.002  # variable assignment
  end  # closes the SOSCF sub-block
end
%mdci
  # special syntax
  MP2FragInter {1 1} {2 2}
end
%basis
  NewGTO  # nested sub-block
    # special syntax inside
    H "def2-SVP"
    S 1
    1 0.05 1.0
```

(continues on next page)

(continued from previous page)

```
  end  # closes the NewGTO sub-block
end
```

Finally, there are input "blocks" which only set a single variable and are not closed with "`end`". These are listed in Table 2.2. For example:

```
%MOInp "MyFile.gbw"
%maxcore 3000
```

2.1.2 Input Priority and Processing Order

In more complicated calculations, the input can get quite involved. Therefore it is worth knowing how it is internally processed by the program:

- First, all the simple input lines (starting with "`!`") are collected into a single string.
- The program looks for all known keywords in a predefined order, regardless of the order in the input file.
- An exception are basis sets: if two different orbital basis sets (e.g. `! def2-SVP def2-TZVP`) are given, the latter takes priority. The same applies to auxiliary basis sets of the same type (e.g. `! def2/J SARC/J`).
- Some simple input keywords set multiple internal variables. Therefore, it is possible for one keyword to overwrite an option, set by another keyword. We have tried to resolve most such cases in a reasonable way (e.g. the more "specific" keyword should take precedence over a more "general" one) but it is difficult to foresee every combination of options.
- Next, the block input is parsed in the order it is given in the input file.
- Most block input keywords control a single variable (although there are exceptions). If a keyword is duplicated, the latter value is used.
- In principle, the same block may exist multiple times with different variables set within. However, some blocks "reset" certain internal data whenever they are opened, or have certain mandatory contents, which must be present in each block instance (and override previous instances). Therefore, it is not recommended to have multiple instances of the same block.

Consider the following (bad) example:

```
! def2-TZVP UKS
%method
  functional BP86
  correlation C_LYP
  SpecialGridAtoms[1]  26, 27
  SpecialGridIntacc 8,  8,  8
  SpecialGridAtoms 28, 29
end
! PBE def2-SVP RKS
```

Using the rules above, one can figure out why it is equivalent to this one:

```
! UKS BLYP def2-SVP
%method
  SpecialGridAtoms 28, 29, 27
  SpecialGridIntacc 8,  8,  8
end
```

2.1.3 Global Memory Use

Some ORCA modules (in particular those that perform some kind of wavefunction based correlation calculations) require large scratch arrays. Each module has an independent variable to control the size of these dominant scratch arrays. However, since these modules are never running simultaneously, we provide a global variable `MaxCore` that assigns a certain amount of scratch memory to all of these modules. Thus:

```
%MaxCore 2000
```

sets 2000 MB as the limit for these scratch arrays. **This limit applies per processing core**. Do not be surprised if the program takes more than that – this size only refers to the dominant work areas. Thus, you are well advised to provide a number that is no more than 75-80% of your physical memory. Some memory-hungry operations will take longer if given less than the required memory, while others will abort completely if `MaxCore` is insufficient. The default value is 4GB, which is plenty for most standard DFT calculations. For coupled clusters and the like, at least 8GB are recommended.

2.1.4 Changing the Default BaseName

ORCA generates a number of output files, as well as many temporary files, which are removed at the end of a successful run. To prevent filename clashes, all generated files start with the same prefix or *BaseName*. This is usually inferred from the name of the input file by removing the extension, i.e. running ORCA with `MyJob.inp` will create `MyJob.gbw`, `MyJob.properties.txt`, etc. It is also possible to set the BaseName explicitly using the `%base` variable. In the following example, the names of all generated files will start with `job1`, regardless of the name of the input file:

```
%base "job1"
```

2.1.5 Jobs with Multiple Steps

Warning

The `$new_job` feature is a deprecated function. Using `$new_job` might result in erratic results and strange behavior of succeeding calculations. Please use the *compound* feature of ORCA for tasks like this – it is safer and by far more powerful!

ORCA supports input files with multiple jobs. This feature is designed to simplify series of closely related calculations on the same molecule or calculations on different molecules. The objectives for implementing this feature include:

- Calculate of a molecular property using different theoretical methods and/or basis sets for one molecule.
- Calculations on a series of molecules with identical settings.
- Geometry optimization followed by more accurate single points and perhaps property calculations.
- Crude calculations to provide good starting orbitals that may then be used for subsequent calculations with larger basis sets.

For example consider the following job that in the first step computes the g-tensor of BO at the LDA level, and in the second step using the BP86 functional.

```
# ------------------------------------------------------
! LSD DEF2-SVP TightSCF KeepInts
# ------------------------------------------------------
%eprnmr gtensor 1 end

* int 0 2
   B  0  0  0   0      0  0
```

(continues on next page)

(continued from previous page)

```
   O  1  0  0   1.2049 0  0
*

# ***********************************************
# ****** This starts the input for the next job   *
# ***********************************************
$new_job
# ------------------------------------------------
! BP86 DEF2-SVP SmallPrint ReadInts NoKeepInts
# ------------------------------------------------
%eprnmr gtensor 1 end

* int 0 2
   B  0  0  0   0      0  0
   O  1  0  0   1.2049 0  0
*
```

What happens if you use the `$new_job` feature is that all calculation flags for the actual job are transferred from the previous job and that only the changes in the settings must be input by the user. Thus if you turn on some flags for one calculation that you do not want for the next, you have to turn them off again yourself (for example the use of the RI approximation)! In addition, the default is that the new job takes the orbitals from the old job as input. If you do not want this you have to overwrite this default by specifying your desired guess explicitly.

2.1.6 List of Input Blocks

Table 2.1 lists the known input block names, along with any accepted synonyms/aliases. The keywords defined in each block are listed in the respective section of the manual and references to these lists are also given in the table. The list of "blocks" which are not closed with `end` is given in Table 2.2.

Table 2.1: Input block keywords. Synonyms are given in parentheses.

Block	Description (Keyword Reference)
`autoci`	Autogenerated single- and multi-reference correlation methods (Section 3.11.9)
`basis`	Basis sets (Table 2.41)
`casresp`	CASSCF static linear response (Section 5.26.6)
`casscf`	CASSCF/NEVPT2 and DMRG calculations (Section 3.13.4)
`chelpg`	CHELPG charges (Section 5.1.8)
`cim`	Cluster-in-molecules calculations (Section 3.10.9)
`cis` (`tddft`)	CIS and TD-DFT calculations (Section 5.6.19)
`compound`	Compound jobs (Section 8.2)
`conical`	Optimization of conical intersections (Section 4.9)
`coords`	Input of atomic coordinates (Section 2.2)
`cosmors`	OpenCOSMO-RS options (Section 2.13.6)
`cpcm`	Conductor-like Polarizable Continuum Model (Section 2.13.8)
`docker`	Host-guest docking algorithm (Table 4.12)
`eda`	(Table 5.28)
`elprop`	Electric properties (Section 5.20.2)
`eprnmr`	EPR and NMR properties (Section 5.21.9; g-tensor: Table 5.13, HFC: Table 5.14, ZFS: Table 5.16, Mössbauer: Table 5.18)
`esd`	Excited state dynamics (Section 5.5.12)
`frag`	Automatic fragmentation procedure (Table 2.62)
`freq`	Vibrational frequencies (Section 4.7.4)

continues on next page

Table 2.1 – continued from previous page

Block	Description (Keyword Reference)
`geom`	Geometry optimization (Table 4.3; Scan: Table 4.4, TS: Table 4.5)
`goat`	Global optimization algorithm (Table 4.9)
`ice (iceci,cipsi)`	Iterative configuration expansion CI calculations (Section 3.14.11)
`irc`	Intrinsic reaction coordinate calculations (Section 4.4.2)
`lft`	Ligand field theory utility `orca_lft` (Section 9.2.15)
`loc`	Localization of orbitals (Section 9.2.5)
`mcrpa`	CASSCF linear response (Section 5.8.6)
`md`	Molecular dynamics (Table 7.1)
`mdci`	Single reference correlation methods (Section 3.10.14; RHF EOM-CC: Table 5.11, UHF EOM-CC: Table 5.12, STEOM-CC: Section 5.10.1, MR-EOM-CC: Section 3.20.4, LED: Section 5.38.7, ADLD/ADEX: Section 5.38.8, HFLD: Table 5.37)
`mecp`	Minimum energy crossing points optimization (Section 4.10.1)
`method`	Choice of computation method and various options • Run types and method classes: Table 2.5 • RI/COSX: Table 2.45 • Grids: Section 2.10.5 • CP-SCF: Section 2.22 • Frozen core: Section 2.21 • Population analysis: Table 5.3 • DFT: Table 3.3 • DFT/hybrid: Table 3.5 • DFT/range-separated: Table 3.7 • DFT/double-hybrid: Table 3.10 • DFT/LibXC: Section 3.3.6 • DFT/dispersion: Table 3.14 • DFT/NLC: Table 3.20 • gCP: Table 2.52 • NDO: Table 3.23 • Native xTB: Table 3.28 • FMM: Section 6.5.4
`mm`	Molecular mechanics force-fields (Section 3.23.5)
`mp2`	MP2 calculations (Section 3.9.1; DLPNO: Section 3.9.10, DLPNO/gradient: Section 3.9.11, DLPNO/response: Section 3.9.11, DLPNO/multi-level: Section 3.9.11, OO-RI-MP2: Section 3.9.14, regularized: Section 3.9.15)
`mrcc`	Multi-reference CC calculations (Section 3.12)
`mrci`	Multi-reference CI calculations (Section 3.19.15)
`mtr`	Normal mode trajectory/scan (Section 5.6.17)
`nbo`	NBO analysis (Section 5.2)
`ndoparas`	Parameters for NDO-based semi-empirical methods (Table 3.21)
`neb`	NEB calculations (Section 4.6.14)
`numgrad`	Numerical gradients (Section 2.23.2)
`output`	Control of output (Table 2.6)
`pal`	Parallel jobs (Section 2.5)
`paras`	Input of geometric parameters, equivalent to `paras` or `pardef` in `%coords` (Section 2.2.4)

continues on next page

Table 2.1 – continued from previous page

Block	Description (Keyword Reference)
`plots`	Plot generation (Section 9.1)
`qmmm`	Multiscale (QM/MM) calculations (Section 6.1)
`rel`	Relativistic options (Table 2.55; SOC: Section 5.28)
`rocis`	Restricted-open-shell CIS (Section 5.7.7)
`rr`	Resonance Raman and absorption/fluorescence band-shape calculations via `orca_asa` (Section 5.16.4)
`scf`	SCF procedure settings (Section 2.6; TRAH-SCF: Section 2.6.7, AVAS: Section 3.13.7, ROHF: Section 3.1.1, native xTB: Table 3.29, ΔSCF: Table 5.27, initial guess: Section 2.20, rotate MOs: Section 2.20.7)
`shark`	SHARK integral package (Section 2.15.6)
`solvator`	Explicit solvation algorithm (Table 4.10)
`symmetry(sym)`	Spatial symmetry recognition (Table 2.67)
`vpt2`	Vibrational perturbation theory (Section 5.19)
`xtb`	Options for the `xtb` program interface (Table 3.26)

Table 2.2: Input keywords, which are prefixed with `%` but have no closing `end`.

Keyword	Value	Description (Reference)
`base`	`"<BaseName>"`	Base name for the files created by the job (Section 2.1.4)
`cclib`	`"<FileName>"`	File with one-particle coupling coefficients for ICE-CI (Section 3.14.12)
`id`	`"<string>"`	**Deprecated!** Identifier for the job used to summarize computed energies (Section 2.1
`ljcoefficients`	`"<FileName>"`	File with Lennard–Jones coefficients for PHVA (Section 4.3.5)
`maxcore`	`4096`	Maximum heap memory to use in MB, default: 4GB (Section 2.1.3)
`moinp`	`"<FileName>"`	GBW file from which to read guess MOs for the `MORead` guess (Section 2.20.5)
`pointcharges`	`"<FileName>"`	File to read external point charges from (Section 2.2.6)

2.1.7 Simple Keyword Lines

It is possible to give a line of keywords that assign certain variables that normally belong to different input blocks. The syntax for this "simple input" is line-oriented. A keyword line starts with the "`!`" sign and can contain any number of space-separated keywords. The input file can contain any number of keyword lines and they do not need to be at the beginning (although that is common practice) – see also *Input Priority and Processing Order*.

```
! Keyword1 Keyword2
! Keyword3
```

Most simple input keywords are documented in the relevant section of the manual. Table 2.3 provides references to these sections, grouped by topic. Since simple keywords are usually related to variables within one or more input blocks, it is also instructive to consult the documentation for the respective block – see Table 2.1.

Table 2.3: References to the documentation of simple input keywords. Related input blocks are also listed.

Keyword Group	Reference	Input block
Run types and method classes	Table 2.4	`method`
Basis sets	Table 2.40	`basis`

continues on next page

Table 2.3 – continued from previous page

Keyword Group	Reference	Input block
DFT functionals	• LDA: Table 3.1 • GGA: Table 3.2 • global hybrid: Table 3.4 • range-separated hybrid: Table 3.6 • global double-hybrid: Table 3.8 • range-separated double hybrid: Table 3.9 • LibXC: Table 3.12 • Non-local correlation: Table 3.19	`method`
Dispersion corrections	Table 3.13	`method`
Composite (3c) methods	Table 3.32	
NDO-based semi-empirical methods	Table 3.27	`method`
Native xTB-based methods	Table 3.22	`method`
Second order Møller–Plesset perturbation theory	Table 3.34	`mp2`
Single-reference correlated methods via MDCI	Table 3.43	`mdci`
Correlated methods using automatic code generation (AUTOCI)	Section 3.11.2	`autoci`
CASSCF	Table 3.47	`casscf`
Multireference correlated methods via MRCI	Table 3.52	`mrci`
Excited states via correlated wavefunction-based methods	• EOM-CC: Table 5.10 • STEOM-CC: Section 5.10 • others: Section 5.4	`mdci`
Initial guess	Table 2.68	`scf`
SCF procedure	Table 2.11	`scf`
Integral approximations (RI/COSX)	Table 2.44	`method`
Relativistic methods	Table 2.54	`rel`
Implicit solvation (CPCM)	Table 2.56	`cpcm`
Spin-orbit coupling operator	Table 5.19	`rel`
Numerical integration grids	Table 2.48	`method`
Integral storage and handling	Table 2.58	`scf`
Population analysis	Table 5.1	`output`
Output control	Table 2.8	`output`
Nudged elastic band method	Section 4.6.14	`neb`

2.2 Input of Coordinates

Coordinates can be either specified directly in the input file or read from an external file, and they can be in either Cartesian ("xyz") or internal coordinate format ("Z-matrix").

2.2.1 Reading coordinates from the input file

The easiest way to specify coordinates in the input file is by including a block like the following, enclosed by star symbols:

```
* CType Charge Multiplicity
...
coordinate specifications
...
*
```

Here `CType` can be one of `xyz`, `int` (or `internal`), or `gzmt`, which correspond to Cartesian coordinates, internal coordinates, and internal coordinates in Gaussian Z-matrix format.

The input of Cartesian coordinates in the "`xyz`" option is straightforward. Each line consists of the label for a given atom type and three numbers that specify the coordinates of the atom. The units can be either Ångström (default) or Bohr. This can be specified via the simple keywords `!Angs` or `!Bohrs`, respectively, or via the variable `Units` in the `%coords` block described below.

```
* xyz Charge Multiplicity
Atom1   x1  y1  z1
Atom2   x2  y2  z2
 ...
*
```

For example for CO$^+$ in a $S = 1/2$ state (multiplicity = $2 \times 1/2 + 1 = 2$)

```
* xyz 1 2
C  0.0   0.0   0.0
O  0.0   0.0   1.1105
*
```

Internal coordinates are specified in the form of the familiar "Z-matrix". A Z-matrix basically contains information about molecular connectivity, bond lengths, bond angles and dihedral angles. The program then constructs Cartesian coordinates from this information. Both sets of coordinates are printed in the output such that conversion between formats is facilitated. The input in that case looks like:

```
* int Charge Multiplicity
Atom1  0  0  0    0.0   0.0   0.0
Atom2  1  0  0    R1    0.0   0.0
Atom3  1  2  0    R2    A1    0.0
Atom4  1  2  3    R3    A2    D1
. . .
AtomN  NA NB NC   RN    AN    DN
*
```

The rules for connectivity in the "`internal`" mode are as follows:

- `NA`: The atom that the actual atom has a distance (`RN`) with.
- `NB`: The actual atom has an angle (`AN`) with atoms `NA` and `NB`.
- `NC`: The actual atom has a dihedral angle (`DN`) with atoms `NA`, `NB` and `NC`. This is the angle between the actual atom and atom `NC` when looking down the `NA-NB` axis.
- Note that - contrary to other parts in ORCA - atoms are counted starting from 1.

Angles are always given in degrees! The rules are compatible with those used in the well known MOPAC and ADF programs.

Finally, `gzmt` specifies internal coordinates in the format used by the Gaussian program. This resembles the following:

```
* gzmt 0 1
    C
    O   1   4.454280
    Si  2   1.612138    1   56.446186
    O   3   1.652560    2   114.631525  1   -73.696925
    C   4   1.367361    3   123.895399  2   -110.635060
    ...
*
```

An alternative way to specify coordinates in the input file is through the use of the `%coords` block, which is organized as follows:

```
%coords
 CTyp   xyz     # the type of coordinates = xyz or internal
 Charge 0       # the total charge of the molecule
 Mult   2       # the multiplicity = 2S+1
 Units  Angs    # the unit of length = angs or bohrs

 # the subblock coords is for the actual coordinates
 # for CTyp=xyz
  coords
     Atom1   x1  y1  z1
     Atom2   x2  y2  z2
  end
 # for CTyp=internal
  coords
     Atom1  0  0  0    0.0   0.0   0.0
     Atom2  1  0  0    R1    0.0   0.0
     Atom3  1  2  0    R2    A1    0.0
     Atom4  1  2  3    R3    A2    D1
      . . .
     AtomN  NA NB NC   RN    AN    DN
  end
end
```

Important

Since ORCA is a C++ based program its internal counting starts from zero. Therefore all electrons, atoms, frequencies, orbitals, excitation energies etc. are counted from zero. User-based counting such as the numeration of fragments is counted from one.

2.2.2 Reading coordinates from external files

It is also possible to read the coordinates from external files. The most common format is a `.xyz` file, which can in principle contain more than one structure (see section *Multiple XYZ File Scans* for this multiple XYZ feature):

```
* xyzfile Charge Multiplicity Filename
```

For example:

```
* xyzfile 1 2 mycoords.xyz
```

A lot of graphical tools like Gabedit, molden or Jmol can write Gaussian Z-Matrices (`.gzmt`). ORCA can also read them from an external file with the following

```
* gzmtfile 1 2 mycoords.gzmt
```

Note that if multiple jobs are specified in the same input file then new jobs can read the coordinates from previous jobs. If no filename is given as fourth argument then the name of the actual job is automatically used.

```
... specification for the first job

$new_job
! keywords
* xyzfile 1 2
```

In this way, optimization and single point jobs can be very conveniently combined in a single, simple input file. Examples are provided in the following sections.

2.2.3 Special definitions

- **Dummy atoms** are defined in exactly the same way as any other atom, by using "DA", "X", or "Xx" as the atomic symbol.
- **Ghost atoms** are specified by adding ":" right after the symbol of the element (see *Counterpoise Corrections*).
- **Point charges** are specified with the symbol "Q", followed by the charge (see *Inclusion of Point Charges*).
- **Embedding potentials** are specified by adding a ">" right after the symbol of the element (see *ECP Embedding*).
- **Non-standard** isotopes or nuclear charges are specified with the statements "`M = ...`" and "`Z = ...`", respectively, after the atomic coordinate definition.

Note

1. The nuclear charge can adopt non-integer values
2. When the nuclear charge is modified throughca "`Z = ...`" statement, the total charge of the system should still be calculated based on the unmodified charge. For example, for a calculation of a single hydrogen atom whose `Z` is set to 1.5, a charge of 0 and a spin multiplicity of 2 should be entered into the charge and multiplicity sections of the input file, despite that the actual total charge is 0.5.

- **Fragments** can be conveniently defined by declaring the fragment number a given atom belongs to in parentheses "`(n)`" following the element symbol (see *Fragment Specification*).

2.2.4 Defining Geometry Parameters and Scanning Potential Energy Surfaces

ORCA lets you define the coordinates of all atoms as functions of user defined geometry parameters. By giving not only a value but a range of values (or a list of values) to this parameters potential energy surfaces can be scanned. In this case the variable `RunTyp` is automatically changed to `Scan`. The format for the parameter specification is straightforward:

```
%coords
 CTyp   internal
 Charge 0
 Mult   1
 pardef
  rCH  = 1.09;     # a C-H distance
  ACOH = 120.0;    # a C-O-H angle
  rCO  = 1.35, 1.10, 26; # a C-O distance that will be scanned
 end
 coords
```

(continues on next page)

(continued from previous page)

```
    C  0  0  0   0     0         0
    O  1  0  0  {rCO}  0         0
    H  1  2  0  {rCH}  {ACOH}    0
    H  1  2  3  {rCH}  {ACOH}  180
 end
end
```

In the example above the geometry of formaldehyde is defined in internal coordinates (the geometry functions work exactly the same way with Cartesian coordinates). Each geometric parameter can be assigned as a function of by enclosing an expression within function braces, "`{}` ". For example, a function may look like `*cos(Theta)*rML+R`. Note that all trigonometric functions expect their arguments to be in degrees and not radians. The geometry parameters are expected to be defined such that the lengths come out in Ångströms and the angles in degrees. *After* evaluating the functions, the coordinates will be converted to atomic units. In the example above, the variable `rCO` was defined as a "Scan parameter". Its value will be changed in 26 steps from 1.3 Å down to 1.1 Å and at each point a single point calculation will be done. At the end of the run the program will summarize the total energy at each point. This information can then be copied into the spreadsheet of a graphics program and the potential energy surface can be plotted. Up to three parameters can be scan parameters. In this way grids or cubes of energy (or property) values as a function of geometry can be constructed.

If you want to define a parameter at a series of values rather than evenly spaced intervals, the following syntax is to be used:

```
%coords
 CTyp   internal
 Charge 0
 Mult   1
 pardef
  rCH = 1.09;    # a C-H distance
  ACOH= 120.0;  # a C-O-H angle
  rCO [1.3 1.25 1.22 1.20 1.18 1.15 1.10];  # a C-O distance that will be scanned
 end
 coords
    C  0  0  0   0     0         0
    O  1  0  0  {rCO}  0         0
    H  1  2  0  {rCH}  {ACOH}    0
    H  1  2  3  {rCH}  {ACOH}  180
 end
end
```

In this example the C-O distance is changed in seven non-equidistant steps. This can be used in order to provide more points close to a minimum or maximum and fewer points at less interesting parts of the surface.

A special feature has also been implemented into ORCA - the parameters themselves can be made functions of the other parameters as in the following (nonsense) example:

```
%coords
 CTyp   internal
 Charge 0
 Mult   1
 pardef
  rCOHalf= 0.6;
  rCO    = { 2.0*rCOHalf };
 end
 coords
    C  0  0  0   0     0         0
    O  1  0  0  {rCO}  0         0
    O  1  0  0  {rCO}  180       0
 end
end
```

In this example the parameter `rCO` is computed from the parameter `rCOHalf`. In general the geometry is computed (assuming a `Scan` calculation) by: (a) incrementing the value of the parameter to be scanned (b) evaluating the functions that assign values to parameters, and (c) evaluating functions that assign values to geometrical variables.

Although it is not mandatory, it is good practice to *first* define the static or scan-parameters and then define the parameters that are functions of these parameters.

Finally, ORCA has some special features that may help to reduce the computational effort for surface scans:

```
%method
 SwitchToSOSCF true       # switches the converger to SOSCF
                          # after the first point. SOSCF may
                          # converge better than DIIS if the
                          # starting orbitals are good.
                          # default = false
 ReducePrint true         # reduce printout after the first point
                          # default=true

 # The initial guess can be changed after the first point.
 # The default is MORead. The MOs of the previous point will,
 # in many cases, be a very good guess for the next point.
 # However, in some cases you may want to be more conservative
 # and use a general guess.

 ScanGuess    OneElec    # the one-electron matrix
              Hueckel    # the extended Hueckel guess
              PAtom      # the PAtom guess
              PModel     # the PModel guess
              MORead     # MOs of the previous point
 end
```

Note

- You can scan along normal modes of a Hessian using the `NMScan` feature as described in section *Normal Mode Scan Calculations Between Different Structures*.
- The surface scan options are also supported in conjunction with TD-DFT/CIS or MR-CI calculations (see section *Potential Energy Surface Scans*).

2.2.5 Mixing internal and Cartesian coordinates

In some cases it may be practical to define some atomic positions in Cartesian and some in internal coordinates. This can be achieved by specifying all coordinates in the `*int` block: using “0 0 0” as reference atoms indicates Cartesian coordinates. Note that for the first atom the flags are “1 1 1”, as “0 0 0” would be the normal values for internal coordinates. Consider, for example, the relaxed surface scan from section *Theory*, where the methyl group is given first in an arbitrary Cartesian reference frame and then the water molecule is specified in internal coordinates:

```
! UKS B3LYP SV(P) TightSCF Opt SlowConv
%geom scan B 4 0 = 2.0, 1.0, 15 end end
* int 0 2
# First atom - reference atoms 1,1,1 mean Cartesian coordinates
  C      1   1   1    -0.865590    1.240463   -2.026957

# Next atoms - reference atoms 0,0,0 mean Cartesian coordinates
  H      0   0   0    -1.141534    2.296757   -1.931942
  H      0   0   0    -1.135059    0.703085   -2.943344
  H      0   0   0    -0.607842    0.670110   -1.127819

# Actual internal coordinates
  H      1   2   3     1.999962   100.445    96.050
  O      5   1   2     0.984205   164.404    27.073
```

(continues on next page)

(continued from previous page)

```
  H       6    5    1      0.972562    103.807     10.843
*
```

Internal and Cartesian coordinates can thus be mixed in any order but it is recommended that the first 3 atoms are specified in Cartesian coordinates in order to define a unique reference frame.

2.2.6 Inclusion of Point Charges

In some situations it is desirable to add point charges to the system. In ORCA there are two mechanisms to add point-charges. If you only want to add a few point charges you can "mask" them as atoms as in the following (nonsense) input:

```
# A water dimer
! BP86 def2-SVP

* xyz 0 1
O           1.4190    0.0000    0.0597
H           1.6119    0.0000   -0.8763
H           0.4450    0.0000    0.0898
Q  -0.834  -1.3130    0.0000   -0.0310
Q   0.417  -1.8700    0.7570    0.1651
Q   0.417  -1.8700   -0.7570    0.1651
*
```

Here the "Q"'s define the atoms as point charges. The next four numbers are the magnitude of the point charge and its position. The program will then treat the point charges as atoms with no basis functions and nuclear charges equal to the "Q" values.

If you have thousands of point charges to treat, as in a QM/MM calculation, it is more convenient, and actually necessary, to read the point charges from an external file as in the following example:

```
# A water dimer
! BP86 def2-SVP

% pointcharges "pointcharges.pc"

* xyz 0 1
O    1.4190    0.0000    0.0597
H    1.6119    0.0000   -0.8763
H    0.4450    0.0000    0.0898
*
```

The program will now read the file "`pointcharges.pc`" that contains the point-charge information and then call the module `orca_pc` which adds the point charge contribution to the one-electron matrix and the nuclear repulsion. The file "`pointcharges.pc`" is a simple ASCII file in the following format:

```
3
 -0.834  -1.3130    0.0000   -0.0310
  0.417  -1.8700    0.7570    0.1651
  0.417  -1.8700   -0.7570    0.1651
```

The first line gives the number of point charges. Each consecutive line gives the magnitude of the point charge (in atomic units) and its position (in Ångström units!). However, it should be noted that ORCA treats point charges from an external file differently than "Q" atoms. When using an external point charge file, the interaction between the point charges is not included in the nuclear energy. This behavior originates from QM/MM, where the interactions among the point charges is done by the MM program. These programs typically use an external point charge file when generating the ORCA input. To add the interaction of the point charges to the nuclear energy, the `DoEQ` keyword is used either in the simple input or the `%method` block as shown below.

```
# A non QM/MM pointcharge calculation
! DoEQ

%pointcharges "pointcharges.pc"

%method
   DoEQ true
end
```

Warning

The `DoEQ` keyword has no effect for semi-empirical calculations (e.g., AM1)!

2.3 Basic Calculation Settings

2.3.1 Run Types

The type of calculation to be performed can be chosen via the simple input – for example, `! Opt` – or via the `RunTyp` variable in `%method`:

```
%method
  RunTyp  Gradient  # Single point energy and gradient
end
```

The full list of available run types is given in Section 2.3.3. Note that some run types are triggered automatically under certain conditions and cannot be requested explicitly via the `RunTyp` keyword.

2.3.2 Method Classes

ORCA provides several classes of methods: Hartree–Fock (HF), Density functional theory (DFT), semi-empirical methods based on the NDDO approach, etc. This choice is controlled via the simple input, e.g. `!HF`, or via the `method` variable in the `%method` block:

```
%method
  method DFT
end
```

The list of available methods is given in Section 2.3.3.

2.3.3 Keywords

Table 2.4: Simple input keywords for basic calculation settings.

Keyword	Description
Run Types	
`Energy`	Single point energy calculation (Default. Aliases: `SinglePoint`, `SP`)
`EnGrad`	Single point energy and gradient calculation (Alias: `EnergyGrad`)
`Opt`	*Geometry optimization*, see also Table 4.1
`MD`	*Molecular dynamics simulation*
`CIM`	*Cluster-in-molecules* calculation
`PrintThermoChem`	*Thermochemistry calculation* from previously calculated Hessian
`PropertiesOnly`	*Compute properties* from previously calculated densities
`Docker`	*Host-guest docking*
`EDA`	*Energy decomposition analysis*
`NMScan`	**Deprecated!** Normal mode scan (Alias: `NormalModeScan`)
`NMGrad`	**Deprecated!** Normal mode gradient (Alias: `NMGradient`)
`MTR`	**Deprecated!** Mode trajectory (Aliases: `MT`, `ModeTrajectory`)
Methods	
`HF`	*Hartree–Fock*; `RHF`, `UHF`, and `ROHF` also set the *wavefunction type*
`DFT`	*Density functional theory*; `RKS`, `UKS`, and `ROKS` also set the *wavefunction type*
`<NDO-Method>`	*NDO*-based method keywords are given in Table 3.22
`<XTB-Method>`	*Native xTB*-based method keywords are given in Table 3.27
`MM`	*Molecular mechanics*

Table 2.5: `%method` block input keywords for basic calculation settings.

Keyword	Options	Description
`RunTyp`	`Energy`	Single point energy calculation (Default. Aliases: `SinglePoint`, `SP`)
	`Gradient`	Single point energy and gradient calculation
	`Opt`	*Geometry optimization* (Aliases: `Geom`, `Geometry`, `GeometryOpt`)
	`Scan`	*Unrelaxed geometry parameter scan* (Alias: `Trajectory`)
	`CIM`	*Cluster-in-molecules* calculation
	`NMScan`	**Deprecated!** Normal mode scan (Alias: `NormalModeScan`)
	`NMGrad`	**Deprecated!** Normal mode gradient (Aliases: `NMGradient`, `NormalModeGradient`)
	`MTR`	**Deprecated!** Mode trajectory (Aliases: `MT`, `ModeTrajectory`)
`Method`	`HF`	(Default) *Hartree–Fock*
	`DFT`	*Density functional theory*
	`<NDO-Method>`	*NDO*-based method options are given in Table 3.23

2.4 Control of Output

ORCA provides various options to control the amount of printed output. This is specifically important as some methods and protocols can create very large amount of output and data. General control of the output is provided by the `PrintLevel` and `Print[<option>]` keywords in the `%output` block: `PrintLevel` can be used to select certain default settings for the print array. Specifying `Print` after `PrintLevel` can be used to modify these defaults.

```
%output
  PrintLevel      Normal
  Print[ Flag ]   0  # turn print off
                  1  # turn print on
                  n  # some flags are more sophisticated
end
```

2.4.1 Print Options

A detailed list of options for `PrintLevel` and `Print` is given in table Table 2.6. A list of option toggled by each printlevel can be found in Table 2.7.

Table 2.6: `%output` block input keywords and options for print control.

Keyword	Option	Description
`PrintLevel`	`nothing`	Deactivates output printing
	`mini`	Sets printlevel to mini
	`small`	Sets printlevel to small
	`normal`	Sets printlevel to normal (default)
	`maxi`	Sets printlevel to maxi
	`large`	Sets printlevel to large
	`huge`	Sets printlevel to huge
	`debug`	Sets printlevel to debug
`Print[ <option> ]`	`P_InputFile`	Echo the input file
	`P_Cartesian`	Print the cartesian coordinates
	`P_Internal`	Print the internal coordinates
	`P_Basis`	$= 1$: Print the basis set information
		$= 2$: Also print the primitives in input format
	`P_OneElec`	Print of the one electron matrix
	`P_Overlap`	Print the overlap matrix
	`P_KinEn`	Print the kinetic energy matrix
	`P_S12`	Print the $S^{-1/2}$ matrix
	`P_GuessOrb`	Print the initial guess orbitals
	`P_OrbEn`	$= 1$: Print orbital energies up to LUMO+9
		$= 2$: Print all orbital energies
	`P_MOs`	Print the MO coefficients on convergence
	`P_Density`	Print the converged electron density
	`P_SpinDensity`	Print the converged spin density
	`P_EHTDetails`	Print initial guess extended Hückel details
	`P_SCFInfo`	Print the SCF input flags
	`P_SCFMemInfo`	Print the estimated SCF memory requirements
	`P_SCFIterInfo`	$= 1$: Print short iteration information
		$= 2$: Print longer iteration information
		$= 3$: In a direct SCF also print integral progress
	`P_Fockian`	Print Fockian matrix
	`P_DIISMat`	Print DIIS matrix
	`P_DIISError`	Print DIIS error
	`P_Iter_P`	Print Density
	`P_Iter_C`	Print MO coefficients
	`P_Iter_F`	Print Fock matrix
	`P_Mayer`	Print Mayer population analysis (Default = on)
	`P_NatPop`	Print Natural population analysis (Default = off)
	`P_NPA`	Print Natural population analysis (Default = off)
	`P_Hirshfeld`	Print Hirshfeld population analysis (Default = off)
	`P_MBIS`	Print MBIS population analysis (Default = off)
	`P_Mulliken`	Print Mulliken population analysis (Default = on)
	`P_AtCharges_M`	Print Mulliken atomic charges
	`P_OrbCharges_M`	Print Mulliken orbital charges
	`P_FragCharges_M`	Print Mulliken fragment charges
	`P_FragBondOrder_M`	Print Mulliken fragment bond orders
	`P_BondOrder_M`	Print Mulliken bond orders
	`P_ReducedOrbPop_M`	Print Mulliken reduced orbital charges
	`P_FragPopMO_M`	Print Mulliken fragment population for each MO
	`P_FragOvlMO_M`	Print Mulliken overlap populations per fragment pair

continues on n

Table 2.6 – continued from previous page

Keyword	Option	Description
	P_AtPopMO_M	Print Mulliken atomic charges in each MO
	P_OrbPopMO_M	Print Mulliken orbital population for each MO
	P_ReducedOrbPopMO_M	Print Mulliken reduced orbital population for each MO
	P_Loewdin	Print Loewdin population analysis (Default = on)
	P_AtCharges_L	Print Loewdin atomic charges
	P_OrbCharges_L	Print Loewdin orbital charges
	P_FragCharges_L	Print Loewdin fragment charges
	P_FragBondOrder_L	Print Loewdin fragment bond orders
	P_BondOrder_L	Print Loewdin bond orders
	P_ReducedOrbPop_L	Print Loewdin reduced orbital charges
	P_FragPopMO_L	Print Loewdin fragment population for each MO
	P_FragOvlMO_L	Print Loewdin overlap populations per fragment pair
	P_AtPopMO_L	Print Loewdin atomic charges in each MO
	P_OrbPopMO_L	Print Loewdin orbital population for each MO
	P_ReducedOrbPopMO_L	Print Loewdin reduced orbital population for each MO
	P_NPA	Natural population analysis
	P_Fragments	Print fragment information
	P_GUESSPOP	Print initial guess populations
	P_UNO_FragPopMO_M	Print Mulliken fragment population per UNO
	P_UNO_OrbPopMO_M	Print Mulliken orbital pop. per UNO
	P_UNO_AtPopMO_M	Print Mulliken atomic charges per UNO
	P_UNO_ReducedOrbPopMO_M	Print Mulliken reduced orbital pop. per UNO
	P_UNO_FragPopMO_L	Print Loewdin fragment population per UNO
	P_UNO_OrbPopMO_L	Print Loewdin orbital pop. per UNO
	P_UNO_AtPopMO_L	Print Loewdin atomic charges per UNO
	P_UNO_ReducedOrbPopMO_L	Print Loewdin reduced orbital pop. per UNO
	P_UNO_OccNum	Print occupation numbers per UNO
	P_AtomExpVal	Print atomic expectation values
	P_AtomBasis	Print atomic basis
	P_AtomDensFit	Print electron density fit
	P_Symmetry	Symmetry basic information
	P_Sym_Salc	Symmetry process printing
	P_SCFSTABANA	Information on progress, convergence, and results of SCF stabilit
	P_DFTD	Print info on Grimme's dispersion correction
	P_DFTD_GRAD	Print gradient info on Grimme's dispersion correction
	P_G1EL2EL	Print one- and two-electron contributions to g-tensor

The various choices for PrintLevel have the following defaults:

Table 2.7: Default print options invoked by the respective `PrintLevel`.

PrintLevel	Print options included	
mini	P_OrbEn	= 1
	P_Cartesian	= 1
	P_InputFile	= 1
	P_SCFIterInfo	= 1
small	all the previous plus	
	P_SCFInfo	= 1
	P_Mayer	= 1
	P_MULLIKEN	= 1
	P_AtCharges_M	= 1
	P_ReducedOrbPop_M	= 1
	P_Loewdin	= 1
	P_AtCharges_L	= 1

continues on next page

Table 2.7 – continued from previous page

PrintLevel	Print options included
	`P_ReducedOrbPop_L = 1`
	`P_Fragments = 1`
	`P_FragCharges_M = 1`
	`P_FragBondOrder_M = 1`
	`P_FragCharges_L = 1`
	`P_FragBondOrder_L = 1`
	`P_DFTD = 1`
`normal`	all the previous plus
	`P_Internal = 1`
	`P_BondOrder_L = 1`
	`P_BondOrder_M = 1`
	`P_FragPopMO_L = 1`
	`P_ReducedOrbPopMO_L = 1`
	`P_SCFIterInfo = 2`
`maxi`	all the previous plus
	`P_OrbEn = 2`
	`P_GuessOrb = 1`
	`P_MOs = 1`
	`P_Density = 1`
	`P_SpinDensity = 1`
	`P_Basis = 1`
	`P_FragOVLMO_M = 1`
	`P_OrbPopMO_M = 1`
	`P_OrbCharges_M = 1`
	`P_DFTD = 2`
	`P_DFTD_GRAD = 1`
`huge`	All the previous plus
	`P_OneElec = 1`
	`P_Overlap = 1`
	`P_S12 = 1`
	`P_AtPopMO_M = 1`
	`P_OrbPopMO_M = 1`
	`P_AtPopMO_L = 1`
	`P_EHTDetails = 1`
	`P_DFTD_GRAD = 2`
`debug`	Prints everything

2.4.2 Simple Input Keywords

For convenience, a variety of simple input keywords that control specific print settings and additional outputs have been implemented as well (cf. Table 2.8).

Table 2.8: Simple input keywords for specific output control.

Keyword	Print options toggled	Description
`MiniPrint`	`PrintLevel mini`	Activates printlevel mini
`SmallPrint`	`PrintLevel small`	Activates printlevel small (default)
`NormalPrint`	`PrintLevel normal`	Activates printlevel normal
`LargePrint`	`PrintLevel large`	Activates printlevel mini
`PrintMOs`	`Print[ p_MOs ]`	Prints MO coefficients
`NoPrintMOs`		Suppress printing of MO coefficients
`PrintBasis`	`Print[ p_basis ]`	Print the basis set in input format
`PrintGap`	`Print[ p_homolumogap ]`	Prints the HOMO/LUMO gap in each SCF iteration.
`ReducedPop`		Prints Loewdin reduced orb.pop per MO
`NoReducedPop`		Deactivates printing of Loewdin reduced orb.pop per MO
`AIM`		Produce a WFN file
`XYZFILE`		Produce an XYZ coordinate file
`PDBFILE`		Produce a PDB file
`UNO`		Produce a *UHF natural orbitals*
`NoPropFile`		Do not write to the *property file*
`KeepTransDensity`		Keep the transition density matrices on disk

2.5 Parallel and Multi-Process Runs

Most of the important modules in ORCA can run in parallel or in multi-process mode: There are parallel versions for Linux, MAC and Windows computers which make use of OpenMPI (open-source MPI implementation) and Microsoft MPI (Windows only). Parallel execution means that the different processes perform the task in synchronous manner, communicating results and synchronizing execution (via MPI). The multi-process mode also employs multiple processes. But these work independently, not knowing - and not needing to know what the other processes are doing. The latest ORCA version even can combine both modes. Please see the remarks in *Multi-Process Calculations* for more details.

Parallel (or multi-process) execution is requested in the input via

```
! PAL4  # everything from PAL2 to PAL8 and Pal16, Pal32, Pal64 is recognized
```

or

```
%pal nprocs 4 end # any number (positive integer)
```

Assuming that the MPI libraries are properly installed on your computer, it is fairly easy to run the parallel version of ORCA: You simply specify the number of parallel processes in the input and call (serial) ORCA (with full path!) The parallelized modules of ORCA are started by the (serial) ORCA-Driver. If the driver finds `PAL4` or `%pal nprocs 4 end` (e.g.) in the input, it will start up the parallel modules instead of the serial ones.

Warning

Do not start the ORCA driver with mpirun!