

ORCA 5.0 Molecular Dynamics Module

This is a subset of the full ORCA manual
which contains only the description
of the molecular dynamics module.

Martin Brehm,
2016 – 2021

<https://brehm-research.de/orcamd>

9.40 Ab initio Molecular Dynamics Simulations

A few years ago, we included an *ab initio* molecular dynamics (AIMD) module into ORCA.¹ As a plethora of different electron structure methods with analytical gradients is already implemented, all these methods are now available also for MD simulations, offering a wide range of accuracy/performance trade-offs.

Despite the relatively short history inside of the ORCA package, the MD module has grown considerably over the last years. A few features found in other MD codes are still missing. In future releases, many more new features and methods will (hopefully) be added to this part of the program. We will always do our best to keep a strict backward compatibility, such that the sample inputs from this section will remain valid in all future releases.

For some more information as well as input examples for the ORCA MD module, please visit

<https://brehm-research.de/orcamd>

9.40.1 Changes in ORCA 5.0

- Added a **Metadynamics** module with many features and options:
 - Can perform one-dimensional and two-dimensional Metadynamics simulations [1] to explore free energy profiles along reaction coordinates, called collective variables (“Colvars”).
 - **Colvars** can be distances (*including projections onto vectors and into planes*), angles, dihedrals, and coordination numbers [2]. The latter allows, *e. g.*, to accurately compute pK_A values of weak acids [3, 4].
 - For all Colvars, groups of atoms (*e. g.*, *centers of mass*) can be used instead of single atoms.
 - Metadynamics simulations can be easily restarted and split over multiple runs.
 - Ability to run well-tempered Metadynamics [5] for a smoothly converging free energy profile.
 - Ability to run extended Langrangian Metadynamics [2], where a virtual particle on the bias profile is coupled to the real system via a spring. The virtual particle can be thermostated.
- Added two modern and powerful **thermostats** (*both available as global and massive*):
 - The widely used Nosé–Hoover chain thermostat (NHC) with high-order Yoshida integrator [6, 7]; allows for a very accurate sampling of the canonical ensemble.
 - The stochastical “Canonical Sampling through Velocity Rescaling” (CSVR) thermostat [8] which has become quite popular recently.
- Can define harmonic and Gaussian **restraints** for all Colvars (*distance, angle, dihedral, coordination number*). This allows for umbrella sampling [9, 10], among other methods. Can also define one-sided restraints which act as lower or upper wall.
- Can now print the instantaneous and average force on **constraints** and **restraints**; this allows for thermodynamic integration [10].
- The target value for **constraints** and **restraints** can now be a ramp, so that it can linearly change during the simulation.
- Can now keep the system’s center of mass fixed during MD runs.
- Can now print population analyses, orbital energies, and .engrad files in every MD step if requested.

¹Strictly speaking, these simulations are Born–Oppenheimer molecular dynamics simulations (BOMD), because they approximately solve the time-independent Schrödinger equation to compute gradients and then move the atoms according to these gradients.

Changes in ORCA 4.2 (Aug 2019)

- Added a Cartesian minimization command to the MD module, based on L-BFGS and simulated annealing. Works for large systems (> 10 000) atoms and also with constraints. Offers a flag to only optimize hydrogen atom positions (*for crystal structure refinement*). See [Minimize](#) command.
- The MD module can now write trajectories in DCD file format (*in addition to the already implemented XYZ and PDB formats*), see [Dump](#) command.
- The [thermostat](#) is now able to apply temperature ramps during simulation runs.
- Added more flexibility to region definition (*can now add/remove atoms to/from existing regions*). Renamed the [Define_Region](#) command to [Manage_Region](#).
- Added two new constraint types which keeps centers of mass fixed or keep complete groups of atoms rigid, see [Constraint](#) command.
- Ability to store the GBW file every *n*-th step during MD runs (*e.g. for plotting orbitals along the trajectory*), see [Dump](#) command.
- Can now set limit for maximum displacement of any atom in a MD step, which can stabilize dynamics with poor initial structures. Runs can be cleanly aborted by “[touch EXIT](#)”. See [Run](#) command.
- Better handling/reporting of non-converged SCF during MD runs.
- Fixed an issue which slowed down molecular dynamics after many steps.
- Stefan Grimme’s xTB method can now be used in the MD module, allowing fast simulations of large systems.

Changes in ORCA 4.1 (Dec 2018)

- Molecular dynamics simulation can now use Cartesian, distance, angle, and dihedral angle constraints. These are managed with the [Constraint](#) command.
- The MD module now features cells of several geometries (cube, orthorhombic, parallelepiped, sphere, ellipsoid), which can help to keep the system inside of a well-defined volume. The cells have repulsive harmonic walls.
- The cells can be defined as elastic, such that their size adapts to the system. This enables to run simulations under constant pressure.
- Trajectories can now be written in XYZ and PDB file format.
- A restart file is written in each simulation step. With this file, simulations can be restarted to seamlessly continue (useful for batch runs or if the job crashed). Restart is handled via the [Restart](#) command; see below.
- Introduced regions (*i. e.*, subsets of atoms), which can be individually defined. Regions can be used to thermostat different parts of the system to different temperatures (*e. g.*, cold solute in hot solvent), or to write subset trajectories of selected atoms.
- The energy drift of the simulation is now displayed in every step (in units of Kelvin per atom). Large energy drift can be caused by poor SCF convergence, or by a time step length chosen too large.
- Physical units in the MD input are now connected to their numeric values via underscore, such as [350_pm](#). A whitespace between value and unit is no longer acceptable. This slightly breaks backward compatibility – sorry.
- Fixed a bug in the time integration of the equations of motion, which compromised energy conservation.
- Fixed crashes for semiempirics and if ECPs were employed. You can now run MD simulations with methods such as PM3 and with ECPs.

9.40.2 Input Format

The molecular dynamics module is activated by specifying “MD” in the simple input line. The actual MD input which describes the simulation follows in the “%md” section at some later position in the input file. The contents of this section will subsequently be referred to as “MD input”.

```
! MD BLYP D3 def2-SVP
%md
  Timestep 0.5_fs # This is a comment
  Initvel 350_K
  Thermostat NHC 350_K Timecon 10.0_fs
  Dump Position Stride 1 Filename "trajectory.xyz"
  Run 200
end
* xyz 0 1
O      -4.54021      0.78439      0.09307
H      -3.64059      0.38224     -0.01432
H      -4.63463      1.39665     -0.67880
*
```

Please note that the MD input is not processed by ORCA’s main parser, but by a dedicated parser in the MD module. Therefore, the MD input is not required to obey the general ORCA syntax rules. The syntax will be described in the following.

In contrast to general ORCA input, the MD input is not based on keywords, but on *commands*, which are executed consecutively on a line-by-line basis starting at the top (like, *e.g.*, in a shell script). This means that identical commands with different arguments may be given, coming into effect when the interpreter reaches the corresponding line. This enables to perform multiple simulations (*e.g.*, pre-equilibration and production run) within a single input file:

```
%md
  Timestep 1.0_fs
  Run 200
  Timestep 2.0_fs
  Run 500
end
```

Work is already under way to add variable definitions, loops, and conditional branching to the MD input.² This will enable even larger flexibility (*e.g.*, to run a simulation until a certain quantity has converged). The MD input is written in the SANscript language (“Scientific Algorithm Notation Script”), which is under development. A first glimpse can be found at

<https://brehm-research.de/sanscript>

²Technically speaking, ORCA will then be a Turing-complete script interpreter, such that *any* computational problem can be solved with ORCA :-)

As in standard ORCA input, **comments** in the MD input are initiated by a “#” sign and span to the end of the current line. Commands can be started both at the beginning of a line and after a command. The only place where a “#” is **not** treated as start of a comment is inside of a string literal (*e. g.*, in file names).

```
%md
  # Comment
  Timestep 0.5_fs # Comment
  Dump Position Filename "trajectory#1.xyz"
end
```

Some more MD input syntax rules:

- The MD input is generally **not** case-sensitive. The only exception are file names on platforms with case-sensitive file systems (such as GNU Linux).
- Empty lines are allowed.
- Commands and options are separated by space or tabulator characters. Any combination of these characters may be used as separator.
- Both DOS and UNIX line break style is acceptable.

Commands

As already noted above, the central item of the MD input is a command. Each input line contains (at most) one command, and these commands are executed in the given order. A command typically takes one or more arguments, which are given behind the command name, separated by whitespaces, tabulator characters, or commas (optional). The order of the arguments for a command is fixed (see command list in section 9.40.4). Commands may have optional arguments, which are always specified at the end of the argument list, after the last non-optional argument. If there exist multiple optional arguments for a command, not all of them need to be specified; however, they need to be specified in the correct order and without gaps:

```
%md
  Command Arg1 Arg2 Arg3           # fine
  Command Arg1, Arg2, Arg3         # fine
  Command Arg1 Arg2 Arg3 Optarg1   # fine
  Command Arg1 Arg2 Arg3 Optarg1 Optarg2 # fine
  Command Arg1 Arg2 Arg3 Optarg2   # will not work
end
```

Apart from arguments and optional arguments, commands can also have *modifiers*. These can be considered as “sub-commands”, which modify a given command, and may possess their own argument lists. Modifiers generally follow after all non-optional and optional arguments, and they may **not** possess optional arguments on their own. If a command has multiple modifiers, the order in which they are given is not important.

In the following input example, “Mod1” and “Mod2” are modifiers of “Command”. “Mod1” takes one argument, “Mod2” does not take arguments:

```
%md
  Command Arg1                               # fine
  Command Arg1 Optarg1                       # fine
  Command Arg1 Mod1 Modarg1 Mod2         # fine
  Command Arg1 Mod2                       # fine
  Command Arg1 Mod2 Mod1 Modarg1       # fine
  Command Arg1 Optarg1 Mod1 Modarg1 Mod2 # fine
end
```

To make this abstract definition a little more illustrative, please consider again one line from the input sample at the beginning of this section:

```
%md
  Dump Position Stride 1 Filename "trajectory.xyz"
end
```

Here, “Dump” is the command, which takes one non-optional argument to specify which quantity shall be dumped – in this case, “Position”. The “Dump” command has two modifiers, namely “Stride” and “Filename”. The former takes one integer argument, the latter a string argument. Swapping the two modifiers (together with their respective arguments, of course) would not change the behavior.

Separating Arguments

As shown above, the arguments which are passed to a command do not need to be separated by commas. However, it is allowed (and recommended) to still use commas. First, it can increase the readability of the input file. Secondly, there exist a few ambiguous cases in which commas (or parentheses) should be used to clarify the intended meaning. One of these cases is the arithmetic minus operator. It can either be used as binary operator (subtracting one number from another), or as unary operator (returning the negative of a number). By default, the minus operator will be considered as binary operator (if possible).

Consider the case in which you want to pass two integer arguments “10” and “-10” to a command. Without commas (or parentheses), the minus is mistreated as binary operator, and only one argument will be passed to the command:

```
Command 10 -10 # Pitfall: treated as "Command (10 - 10)", i. e., "Command 0"
Command 10, -10 # Two arguments, as intended
Command 10 (-10) # Also works
```

Physical Units

In many cases, it is required to specify quantities which bear a physical unit in an input file (*e. g.*, temperature, time step lengths, ...). For many quantities, there are different units in widespread use, which always leads to some confusion (just consider the “kcal vs kJ“ case). ORCA handles this problem by defining default units for each quantity and requiring that all quantities are given in their default unit. ORCA’s default units are the atomic units, which are heavily used in the quantum chemistry community, but not so much in the molecular dynamics community. As an *ab initio* molecular dynamics module exists in the small overlap region of both communities, some “unit conflicts” might arise. To prevent those from the beginning, it is allowed to specify units of personal choice within ORCA’s MD input.

Luckily, this is as simple and convenient as it sounds. The parser of the MD module checks if a unit is given after a numeric constant, and automatically converts the constant to the internal default unit. If no explicit unit is given, the default unit is assumed. Please note that the default units within the MD module are not necessarily atomic units (see table below). Units are connected to the preceding numerical value by an underscore:

```
%md
  Timestep 1.0_fs
  Timestep 41.3_au # identical
  Timestep 1.0    # identical, as default time unit in MD module is fs
end
```

In the following, all units which are currently known to the MD module’s parser are listed, sorted by physical quantities. The default unit for each quantity is printed in **bold letters**. Additive constant and factor are applied to convert a unit into the default unit. The additive constant is applied before the factor. A “–” means that the constant/factor is not applied. More units will be probably added in the future.

Unit Symbol	Additive Constant	Factor
— Length Units —		
Angstrom	–	–
A	–	–
Bohr	–	0.5291
pm	–	0.01
nm	–	10.0
— Time Units —		
fs	–	–
ps	–	1000
au	–	0.02419
— Temperature Units —		
Kelvin	–	–
K	–	–
Celsius	273.15	–
C	273.15	–
— Angle Units —		
Deg	–	–
Rad	–	180/ π

9.40.3 Discussion of Features

Restarting Simulations

Ab initio molecular dynamics simulation are computationally expensive, and will typically run for a long time even in the case of medium-sized systems. Often, it is desirable to perform such a simulation as a combination of multiple short runs (*e. g.*, if the queuing system of the cluster imposes a maximum job time). The ORCA MD module writes a restart file in each simulation step, which allows for the seamless continuation of simulations. This restart file has the name “**basename.mdrestart**”, where **basename** is the project’s base name. To load an existing restart file, use the **Restart** command (*see command list below*).

In the first run of a planned sequence of runs, no restart file exists yet. for this case, the **Restart** command offers the **IfExists** modifier. The restart file is only loaded if it exists. If not, the restart is simply skipped, and no error is thrown. By using this modifier, you can have the **Restart** command already in place in the first run of a sequence (where no restart file exists in the beginning), and do not need to modify the input after the first run has finished.

Concerning the **Dump** command, it is good to know that trajectory files are appended (*not* overwritten) by default. If you ever want to overwrite an existing trajectory file by a **Dump** command, use the **Replace** modifier.

Please note that **only** the positions, velocities, thermostat internal state (*only for NHC*), Metadynamics hills, and time step counters are restarted when executing a **Restart** command. All other properties (thermostats, regions, trajectory dumps, constraints, cells, etc.) are **not** restarted. They should all remain in the input file, as executed in the first run of a sequence. Just add the **Restart** command after all other relevant commands have been executed, directly before the **Run** command.

To conclude this discussion, a short example is given. If the MD input file

```
%md
  Timestep 0.5_fs
  Initvel 300_K
  Thermostat NHC 300_K Timecon 10.0_fs
  Dump Position Stride 1 Filename "trajectory.xyz"
  Restart IfExists
  Run 100
end
```

is subsequently executed ten times (without any modification), the resulting trajectory file will be identical (*apart from numerical noise*) to that obtained if the following input is executed once:

```
%md
  Timestep 0.5_fs
  Initvel 300_K
  Thermostat NHC 300_K Timecon 10.0_fs
  Dump Position Stride 1 Filename "trajectory.xyz"
  Run 1000
end
```

Regions

In the ORCA MD module, **regions** can be defined. This concept does *not* refer to regions in space, but rather to subsets of atoms in the system. A region is nothing more than a list of atoms. Regions may overlap, *i. e.*, atoms can be part of more than one region at a time. The atoms which are part of a certain region remain the same until the region is manually re-defined, *i. e.*, regions are fixed and do not adapt to any changes in the system. There exist a few pre-defined regions which have a name. User-defined regions, in contrast, only carry an integer identifier. The following regions are pre-defined in any case:

- **all**: Contains all atoms of the system. This is the default if no region is specified in some commands, so by default, these commands will always act on the whole system.
- **active**: This region contains all movable (“non-frozen”) atoms. By default, it is identical to the **all** region. Atoms inside of this region are updated by the time integration in a molecular dynamics run, displaced in a minimization, and are considered for computing the kinetic energy.
- **inactive**: This region contains all atoms which are *not* part of the **active** region. These atoms are “frozen”; they are ignored by the time integration / minimization, and also not considered for the computation of the kinetic energy. They simply remain on their initial positions. This is in principle identical to applying Cartesian constraints to the atoms; however, it is much faster. As constraints have to be solved iteratively (*see below*), Cartesian constraints become quite computationally demanding if applied to thousands of atoms.

From these three pre-defined regions, only the **active** region can be manually modified. Changes in the composition of the **active** region automatically modify the **inactive** region. The **all** region obviously cannot be changed.

In case of a QM/MM simulation, the following four additional regions can be used:

- **qm**: This is the “quantum mechanics” region – it contains all atoms which are treated by the electron structure method.
- **mm**: This is the “molecular mechanics” region – it contains all atoms which are treated by a force-field approach. It exactly contains those atoms which are not part of the **qm** region.
- **active_qm**: Contains exactly those atoms which are part of both the **qm** and the **active** regions.
- **active_mm**: Contains exactly those atoms which are part of both the **mm** and the **active** regions.

These regions can **not** be modified in the MD input. The MD module just reads the region definitions from the QM/MM module, but is not able to make any changes here.

Regions can be useful for many purposes. For example, a “realistic” wall of atoms can be built around the system by defining the **active** region such that it only contains the non-wall atoms. The wall atoms will then be frozen. Apart from that, trajectories of regions can be written to disk, only containing the “interesting” part of a simulation. Furthermore, velocity initialization can be applied to regions, enabling to start a simulation in which different sets of atoms possess different initial temperatures. Thermostats can be attached to regions to keep different sets of atoms at different temperatures during the whole simulation. This allows for sophisticated simulation setups (cold solute in hot solvent, temperature gradient through the system, etc).

Regions are defined or modified by the [Manage_Region](#) command. Many other commands take regions as optional arguments. Please see the command list below.

Metadynamics

Metadynamics is a powerful tool to analyze free energy profiles of reactions and other processes (*solvation, aggregation, conformer change, dissociation*) based on molecular dynamics simulations. It has been developed by Laio and Parrinello in 2002 [1]. In principle, the frequency of observing a certain process in MD simulations is directly related to the free energy barrier of the process. However, many interesting processes (*such as chemical reactions*) possess such a high free energy barrier that they will never occur on the time scales typical for AIMD simulations (100 ps). To increase the frequency at which such processes happen, so-called rare event sampling methods can be employed. Metadynamics is one among those. It works by building up a bias potential as a sum of Gaussian hills, so that free energy minima are slowly filled up and the system is gradually pushed away from its resting points.

Please note that there is also a method with the same name for exploring conformation space that has been published by Grimme in 2019 [11]. It is in principle based on the original “Parrinello” Metadynamics, but with several modifications and extensions. The ORCA MD module contains the original Parrinello variant of Metadynamics [1], together with several extensions such as well-tempered Metadynamics [5] and extended Lagrangian Metadynamics [2]. The Grimme method for conformer search will probably be implemented in the future.

In Metadynamics, one has to define one or more “collective variables” (Colvars) along which the free energy profile of the system will be sampled. A Colvar is in principle nothing more than a continuous function of all atom positions which returns a real number. A simple example of a Colvar is the distance between two atoms, which could be used to explore the free energy profile of a bond formation or cleavage. In the ORCA MD module, Colvars can be defined via the `Manage_Colvar` command. Available Colvar types are distances (*including projections onto lines or into planes*), angles, dihedral angles, and coordination numbers [2]. The latter allows, *e. g.*, to accurately compute pK_A values of weak acids in solvent [3,4]. For the distances, angles, and dihedral angles, atom groups instead of single atoms can be specified, so that, *e. g.*, the distance between the centers of mass of two molecules can be defined as a Colvar.

Based on one or two Colvars (ORCA supports one-dimensional and two-dimensional Metadynamics), a Metadynamics simulation can be set up. There are many parameters to choose, which are described in the section of the `Metadynamics` command. After all parameters have been set, the actual simulation is simply started via the `Run` command. It is also possible to restart Metadynamics simulations so that they can be split into multiple successive runs; see the `Restart` command. A full example for a two-dimensional well-tempered extended Lagrangian Metadynamics simulation can be found on [page 25](#).

Note that Metadynamics simulations typically require very much computational time (*at least several 10 000 MD steps for a roughly converged result, depending on the Colvar choice*). So this is by no means a method to “shortly try out”. However, there are no cheaper methods for predicting free energy profiles (*apart from very simple approximations such as the harmonic oscillator*), and the predictive power of computing free energy profiles comes at a price.

9.40.4 Command List

In the following, an alphabetical list of all commands currently known to the MD module is given. The description of each command starts with a small box which contains the command's name and a table of arguments and modifiers. The last-but-one column in the table specifies the type of each argument. Possible types are "Integer", "Real", "String", and "Keyword". In the latter case, the last column contains a list of allowed keyword values in { **braces** }. If the type is "Real" and is a physical quantity with unit, the quantity is given in the last column in [square brackets]. Each such box is followed by a textual description of the corresponding command.

Command Overview

Command	Page	Description
Cell	12	Defines and modifies cells
Constraint	15	Manages constraints
Dump	17	Controls trajectory output
Initvel	18	Randomly initializes atom velocities
Manage_Colvar	19	Manages collective variables ("Colvars")
Manage_Region	21	Manages regions
Metadynamics	22	Sets parameters for Metadynamics runs
Minimize	26	Performs a Cartesian energy minimization
PrintLevel	27	Controls the output verbosity
Randomize	28	Sets the random seed
Restart	28	Restarts a simulation to seamlessly continue
Restraint	29	Manages restraints on Colvars
Run	30	Performs a molecular dynamics run
SCFLog	31	Controls the ORCA log file output
Screendump	31	Prints current MD state to screen
Thermostat	32	Manages thermostats
Timestep	33	Sets the integrator time step Δt

Cell				
Mandatory Arguments:		—		
Optional Arguments:		—		
Modifiers:	Cube	<i>see text</i>
	Rect	<i>see text</i>
	Rhomb	<i>see text</i>
	Sphere	<i>see text</i>
	Ellipsoid	<i>see text</i>
	None	—	—	—
	Spring	k	Real	<i>see text</i>
	Elastic	t_{avg}	Real	[time]
		c_{response}	Real	<i>see text</i>
	Anisotropic	—	—	—
	Pressure	<i>see text</i>
	Fixed	—	—	—

Defines a harmonic repulsive wall around the system (*the wall is “soft” with a spring constant and atoms can slightly penetrate; “hard” repulsive walls are not supported*). This helps to keep the molecules inside of a well-defined volume, or to keep a constant pressure in the system. In the latter case, the cell can be defined as elastic, such that it exerts a well-defined pressure (*see below*). Please note that ORCA does not feature periodic boundary conditions, and therefore, all cells are non-periodic (just repulsive walls). There are several cell geometries available (*only one type of cell can be active at a time*):

- **Cube:** Defines a cubic cell. If two real values p_1 and p_2 are specified as coordinates, the cell ranges from (p_1, p_1, p_1) to (p_2, p_2, p_2) . If only one real value p is supplied, the cell ranges from $(-\frac{p}{2}, -\frac{p}{2}, -\frac{p}{2})$ to $(\frac{p}{2}, \frac{p}{2}, \frac{p}{2})$, *i. e.* it is centered at the origin with edge length p .
- **Rect:** Defines an orthorhombic cell. Six real values $x_1, y_1, z_1, x_2, y_2,$ and z_2 have to be specified as coordinates (*in this order*). The cell will range from (x_1, y_1, z_1) to (x_2, y_2, z_2) .
- **Rhomb:** Defines a parallelepiped-shaped cell (also termed as rhomboid sometimes). You have to specify twelve real values in total. The first three define one corner point p of the cell, and the remaining nine define three cell vectors $v_1, v_2,$ and v_3 , each given as Cartesian vector components. The cell is then defined as the set of points $\{p + c_1v_1 + c_2v_2 + c_3v_3 \mid 0 \leq c_1, c_2, c_3 \leq 1\}$. The vectors $v_1, v_2,$ and v_3 do not need to be orthogonal to each other, but they may not all lie within one plane (cell volume would be zero).
- **Sphere:** Defines a spherical cell. You need to specify four real values $c_x, c_y, c_z,$ and r . The cell will then be defined as a sphere around the central point (c_x, c_y, c_z) with radius r .
- **Ellipsoid:** Defines an ellipsoid-shaped cell. As first three arguments, you have to specify three real values $c_x, c_y, c_z,$ which define the center of the ellipsoid to be (c_x, c_y, c_z) . As fourth argument, a keyword has to follow, which may either be “XYZ” or “Vectors”. In the “XYZ” case, three more real values $r_x, r_y,$ and r_z have to be specified, which define the partial radii of the ellipsoid along the X, Y, and Z coordinate axes. If instead “Vectors” was given, nine more real values $v_x^1, v_y^1, v_z^1, v_x^2, v_y^2, v_z^2, v_x^3, v_y^3, v_z^3$ have to follow after the keyword. These values define three vectors $v^1 := (v_x^1, v_y^1, v_z^1), v^2 := (v_x^2, v_y^2, v_z^2),$ and $v^3 := (v_x^3, v_y^3, v_z^3),$ which are the principal axes of the ellipsoid. These vectors have to be strictly orthogonal to each other. The length of each vector defines the partial radius of the ellipsoid along the corresponding principal axis.

All cell types define a harmonic potential $E_{\text{cell}}(r) := k \cdot r^2$ which acts on all atoms in the system **outside of the cell**, where r is the closest distance from the atom's center to the defined cell surface. Atoms whose center is inside of the cell or directly on the cell surface do not experience any repulsive force. Following from the definition, the force which acts on an atom outside of the cell is always parallel to the normal vector of the cell surface at the point which is closest to the atom center. This is trivial in case of cubic, rectangular, rhombic, and spherical cells, but not so trivial for ellipsoid-shaped cells.

The spring constant k in the above equation (*i. e.*, the “steepness” of the wall) can be specified by the “**Spring**” modifier, which expects one real value as argument. The spring constant has to be specified in the unit $\text{kJ mol}^{-1} \text{\AA}^{-2}$, other units cannot be specified here. The default value is $10 \text{ kJ mol}^{-1} \text{\AA}^{-2}$. Larger spring constants reduce the penetration depth of atoms into the wall, but may require shorter integration time steps to ensure energy conservation. If jumps in the total energy occur, try to use a smaller spring constant (*e. g.*, the default value).

The command “**Cell None**” disables any previously defined cell.

If you want to perform simulations under constant pressure, you can define an elastic cell. Then, ORCA accumulates the force which the cell exerts on the atoms in each time step, and divides this total force by the cell surface area to obtain a pressure. As this momentarily pressure heavily fluctuates, a running average is used to smooth this quantity. If the averaged pressure is larger than the external pressure which was specified, the cell will slightly grow; if it was smaller, the cell will slightly shrink. In the beginning of a simulation, the cell size will not vary until at least the running average history depth of steps have been performed.

An elastic cell is enabled by using the “**Elastic**” modifier after the cell geometry definition. Subsequently, two real values t_{avg} and c_{response} are required. While t_{avg} defines the length of the running average to smooth the pressure (*in units of physical time, not time steps*), the c_{response} constant controls how fast the cell size will change at most. More specific, c_{response} is the fraction of the cell volume growth per time step if the ratio of averaged and external pressure would be infinite, and at the same time the fraction of the cell volume reduction per step if the aforementioned ratio is zero. Put into mathematical form, the cell volume change per time step is

$$V_{\text{new}} := \begin{cases} V_{\text{old}} \cdot \frac{c_{\text{response}} \cdot \frac{\langle p \rangle}{p_{\text{ext}}} + 1}{c_{\text{response}} + 1} & \text{if } \frac{\langle p \rangle}{p_{\text{ext}}} \leq 1, \\ V_{\text{old}} \cdot \frac{(c_{\text{response}} + 1) \cdot \frac{\langle p \rangle}{p_{\text{ext}}}}{\frac{\langle p \rangle}{p_{\text{ext}}} + c_{\text{response}}} & \text{if } \frac{\langle p \rangle}{p_{\text{ext}}} > 1, \end{cases} \quad (9.1)$$

where $\langle p \rangle$ represents the averaged pressure the system exerts on the walls, and p_{ext} is the specified external pressure. Good starting points are $t_{\text{avg}} = 100 \text{ fs}$ and $c_{\text{response}} = 0.001$. Please note that larger values of c_{response} or smaller values of t_{avg} may lead to uncontrolled fluctuations of the cell size. An already defined fixed cell can be switched to elastic by the command “**Cell Elastic ...**” (the dots represent the two real arguments).

By default, the size change of an elastic cell due to pressure is performed *isotropically*, *i. e.*, the cell is scaled as a whole, and exactly retains its aspect ratio. By specifying the “**Anisotropic**” modifier after switching on an elastic cell, the cell pressure is broken down into individual components, and the size of the cell is allowed to change independently in the individual directions. This, of course, only makes sense for the cell geometries **Rect**, **Rhomb**, and **Ellipsoid**. An already defined isotropic cell can be switched

to anisotropic by simply executing “**Cell Anisotropic**”.

In case of an elastic cell, the external pressure is defined by the modifier “**Pressure**”, which expects either one or three real values as arguments. If one argument is given, this is the isotropic external pressure. If three arguments are supplied, these are the components of the pressure in X, Y, and Z direction (*in case of orthorhombic cells*) or along the direction of the three specified vectors (*in case of parallelepiped-shaped and ellipsoid-shaped cells*). This allows for anisotropic external pressure (probably only useful for solid state computations). Both the pressure and the pressure components have to be specified in units of bar ($= 10^5 \text{ N m}^{-2}$), other units cannot be used. If this modifier is not used, the default pressure will be set to 1.0 bar (isotropic) if an elastic cell is used. The external pressure of an already defined cell can be changed by the command “**Cell Pressure ...**” (the dots represent the real argument(s)).

As all cells are non-elastic by default, there is no keyword to explicitly request this at the time of cell definition. However, possible applications might require to use an elastic cell during equilibration period, and then “freeze” this cell at the final geometry for the production run. This can be achieved by using the “**Cell Fixed**” command (without any additional arguments).

If the cell is elastic, there is a volume work term which contributes to the total energy of the system. ORCA computes this term in every step and adds it to the potential energy. Without this contribution, the conserved quantity would drift excessively in elastic cell runs.

To completely switch off a previously defined cell, simply use “**Cell None**”.

Please note that cells are **not** automatically restarted by using the **Restart** command.

Examples:

Cubic cell with edge length 10 Å centered around origin:

```
Cell Cube 10
```

Spherical cell with radius 5 Å centered around origin and $20 \text{ kJ mol}^{-1} \text{ Å}^{-2}$ wall steepness:

```
Cell Sphere 0, 0, 0, 5 Spring 20
```

Elastic orthorhombic cell from $(-2, -2, 0)$ to $(12, 12, 10)$, $t_{\text{avg}} = 100 \text{ fs}$, $c_{\text{response}} = 0.001$:

```
Cell Rect -2, -2, 0, 12, 12, 10 Elastic 100, 0.001
```

Ellipsoid-shaped cell centered on origin with partial radii 5, 10, 15 Å along the X, Y, Z axes:

```
Cell Ellipsoid 0, 0, 0 XYZ 5, 10, 15
```

The commas are optional, but make sure to use them with negative numbers. By default, the minus operator will act as binary operator if possible (*see [discussion above](#)*).

Constraint				
Mandatory Arguments:		<i>operation</i>	Keyword	{ Add, Remove, List }
		<i>type</i>	Keyword	{ Cartesian, Distance, Angle, Dihedral, Center, Rigid }
		<i>atom(s)</i>	Integer	—
Optional Arguments:		—		
Modifiers:	Target	<i>value(s)</i>	Real	—
	Ramp	<i>value(s)</i>	Real	—
	Weights	<i>see text</i>
	All	—		
	Noprint	—		

Manages constraints in the molecular dynamics simulation. Unlike **Restrains**, constraints are geometric relations which are strictly enforced at every time (*i. e.*, they do not fluctuate around their target value). All atoms involved in constraints have to be included in the active region. In principle, constraints also work in Cartesian geometry optimizations with the **Minimize** command, but the performance together with L-BFGS may be poor (except for Cartesian constraints, which work flawlessly in L-BFGS). In these cases, try to use the simulated annealing method instead.

The simplest possibility is to constrain the Cartesian position of an atom to some value. A zero-based atom index is required. The command **Constraint Add Cartesian 3** would fix the fourth atom in the simulation at its current position in space. If the desired position shall be explicitly given, it can be specified via the **Target** modifier, *e. g.*, **Constraint Add Cartesian 3 Target 5.0 1.0 1.0**. To determine which dimensions to fix, one of the **XYZ, XY, XZ, YZ, X, Y, or Z** modifiers can be added. For example, **Constraint Add Cartesian 3 X Target 1.0** would constrain the X coordinate of atom 3 to the absolute value 1.0, but would not influence movement along the Y and Z coordinate at all.

By using the **Distance** keyword, distances between atoms can be fixed. The command **Constraint Add Distance 3 5** would fix the distance between atom 3 and 5 to its current value. You need to specify exactly two atom indices; multiple distance constraints are entered via multiple **Constraint** commands. Also here, a desired distance value can be given via the **Target** modifier, such as **Constraint Add Distance 3 5 Target 350_pm**.

Similarly, angles and dihedral angles between atoms can be fixed with the **Angle** and **Dihedral** keywords. Angles are defined by three atom indices, and dihedral angles by four atom indices. Also here, target values may be specified. Any combination of Cartesian, distance, angle, and dihedral constraints may be used simultaneously, and may even be applied to the same group of atoms. A molecule can be made completely rigid by constraining all its bonds, angles, and torsions. Please make sure that your constraints are not over-determined, and do not contradict each other. Otherwise, they can't be enforced and the simulation will print warnings or crash.

A different and powerful class of constraints can be defined with the **Center** argument. Directly after the keyword, a list of integer atom numbers is expected. This list can be a combination of numbers and ranges, *e. g.*, "1, 3, 5..11, 14". The weighted average position of this subset of atoms is then constrained to a fixed position in Cartesian space. By default, the weights are taken as the atom masses, such that the center of mass of the selected atoms is kept fixed. This allows, *e. g.*, to run a MD simulation of two molecules with fixed center of mass, such that their center of mass distance remains constant. Custom weights for the definition of the center can be entered by using

the **Weights** modifier after the atom list. It expects exactly the same number of real arguments as the length of the specified atom list. The geometric center of a group of atoms can be held fixed by setting all weights to 1.0, for example “**Constraint Add Center 2, 5.7 Weights 1.0 1.0 1.0 1.0**”. If desired, a **Target** for the center position can be given, which expects three real numbers for the X, Y, and Z coordinate after the keyword. If no target is specified, the current center position is held fixed.

With the **Rigid** type of constraints, complete groups of atoms can be kept rigid, *i. e.*, keep all their distances and angles relative to each other, but move as a whole. After the **Rigid** keyword, a list of atom numbers is expected. More than one group of atoms can be kept rigid at the same time – just call the **Constraint Add Rigid** command multiple times with different atom lists. Internally, the rigid constraint is realized by defining the correct number of distance constraints. Such a large number of distance constraints is hard to converge; therefore, warning messages that RATTLE did not converge will **not** be shown if a rigid constraint is active. Almost planar (or even linear) groups of atoms are hard to keep rigid by using only distance constraints. It might help to add a dummy atom outside of the plane and include this into the constraint.

ORCA supports constraints with linearly changing target value during the simulation. To define such a constraint, write “**Ramp**” directly after the “**Target**” modifier. After “**Ramp**”, twice the number of real numbers that would have been required for “**Target**” follows (two instead of one for distances, angles, and dihedrals; six instead of three for “**Cartesian XYZ**”, and so on). The first half of these arguments are the starting values, the second half are the final target values. For example, “**Constraint Add Distance 3 5 Target Ramp 300_pm 400_pm**” will define a distance constraint with a target value rising from 300 pm to 400 pm. The ramp will be performed once during the **Run** command which follows next after the constraint definition. Therefore, the number of steps specified in this **Run** command also specifies the rate at which the constraint target is modified. After the ramp has been completed once, the final (constant) target value(s) will be used for all subsequent **Run** commands.

If an already defined constraint is defined again, it is overwritten, *i. e.*, the old version of the constraint is automatically deleted first.

Constraints are removed with the **Remove** keyword. You can either remove single constraints, *e. g.*, **Constraint Remove Distance 3 5**, or groups of similar constraints. To remove all angle constraints, use **Constraint Remove Angle All**. To remove all restraints, enter **Constraint Remove All**.

The **List** argument prints all currently active constraints to the screen and log file. No additional arguments can be specified.

By default, the external force acting on each constraint is computed in every MD step and written to a file named “**basename-constraints.csv**” (*one column per constraint*). This can be useful – the average force acting on a constraint can be, *e. g.*, used for thermodynamic integration [10]. If a large number of constraints is defined, this might waste computer time if it is not required. In these cases, the constraints can be defined with the **Noprint** modifier. For such constraints, the acting forces are not computed and not written to the file. Note that constraints which have been pre-defined (*e. g.*, *by the force field for rigid molecules such as TIP3P water*) automatically have this modifier.

Please note that each constraint decreases the number of the system’s degrees of freedom (DoF) by one. This effect is included, *e. g.*, in the temperature computation, where the DoF count enters. From this consideration, it can also be understood that a constraint behaves significantly different from a restraint with very large spring constant: In the former case, the DoF is removed from the system; in the latter case, the DoF is still there, but can only move in a tiny interval.

It is computationally inefficient to define a large number of Cartesian constraints if a subset of atoms simply shall be fixed. A more efficient approach is to define an active region which only contains the atoms which shall be movable (see [Manage_Region](#) command). All atoms outside of the active region will not be subject to time integration and therefore keep their positions. However, please note that these atoms may not be involved in any other (distance, angle, dihedral) constraint.

Dump					
Mandatory Arguments:	<i>quantity</i>	Keyword	{ Position , Velocity , Force , GBW , EnGrad }		
Optional Arguments:	–				
Modifiers:	Format	<i>fmt</i>	Keyword	{ XYZ , PDB , DCD }	
	Stride	<i>n</i>	Integer	–	
	Filename	<i>fname</i>	String	–	
	Region	<i>region</i>	
	Replace	–			
	None	–			

Specifies how to write the output trajectory of the simulation. The *quantity* argument can be one of the keywords **Position**, **Velocity**, **Force**, **GBW**, and **EnGrad**. While the velocities are written in Angstrom/fs, the unit of the forces is Hartree/Angstrom. The following paragraphs only apply to the first three quantities. Dumping GBW and EnGrad files works differently, and is described at the very end of this section.

The **Stride** modifier specifies to write only every *n*-th time step to the output file (default is *n* = 1, *i. e.*, every step). A stride value of zero only writes one frame to the trajectory at the time when the **Dump** command is called – no further frames will be written during the run. This can be helpful, *e. g.*, to write an initial PDB snapshot for DCD trajectories, or to keep a single GBW file at some point.

The **Format** modifier sets the format of the output file. Currently, only the **XYZ**, **PDB**, and **DCD** formats are implemented. Please note that the **DCD** format is not well-defined, and different programs use different formats with this extension. Furthermore, **DCD** files do not store atom type information and are only valid together with a PDB snapshot of the system (a single PDB snapshot can be written via “**Dump Position Format PDB Stride 0**”). If not specified, **ORCA** tries to deduce the format from the file extension of the specified file name. If also no file name is given, trajectories will be written in **XYZ** format by default.

The **Filename** modifier sets the output file name. If not specified, the default file name will have the form “**proj-qty-rgn.ext**”, where **proj** is the base name of the **ORCA** project, **qty** is one of **postrj**, **veltrj**, or **frctrj**, **rgn** specifies the name or number of the region for which the dump is active, and **ext** is the file extension selected by the **Format** modifier.

If the trajectory file already exists at the beginning of a **Run** command, new frames will be appended to its end by default. If you want to overwrite the existing file instead, use the **Replace** modifier. The old existing file is erased only once after a dump with this modifier has been specified. If multiple **Run** commands follow after the dump definition, the trajectory will **not** be replaced before each of these runs, only before the first one among them. To overwrite the file another time, simply re-define the dump with the **Replace** modifier. If the file does not yet exist at the beginning of a run, this modifier has no

effect. Appending frames to DCD trajectories is not possible (*because they store the total frame count in the header*). Therefore, **Replace** is automatically switched on if the format is DCD.

With the **Region** modifier, the trajectory output can be restricted to a specific region (*i. e.*, subset of atoms). This modifier expects one argument, which is either the name of a pre-defined region or the number of a user-defined region (*see above*). If not specified, the trajectory of the whole system will be written. Multiple dump commands for multiple regions can be active at the same time, but each pair of region and quantity (*position/velocity/force*) can have only one attached dump command at a time (re-defining will overwrite the dump settings).

Use the **None** modifier to disable writing this quantity to an output file. The command “**Dump Position None**” will disable writing of all position trajectories for all regions. To disable only the dump for a specific region, use “**Dump Position Region r None**”, where **r** is the name or number of the region.

The default is to write a position trajectory with **Stride 1** and **Format XYZ** to a file named “**proj-postrj-all.xyz**”, where “proj” is the base name of the ORCA project. If you want to create no output trajectory at all, use “**Dump Position None**” as described above.

The **Dump GBW** command keeps a copy of the GBW file every n steps, which can be used for computing properties along the MD trajectory, *e. g.*, plotting orbitals. This does not yield a trajectory, as all the GBW files are stored individually. The value of n is controlled by the **Stride** modifier. The file names are formed by appending the step number (*six digits with leading zeros*) followed by “.gbw” to the **Filename** argument. Therefore, this argument should not contain the “.gbw” extension by itself. If the **Filename** modifier is not specified, the default will be “proj-step”, where “proj” is the base name of the ORCA project. This will lead to files such as “**proj-step000001.gbw**”, etc. The **Format** and **Region** modifiers can not be used for **Dump GBW**.

In a very similar way, **Dump EnGrad** stores an ORCA **.engrad** file (*energy and gradient*) every n steps. All the **.engrad** files are stored individually (*not as a continuous trajectory*). The value of n is controlled by the **Stride** modifier. By default file names such as “**proj-step000001.engrad**” will be used.

Initvel			
Mandatory Arguments:	<i>temp</i>	Real	[temperature]
Optional Arguments:	—		
Modifiers:	Region	<i>region</i>
	No_Overwrite	—	

Initializes the velocities of the atoms by random numbers based on a Maxwell–Boltzmann distribution, such that the initial temperature matches *temp* (see also section 9.40.5.2). Please note that this overwrites all velocities, so do not call this command when your system is already equilibrated (*e. g.*, to change temperature – use a thermostat instead).

The total linear momentum of the initial configuration is automatically removed, such that the system will not start to drift away when the simulation begins. This only concerns the initial configuration. Total linear momentum might build up during the simulation due to numeric effects.

With the **Region** modifier, the initialization of velocities can be performed for a specific region (*i. e.*,

subset of atoms). This modifier expects one argument, which is either the name of a pre-defined region or the number of a user-defined region (*see above*). If not specified, the command acts on the whole system.

The **No_Overwrite** modifier only initializes the velocities if no atom velocities have been defined/read before. This is useful in combination with the **Restart** command: After reading an existing restart file, the velocities are already known, and the initialization will be skipped if this modifier is used. The following combination of commands in a MD input would initialize the velocities only upon first execution, and restart the positions and velocities on all following executions of the same input:

```
Restart IfExists
Initvel 350_K No_Overwrite
```

If neither the **Initvel** command nor a **Restart** command is not invoked before a **Run** call, the atom velocities will be initialized to zero before starting the run.

Manage_Colvar				
Mandatory Arguments:		<i>operation</i>	Keyword	{ Define }
		<i>id</i>	Integer	–
		<i>type</i>	Keyword	{ Distance, Angle, Dihedral, CoordNumber }
Optional Arguments:		–		
Modifiers:	Atom	<i>atom</i>	Integer	–
	Group	<i>atomlist</i>	Integers	–
	Weights	<i>weights</i>	Reals	–
	Cutoff	<i>cutoff</i>	Real	[length]
	Noprint	–		

Defines collective variables (“Colvars”) which are used for **Metadynamics** or to impose **Restrains** on the system. In a general sense, a Colvar is simply a continuous function of all the atom positions which returns a real number. As Colvars don’t have any effect on the simulation by themselves, they can currently only be defined or re-defined; there is no requirement for deleting them. The second argument of the **Manage_Colvar** command is the number of the Colvar. This number is used to address the Colvar later. Allowed numbers are within the range of 1 . . . 10000. If a Colvar number which had previously been defined is defined again, it is simply overwritten (*and all restraints based on the old Colvar are deleted!*). The third mandatory argument is the type of the Colvar, which can be **Distance**, **Angle**, **Dihedral**, and **CoordNumber**. More Colvar types will probably be added in the future (*feel free to make suggestions in the forum!*).

Distance Colvars are defined between two points in space. Each point can either be a single atom (expressed by “**Atom**”) or the weighted average (center) of the positions of a group of atoms (expressed by “**Group**”). For example, the command “**Manage_Colvar Define 1 Distance Atom 0 Atom 7**” defines Colvar 1 to be the distance between atoms 0 and atom 7 (*as always, the atom count starts at zero*). On the other hand, the command “**Manage_Colvar Define 2 Distance Group 0 1 2 Group 3 4 5**” sets Colvar 2 to be the distance between the centers of atoms 0, 1, 2 and atoms 3, 4, 5. If many atoms

shall be selected, the range syntax “**Group 0..2**” can be used, including multiple such ranges if required, such as in “**Group 0..2, 5, 7..11**” (see also discussion of the [Manage_Region](#) command). By default, the center of mass is used for groups. However, weights can be manually specified if required by using the “**Weights**” modifier directly after the atom list for the center is finished. “**Weights**” expects as many real numbers as the group possesses atoms, for example “**Manage_Colvar Define 2 Distance Atom 0 Group 1 2 3 Weights 1.0 1.0 1.0**”. The “**Atom**” and “**Group**” syntax can be mixed, *e.g.*, to define the distance between a single atom and a center of mass. When defining distance Colvars, one of the modifiers **X**, **Y**, **Z**, **XY**, **XZ**, **YZ**, and **XYZ** may be specified directly after “**Distance**”. The first three among them denote that the positions shall be projected onto the corresponding Cartesian vector before computing the distance. The following three modifiers require that the two positions are projected into the corresponding Cartesian plane prior to computing the distance. The last one is the default (*just measure the standard distance in 3D space*) and does not need to be specified explicitly.

In a very similar manner, angle Colvars can be defined. Instead of two points in space, an angle Colvar is defined via three points in space, each of which can either be an “**Atom**” or a “**Group**” (see above). For example, the command “**Manage_Colvar Define 3 Angle Group 0 1 2 Atom 3 Atom 4**” defines Colvar 3 to be the angle spanned by the mass center of atoms 0, 1, 2, atom 3, and atom 4, respectively.

Dihedral Colvars are defined through four points in space, each of which can either be an “**Atom**” or a “**Group**” (see above). For example, the command “**Manage_Colvar Define 4 Dihedral Atom 0 Group 1..5 Atom 6 Atom 7**” defines Colvar 4 to be the dihedral angle spanned by atom 0, the mass center of atoms 1, 2, 3, 4, 5, atom 6, and atom 7, respectively.

The Colvar type “**CoordNumber**” has been suggested in literature [2] to measure the coordination number of some atom species around some other atom. An example where this type of Colvar has been successfully applied is the calculation of pK_A values of weak acids in solvent via Metadynamics [3,4]. The Colvar is defined by the following equation

$$C := \frac{1}{N_A} \sum_i^{N_A} \sum_j^{N_B} \frac{1 - \left(\frac{r_{ij}}{r_{\text{cut}}}\right)^6}{1 - \left(\frac{r_{ij}}{r_{\text{cut}}}\right)^{12}}, \quad (9.2)$$

where N_A is the set of atoms which is coordinated (*typically only one atom*), N_B is the set of coordinating atoms, r_{ij} is the distance between atoms i and j , and r_{cut} is a constant cutoff distance which specifies a threshold for coordination. After “**CoordNumber**”, two atoms or groups of atoms must follow, which correspond to N_A and N_B , respectively. The distance cutoff is specified after the “**Cutoff**” modifier which should follow the two group definitions. For example, the command “**Manage_Colvar Define 5 CoordNumber Atom 0 Group 1..10 Cutoff 200_pm**” defines Colvar 5 as the coordination number of the group of atoms 1 to 10 around atom 0 with a distance cutoff of $r_{\text{cut}} = 200$ pm.

For every defined Colvar, the temporal development of the position and the external force acting on the Colvar is written to a text file named “**basename-colvars.csv**” in every MD step by default. If a large number of Colvars are defined, this might be a waste of time and disk space. In these cases, the “**Noprint**” modifier can be specified when defining the Colvar. Colvars defined with this modifier will not appear in the text file, and the force acting on the Colvar will not be computed (*if not required otherwise, e.g., for restraints*).

Manage_Region				
Mandatory Arguments:		<i>identifier</i>	Keyword/Integer	...
		<i>operation</i>	Keyword	{ Define, AddAtoms, RemoveAtoms }
		<i>atomlist</i>	Integer(s)	–
Optional Arguments:		–		
Modifiers:	Element	<i>elem</i>	String	–

Defines or modifies regions. Regions are just subsets of atoms from the system – see [Section 9.40.3](#) above.

As described above, there exist several pre-defined regions which are identified by names. The only such pre-defined region which can be re-defined by the user is the **active** region. All atoms in this region are subject to time integration in molecular dynamics and displacement in minimization runs. All other atoms are simply ignored and remain on their initial positions. Please note that the **active** region may never be empty.

To re-define the **active** region, use the command “**Manage_Region active Define 1 5 7 ...**”. The integer arguments after **active** are the numbers of the atoms to be contained in the region, in the order given in the ORCA input file. Atom numbers are generally zero-based in ORCA, *i. e.*, counting starts with 0.

Apart from that, user-defined regions are supported. These are identified with an integer number instead of a name. The integer numbers do not need to be sequential, *i. e.*, it is fine to define region 2 without defining region 1. To give an example, the command “**Manage_Region 1 Define 17 18 19**” defines region 1, and adds atoms 17, 18, and 19 to this newly defined region. Using **Define** without an atom list, such as in “**Manage_Region 1 Define**”, deletes the user-defined region, as it will be empty then. Atoms can be added to or removed from previously defined regions (including the **active** region) with the **AddAtoms** and **RemoveAtoms** operations. The atom numbers specified after the operation name are added to or removed from the region. For example, “**Manage_Region active RemoveAtoms 15 16 17**” will remove atoms 15 to 17 from the **active** region (and add them to the **inactive** region instead).

If you want to specify a range of atoms, you can use the syntax “**a..b**” to include all atom numbers from **a** to **b**. If you want only, *e. g.*, every third atom in a range, you can use “**a..b..i**” to add the range from **a** to **b** with increment **i**. As an example, “**2..10..3**” will expand to the list **2, 5, 8**. You can mix atom numbers and ranges, as shown in the following two examples (*as always, the commas are optional*):

```
Manage_Region active Define 1, 4, 5..11, 14, 17..30..2
Manage_Region active RemoveAtoms 4, 15..17
```

Instead of an atom list, the **Element** modifier can be used, followed by a string which represents an element label. This will have the same effect as specifying an atom list with all atoms of this element type instead. Don't forget the double quotes around the element label string. For example, **Manage_Region active RemoveAtoms Element "H"** removes all hydrogen atoms from the active region.

Metadynamics				
Mandatory Arguments:		–		
Optional Arguments:		–		
Modifiers:	Off	–		
	Reset	–		
	Colvar	<i>colvar</i>	Integer	–
	Scale	<i>scale</i>	Real	...
	Wall	<i>side</i>	Keyword	{ Lower , Upper }
		<i>target</i>	Real	phys. unit
		<i>k</i>	Real	$\text{kJ mol}^{-1} \text{phys. unit}^{-2}$
	HillSpawn	<i>frequency</i>	Integer	–
		<i>height</i>	Real	kJ mol^{-1}
		<i>sigma</i>	Real	phys. unit
	Range	<i>from</i>	Real	phys. unit
		<i>to</i>	Real	phys. unit
		<i>resolution</i>	Integer	–
	Store	<i>store</i>	Integer	–
	Temperature	<i>temp</i>	Real	[temperature]
	WellTempered	<i>biastemp</i>	Real	[temperature]
Lagrange	<i>mass</i>	Real	a.m.u.	
	<i>k</i>	Real	$\text{kJ mol}^{-1} \text{scaleunit}^{-2}$	
	<i>target_temp</i>	Real	[temperature]	
	<i>tau</i>	Real	[time]	

Sets the parameters for a **Metadynamics** simulation [1]. After all parameters have been set, the actual simulation can be started by a **Run** command. The parameters can either all be set in a single call to the **Metadynamics** command, or distributed over multiple such calls to avoid very long lines. In both cases, there are some rules for the order of parameter settings. All Colvars for the Metadynamics need to be specified before setting any other parameters. Modifiers which are related to Colvars (such as **Scale**, **Wall**, or **Range**) only apply to the Colvar that was specified last before them in the **Metadynamics** command.

The **Colvar** modifier specifies a Colvar to be used in the Metadynamics simulation. It expects one integer argument, which is the number of the Colvar, as defined before via the **Manage_Colvar** command. The ORCA MD module supports one- and two-dimensional Metadynamics, so either one or two **Colvar** modifiers can be given. The number of **Colvar** modifiers specified defines the dimensionality of the Metadynamics simulation. Please note that modifiers which are related to Colvars (such as **Scale**, **Wall**, or **Range**) only apply to the Colvar that was specified last before them, so after specifying the first Colvar, these should be set before specifying the second Colvar.

Colvars can have different physical units, such as Angstrom for distances and degree for angles. In a multi-dimensional Metadynamics run, the different numerical magnitude of the corresponding numbers can be an issue: Angles span over a range of 180 degree, while distances will often be within an interval of only 10 Angstrom. To bring all Colvars to a similar scale, the Metadynamics module internally divides every Colvar by a user-supplied constant. These internal values are dimensionless, they will be referred to as “scale units”. For a previously specified Colvar, the scale constant can be set via the **Scale**

modifier. It expects one real argument which has to be specified in physical units of the Colvar (length units for distance Colvars, angle units for angle and dihedral Colvars). Coordination number Colvars are dimensionless anyway. If not specified, reasonable default values for the scale are used, which are 1.0 Angstrom for distance Colvars, 20.0 degree for angle and dihedral Colvars, and 0.2 for coordination number Colvars. Note that the **Scale** modifier only applies to the Colvar given last before it in the **Metadynamics** command.

As an example, consider the following commands to set up a two-dimensional Metadynamics simulation:

```
Manage_Colvar Define 1 Distance Atom 0 Atom 1
Manage_Colvar Define 2 Angle Atom 0 Atom 1 Atom 2
Metadynamics Colvar 1 Scale 1.0_A Colvar 2 Scale 10.0_Deg
```

To keep Colvars within the region of interest during Metadynamics simulations, harmonic walls can be imposed on Colvars. This is achieved via the **Wall** modifier. As a first argument, it expects the direction of the wall, which can be **Lower** or **Upper**. The second argument is the position of this wall – given in physical units of the Colvar (*e. g.*, Angstrom for distance Colvars), **not** in scale units. As an optional third real argument, the spring constant of the harmonic wall can be specified in $\text{kJ mol}^{-1} \text{unit}^{-2}$, where unit is the default physical unit of the Colvar (Angstrom for distances, degree for angles). If omitted, a spring constant of $50 \text{ kJ mol}^{-1} \text{Angstrom}^{-2}$ for distances, $0.5 \text{ kJ mol}^{-1} \text{degree}^{-2}$ for angles, and $250.0 \text{ kJ mol}^{-1}$ for coordination numbers is used (*a reasonable choice*). Both lower and upper wall can be defined after one **Wall** modifier, such as in “**Metadynamics Colvar 1 Wall Lower 3.0_A 50.0 Upper 10.0_A 50.0**”. Note that one-sided harmonic walls can also be imposed on Colvars via the **Restraint** command. For standard Metadynamics, this is redundant. However, for extended Lagrangian Metadynamics (*see below*), it makes a difference: Restraints act on the Colvar and therefore on the real atomistic system, whereas the walls defined in the **Metadynamics** command act directly on the virtual particle. Again, note that the **Wall** modifier only applies to the Colvar given last before it in the **Metadynamics** command.

The last modifier which applies to Colvars is the **Range** modifier. It has no influence on the Metadynamics run itself, and only controls the output of the free energy profiles. The **Range** modifier expects three arguments. The first two have to be real numbers and define the lower and upper interval borders for which the free energy profile with respect to this Colvar shall be output. The third argument is of integer type and controls the number of grid points to produce for this interval. In two-dimensional Metadynamics, both Colvars can have associated **Range** modifiers, which then control the interval and resolution of the 2D grid for the free energy profile. In this case, the grid should not be much finer than 100×100 ; otherwise, the evaluation of the grid points will become quite slow. If no **Range** modifier is given, default values are used (range of $0 \dots 20$ Angstrom for distance Colvars, $0 \dots 180$ degree for angle Colvars, $-180 \dots 180$ degree for dihedral Colvars, and $0 \dots 3$ for coordination number Colvars). As above, note that the **Range** modifier only applies to the Colvar given last before it in the **Metadynamics** command.

Addition of new Gaussian hills to the bias potential is controlled via the **HillSpawn** modifier. It expects three arguments. The first argument has to be of integer type and defines the hill spawning frequency, *i. e.*, every how many MD steps a new hill is added (*typically every 10–50 fs*). The second argument is a real number and specifies the height of each new hill in units of kJ mol^{-1} (*typically 0.1–1.0 kJ mol⁻¹*). The third argument sets the width of the Gaussian hills σ (which is the standard deviation, **not** the variance σ^2) in “scale units” – see the **Scale** modifier above. In two-dimensional Metadynamics simulations, the width applies for both dimensions at the same time, and the scales of the two Colvars need to be adjusted

to obtain the correct “aspect ratio” of the hill width. Standard choices for σ are 0.1–1.0 scale units. The hill spawning parameters can be changed at any point during a Metadynamics simulations, not modifying the hills which are already present. If spawning of new hills shall be temporarily suspended during a Metadynamics simulations, “**Metadynamics HillSpawn Off**” can be specified. If you want to delete all hills, consider the **Reset** modifier described below.

The **Store** modifier controls how often the current intermediate free energy profile is saved to disk. It expects one integer argument which specifies the number of MD simulation time steps between two such stores. In case of one-dimensional Metadynamics, these free energy profiles have the file names “**basename-metadynamics_profile_###.csv**”, where “**###**” indicates the step number after which the profile was written. In addition to that, a file “**basename-metadynamics_profile_history.csv**” is written, which contains all the previously computed free energy profiles as columns, so that they can easily be printed in one single plot. For two-dimensional Metadynamics, Gnuplot source files with file names “**basename-metadynamics_2d_profile_###.gp**” are written, which can be converted into contour plots with the freeware tool Gnuplot (*runs both on Windows and GNU Linux*). The raw data for these contour plots can be found in corresponding files named “**basename-metadynamics_2d_profile_###.gp.csv**”. Note that in both cases, the free energy scale origin is set to the deepest free energy well, so that all numbers are positive. If the **Store** modifier is not specified, the free energy profiles are stored every 1000 MD steps per default.

The **WellTempered** modifier switches on well-tempered Metadynamics [5]. In contrast to standard Metadynamics, the free energy profile converges towards a limit for long runs with this approach. In short terms, this approach scales down the hill size at positions where already many hills have been spawned before, so that the changes in the bias potential become smaller over time (*convergence*). The **WellTempered** modifier expects one real argument, which is the so-called bias temperature, specified in temperature units. The bias temperature should be chosen in a way so that $\frac{1}{2} \cdot k_B \cdot T_{\text{Bias}}$ is around the same size as the largest barrier which the simulation shall overcome. For example, a bias temperature of 12000 K is well-suited to overcome barriers of around 100 kJ mol⁻¹. Note that the Metadynamics module needs to know the simulation temperature in order to reconstruct the free energy profile in a well-tempered Metadynamics run. Typically, a thermostat should be active during a Metadynamics run, keeping the simulation temperature constant. In this case, the temperature is simply obtained from the thermostat. However, if no thermostat for the region **all** is specified, the simulation temperature has to be specified manually for well-tempered Metadynamics. This can be achieved by the **temperature** modifier, which expects one real argument – the simulation temperature in temperature units.

The **Lagrange** modifier switches on extended Lagrangian Metadynamics [2]. In this variant, a virtual particle (*with mass and velocity*) moves in the space spanned by the Colvars, and the only connection between this particle and the real atomistic system is a harmonic spring. The bias potential (*the Gaussian hills*) only acts on the virtual particle. The first argument is the mass of the virtual particle in a.m.u. The second argument is the harmonic spring constant in units of kJ mol⁻¹, which is evaluated in scale units – see the **Scale** modifier above. Typical values depend on the system and Colvars, but might be 100 a.m.u. and 10 kJ mol⁻¹. Optionally, a third and fourth parameter can be given to switch on thermostating of the virtual particle. A simple Berendsen thermostat is applied here. The third argument is the target temperature of the virtual particle, and the fourth argument is the thermostat time constant τ . A good choice would be a target temperature of 100 K and $\tau = 10$ fs. Note that in contrast to the normal Berendsen thermostat, the virtual particle is only cooled, but never heated. In

other words, the thermostat only becomes active if the instantaneous temperature of the virtual particle becomes larger than the target temperature. This is to ensure that the virtual particle can change its direction – otherwise, it might happen that it is driven in the same direction for very long time intervals.

The **Reset** modifier resets the bias profile, *i. e.*, it deletes all hills which had been spawned, so that the bias profile becomes flat again. All other parameters of the Metadynamics simulation are not modified. If, for example, hill spawning is still on, then new hills will be spawned in the next simulation run.

The **Off** modifier completely switches off Metadynamics. It deletes all hills and turns off the Metadynamics module. It also resets the choice of Colvars for Metadynamics, so you will need to use this first if you want to set up a second different Metadynamics run within the same input script. This modifier can only be given as first argument to the **Metadynamics** command, and no further arguments can follow.

A restart file for the Metadynamics module (file name “**basename.metarestart**”) is written each time a new hill has been spawned. The **Restart** command detects this file and automatically restarts the Metadynamics run (*i. e.*, loads all hills and the positions and velocities of the extended Lagrangian virtual particle if active). However, this only happens when Metadynamics is active and set up at the time when the **Restart** command is invoked. The parameters for the Metadynamics simulation are **not** restarted. Therefore, leave all parameter settings via calls to the **Metadynamics** command in place in your input file, and simply call the **Restart** command after all those, directly before the **Run** command.

Please see also the discussion on Metadynamics in [Section 9.40.3](#).

This section is concluded with a full example for a two-dimensional well-tempered extended Lagrangian Metadynamics run with restart ability (*just run the same input again to continue the simulation where it ended last*). The two Colvars are defined as distances between atoms. You need to adapt all parameters in [blue](#) to your question and system:

```
Timestep 0.5_fs
Initvel 350_K
Thermostat NHC 350_K Timecon 100.0_fs
Dump Position Stride 1 Filename "trajectory.xyz"

Manage_Colvar Define 1 Distance Atom 0 Atom 1
Manage_Colvar Define 2 Distance Atom 2 Atom 3

Metadynamics Colvar 1 Scale 1.0_A Wall Lower 3.0 50.0 Upper 10.0 50.0 Range 0.0 15.0 100
Metadynamics Colvar 2 Scale 1.0_A Wall Lower 1.0 50.0 Upper 8.0 50.0 Range 0.0 13.0 100
Metadynamics HillSpawn 40 0.5 0.5 Store 2000
Metadynamics WellTempered 6000_K
Metadynamics Lagrange 100.0 10.0 200.0_K 10.0_fs

Restart IfExists
Run 100000
```

Minimize				
Mandatory Arguments:		–		
Optional Arguments:		<i>method</i>	Keyword	{ Combined , L-BFGS , Anneal }
Modifiers:	Steps	<i>n</i>	Integer	–
	MaxGrad	<i>thres</i>	Real	[kJ mol ⁻¹ Å ⁻¹]
	RMSGrad	<i>thres</i>	Real	[kJ mol ⁻¹ Å ⁻¹]
	TempConv	<i>thres</i>	Real	[temperature]
	Accel	<i>value</i>	Real	–
	Damp	<i>value</i>	Real	–
	StepLimit	<i>value</i>	Real	[length]
	History	<i>n</i>	Integer	–
	Noise	<i>value</i>	Real	[length]
OnlyH			–	

Performs a Cartesian energy minimization of the system. For molecules, this is less efficient than ORCA's built-in geometry optimization in internal coordinates (*i. e.*, *requires more steps to converge*). However, the algorithms employed here also work with large atom counts (*e. g.*, *50 000*) as sometimes encountered in QM/MM simulations, which is absolutely out of scope of ORCA's primary optimization module. Furthermore, the minimization also works under all types of constraints (which some limitations in the case of L-BFGS) that have been set with the **Constraint** command, and also includes the effect of the repulsive simulation cell if activated. Only atoms contained in the active region are displaced, while all other atoms are kept at their positions.

The simplest way of performing a minimization is simply calling the **Minimize** command without arguments. This defaults to the L-BFGS method, which is fairly robust and efficient. If the minimization seems unstable, try to reduce the **History** or **StepLimit** parameters. L-BFGS may sometimes show poor performance with constraints other than Cartesian type. Apart from that, there is also a simulated annealing method implemented, which can be selected by specifying **Anneal** as the first argument. In contrast to L-BFGS, the simulated annealing method works equally well with all types of constraints. There is also a **Combined** method, which is a combination of some L-BFGS steps in the beginning, followed by a simulated annealing run until the temperature falls below a threshold, and another final L-BFGS run until the convergence criteria are reached.

With the **Steps** modifier, the maximum number of minimization steps can be specified. If this number of steps has been performed, the minimization finishes, no matter if the convergence criteria are fulfilled or not. The default value is 500.

The **MaxGrad** and **RMSGrad** modifiers control the convergence thresholds for the largest gradient on some atom and the root mean square average of the gradients. The default values are currently set to 5.0 and 1.0 kJ mol⁻¹ Å⁻¹, respectively, which is about the same criterion as the default setting in the primary ORCA geometry optimization.

If the **TempConv** modifier is given, a simulated annealing run finished after the temperature was monotonously decreasing within 5 successive steps, and dropped below the specified value. Note that the simulated annealing run will finish if either this condition is reached, or the gradient thresholds are observed. It is not required to fulfill both criteria.

The **Accel** modifier specifies the acceleration factor for simulated annealing runs (*has no effect on L-BFGS*). As long as the angle between velocity vector and gradient vector of some atom is below 90 degrees, the gradient is multiplied by this factor and the velocity is multiplied by a fraction of this factor. This helps to enforce a faster movement in gradient direction. The default value is 4.0. If this feature is not desired, use **Accel 1.0** to switch it off (1.0 means “no artificial acceleration”).

The **Damp** modifier is the damping factor for simulated annealing runs (*has no effect on L-BFGS*). Atom velocities are multiplied by this factor in every integration step. The default value is 0.98. Smaller values will make the algorithm more stable and less prone to oscillations and overshoots, but will also require significantly more steps to converge. Don't use values ≥ 1 , as then it won't be an “annealing” anymore :-)

The **StepLimit** modifier specifies the maximum displacement of any atom (*in length units*) that can happen in one step of a minimization run. This can help to avoid large, unreliable steps which could lead to abrupt jumps in geometry and very high potential energies. This modifier concerns both L-BFGS and simulated annealing runs. Negative values disable the step limit. The step limit is disabled by default. If you need to switch it on, try something in the order of 0.1 Å.

The **History** modifier controls the depth of gradient and position vector history that is used in the L-BFGS method to approximate the inverse Hessian. The default value is 20. Smaller values can help to stabilize the algorithm.

With the **Noise** modifier, small random numbers can be added to the atom positions before the minimization starts. This can help to escape local maxima and saddle points in the minimization. For example, a minimization of an initially linear water molecule would not be able to leave this maximum – but with some random “noise”, it will be possible. The modifier expects one real argument which specifies the maximum atom displacement in length units (something like 0.01 Å will be reasonable). This feature is switched off by default.

If the **OnlyH** modifier is given, all non-hydrogen atoms are removed from the active region before the minimization starts. After the minimization has finished, the original active region is restored. This is helpful if only hydrogen positions shall be optimized, *e. g.*, to refine experimental crystal structures.

PrintLevel			
Mandatory Arguments:	<i>value</i>	Keyword	{ Low , Medium , High , Debug }
Optional Arguments:	–		
Modifiers:	–		

Controls the amount of information which is printed to the screen during the simulation. **Debug** should be used only in rare cases, because it might slow the simulation down heavily.

The default value is **Medium**.

Randomize			
Mandatory Arguments:	–		
Optional Arguments:	<i>seed</i>	Integer	–
Modifiers:	–		

There are a few algorithms in the ORCA MD module which rely on random numbers, *e. g.*, the initialization of atom velocities with the **Initvel** command. These random numbers are so-called “pseudo-random numbers”, produced by a deterministic generator. This generator has a *state*, which is simply an integer number. If initialized to the same state, the generator will always create the same sequence of “random” numbers. This sounds like a deficiency at first thought, but is a very important feature for scientific reproducibility and for debugging purposes. If you start the same MD input file with “random” velocity initialization a couple of times, the trajectory will be exactly identical in all runs.

However, there are cases in which this behavior is not desired, *e. g.*, if you want to average a property over multiple trajectories of the same system. In these cases, call the **Randomize** command in the beginning of the input. If no argument is given, the random number generator is initialized with the current system time as a seed. MD runs started at different times will have different random velocities in the beginning. If you want more control over this process, you can also specify a positive integer number as argument, which is used as initial random seed. Simulations started with the same seed argument will have identical initial random velocities (if all other system parameters such as atom count, atom types, ... remain identical).

Without a call to **Randomize**, a seed of 1 is always used.

Restart			
Mandatory Arguments:	–		
Optional Arguments:	<i>fname</i>	String	–
Modifiers:	IfExists	–	

Reads a restart file to continue a previous molecular dynamics run. Such a restart file is written after every simulation step, such that a crashed simulation may easily be recovered. The file name of the restart file may be given via *fname*; otherwise, it is deduced from the project’s base name as **<basename>.mdrestart**.

If the **IfExists** modifier is specified, a restart is only performed if the restart file exists. The error and abort that would normally occur in case of a non-existent restart file are suppressed by this flag. This is useful in the first of a series of batch runs, where the restart file does not yet exist in the beginning.

Please note that the following quantities are stored to/loaded from restart files:

- Atom Positions
- Atom Velocities
- Thermostat internal state (*only for NHC*)
- Metadynamics hills and extended Lagrangian internal state
- Simulation step number and elapsed physical time

All other quantities (timestep, regions, thermostat, constraints, cells, etc.) are **not** restarted and need to be set in the input file, typically **before** the **Restart** command. It is safe to just call the **Restart** command immediately before the **Run** command.

Please see also the discussion on restarting simulations in [Section 9.40.3](#).

Restraint				
Mandatory Arguments:	<i>operation</i>	Keyword	{ Add , Reset }	
		Keyword	Colvar	
	<i>colvar</i>	Integer	–	
Optional Arguments:	–			
Modifiers:	Harmonic	–		
	Gaussian	–		
	Spring	–		
	Sigma	<i>Gaussian Sigma</i>	Real	...
	Height	<i>Gaussian Height</i>	Real	kJ mol^{-1}
	Target	<i>target</i>	Real	...
	Lower	<i>lower wall</i>	Real	...
	Upper	<i>upper wall</i>	Real	...
	Ramp	<i>initial target</i>	Real	...
		<i>final target</i>	Real	...
Noprint	–			

This command imposes restraints on collective variables (“Colvars”) defined before via the **Manage_Colvars** command. As a first argument, it expects the kind of operation to perform, which can be **Add** and **Reset**. The second argument needs to be the keyword “**Colvar**”, and the third argument is an integer number specifying the Colvar on which the operation shall be performed.

If the first argument is “**Add**”, a new restraint is added to the specified Colvar. Note that an arbitrary number of restraints of different types can be active on a Colvar at the same time. The next argument after the Colvar number needs to be the type of the restraint. Currently, **Harmonic** and **Gaussian** are allowed. When adding harmonic restraints, the **Spring** modifier can be given, specifying the harmonic spring constant of the restraint in $\text{kJ mol}^{-1} \text{unit}^{-2}$, where unit is the default physical unit of the Colvar (Angstrom for distances, degree for angles). If not specified, a value of $50 \text{kJ mol}^{-1} \text{unit}^{-1}$ is used. When adding Gaussian restraints, the **Height** and **Sigma** modifiers are allowed. The former sets the height of the Gaussian hill in kJ mol^{-1} , while the latter sets the width of the Gaussian function in physical Colvar units (σ is the standard deviation, **not** the variance σ^2). The height can be either positive or negative, allowing for both Gaussian hills and Gaussian wells. If not specified, the default values of -10kJ mol^{-1} for the height (*i. e.*, *Gaussian well*) and 10 Colvar units (*e. g.*, *Angstrom or degree*) for sigma are used.

The position of the new restraint is controlled via the **Target** modifier, which expects one real argument in Colvar units. If the restraint shall be an one-sided wall, the modifiers **Lower** and **Upper** can be used instead of **Target**. It is also possible to specify both **Lower** and **Upper** in order to define a lower and an upper wall at different positions in one command. If **Ramp** is given directly after **Target**, **Lower**, or **Upper**, a restraint with linearly moving target position over time is defined. **Ramp** expects two arguments,

which are the initial restraint position, and the final restraint position after the next subsequent **Run** command.

The following example shows how to assign a harmonic two-sided restraint with different lower and upper wall parameters to a distance Colvar with number 7 that has been previously defined via the **Manage_Colvars** command:

```
Restraint Add Colvar 7 Harmonic Lower 400_pm Spring 50.0
Restraint Add Colvar 7 Harmonic Upper 800_pm Spring 80.0
```

By default, some additional data (*current position, potential energy, external force, internal force*) for each restraint is printed to a file with the name “**basename-restraints.csv**” in every MD step. This data can be used, *e. g.*, for thermodynamic integration. If a large number of restraints is defined, this can waste time and disk space. To switch this off for a restraint, specify the **Noprint** modifier when defining it.

If the first argument was “**Reset**”, all restraints imposed on the specified Colvar are deleted. No further arguments or modifiers (*apart from the three mandatory arguments described above*) can be given.

Run				
Mandatory Arguments:	<i>n</i>	Integer	–	
Optional Arguments:	–			
Modifiers:	StepLimit	<i>value</i>	Real	[length]
	CenterCOM	–		

Performs a molecular dynamics run over *n* time steps with the current settings, applying the velocity Verlet algorithm to solve the equations of motion (see section 9.40.5.1). You might want to call commands like **Timestep**, **Initvel**, **Thermostat**, and **Dump** before. Please note that only atoms within the **active** region will be subject to time integration. All other atoms will be skipped, and will therefore retain their initial positions.

The **StepLimit** modifier can be used to limit the maximum displacement of any atom in a MD time integration step. In addition to the displacement, also the velocities will be limited to a maximum of $value \cdot \Delta t$. This can help to stabilize the dynamics if the initial geometry is poor and large forces are acting (close atoms, etc.). The keyword expects one real argument in distance units. A reasonable choice would be 0.1 Å.

If the **CenterCOM** modifier is given, the center of mass (CoM) of the total system is kept fixed. Normally, the CoM should not drift anyway, because the velocity initialization is performed in a way which gives the CoM a zero initial velocity, and the conservation of momentum should keep it like that. However, numerical errors and massive **Thermostats** (*among other factors*) can break this momentum conservation, leading to a drift of the CoM over time. If this shall be avoided, specify this modifier.

If no call to **Initvel** occurred before this command, the atom velocities are initialized to zero. If no call to **Timestep** occurred before this command, a default time step of 0.5 fs is set.

You can cleanly end a MD run by creating an empty file with the name “**EXIT**” (*note the all-uppercase letters on case-sensitive file systems*). On Unix operating systems such as GNU Linux, this can easily be achieved by the command “**touch EXIT**”. ORCA will detect the file, abort the MD run, and delete the

file. You will still get the remaining output (*such as the timing statistics*), and you don't have to delete all the remaining “.tmp” files, which both would not be the case if you would have killed the process instead.

SCFLog			
Mandatory Arguments:	<i>value</i>	Keyword	{ Discard , Last , Append , Each }
Optional Arguments:	—		
Modifiers:	—		

Controls how/if the detailed output from the electron structure calculation (*i. e.*, integrals, scf, gradient, ...) will be written to log files. **Discard** completely discards the output. **Last** only keeps the last output for each program call (useful to read error message if simulation aborts). **Append** redirects all the output into one single log file (“basename.scf.log”, “basename.int.log”, “basename.grad.log”, ...), appending each step at the end of the file. **Each** writes the output for each step and each program to different log files, which have the step number in their file names.

The amount of information which is printed to the SCF log file can be controlled by the standard ORCA print flags, such as “%output PrintLevel Maxi end”. Note that by default, ORCA reduces the print level after the first SCF. Due to this, properties such as orbital energies and population analyses will only be printed once by default. If you want to keep the print level constant for subsequent SCF runs, disable this feature via “%method ReducePrint false end” in the ORCA input.

The default value is **Append**. Note that this can lead to large log files in long runs.

Screendump	
Mandatory Arguments:	—
Optional Arguments:	—
Modifiers:	—

Prints the current state of the MD module (atom positions, velocities, potential and kinetic energy, cell properties, etc.) to the screen and log file in a well-defined and “grepable” format. This is mostly useful for unit testing, *e. g.*, to verify if the system state after a MD run equals the state obtained from some other ORCA binary distribution.

Thermostat				
Mandatory Arguments:	<i>type</i>	Keyword	{ Berendsen , CSVR , NHC , None }	
Optional Arguments:	<i>temperature</i>	Real	[temperature]	
Modifiers:	Timecon	<i>tau</i>	Real	[time]
	Ramp	<i>target_temp</i>	Real	[temperature]
	Chain	<i>chain_length</i>	Integer	–
	MTS	<i>mts</i>	Integer	–
	Yoshida	<i>yoshida</i>	Integer	–
	Region	<i>region</i>
	Massive			–

Changes the atom thermostat settings for subsequent simulation runs. “*Type*” sets the thermostat type. Currently, three thermostat types are implemented: Berendsen [12], Nosé–Hoover chains (NHC) [6, 7], and “Canonical Sampling through Velocity Rescaling” (CSVR) [8]. The very basic and robust Berendsen thermostat should only be used for early pre-equilibration runs, as it does **not** sample the canonical ensemble and leads to problems such as the flying ice cube effect. Both the NHC and the CSVR thermostats are very sophisticated, and correctly sample the canonical ensemble. One of these two should be used in all standard NVT simulations. Use **None** as type to disable the thermostat.

The optional *temperature* argument sets the target temperature to which the system is thermostated. If this argument is omitted, the temperature from the last call to the **Initvel** command is used (if no such call was invoked before, the simulation is aborted).

The **Timecon** modifier sets the coupling strength of the thermostat (large time constants correspond to weak coupling). The default value is 10 fs, which is a relatively strong coupling. For a production run, 100 fs would be appropriate. Values in the range of 10 ... 100 fs are reasonable (see also section 9.40.5.3).

If the **Ramp** modifier is used, a temperature ramp can be applied during a MD run. The final temperature at the end of the ramp has to be specified directly after the modifier. The initial temperature at the beginning of the ramp is taken from the *temperature* argument (or from the last **Initvel** command if this argument is missing). The temperature ramp is applied only to the **Run** command which first follows the ramp definition. The slope of the ramp is chosen such that the final temperature is reached at the end of the run. Any subsequent **Run** command will simply use the final temperature for thermostating. To apply another temperature ramp, you need to explicitly define it again.

The **Chain**, **MTS**, and **Yoshida** modifiers only apply to NHC thermostats. They specify the chain length of the Nosé–Hoover chain (default: 3), the number of multiple time steps in which the thermostat integration is performed (default: 2), and the order of the Yoshida integrator used (default: 3, allowed: 1, 3, 5, 7), respectively. Normally, there is little need to modify one of these parameters. For more information, refer to the original publications [6, 7].

The **Massive** modifier activates *massive thermostating*, which means that each degree of freedom is assigned to an independent thermostat. This is useful for pre-equilibration runs (helps to reach energy equipartition) and should not be used during production runs, as it might heavily distort the dynamics. Note that massive thermostats also break the conservation of momentum (both linear and angular), so better specify the **CenterCOM** modifier for the **run** command if this is an issue. Please also note that massive NHC thermostats of large systems can be quite slow, because each NHC thermostat is a

dynamical system on its own which needs to be time integrated.

With the **Region** modifier, the thermostat can be attached to a specific region (*i. e.*, subset of atoms). This modifier expects one argument, which is either the name of a pre-defined region or the number of a user-defined region (*see above*). If not specified, the thermostat acts on the whole system. Multiple thermostats for multiple regions can be active at the same time, but each region can have only one attached thermostat at a time (re-defining will overwrite the thermostat settings).

The command “**Thermostat None**” will remove all thermostats from all regions. If you want to disable a thermostat for a specific region only, use “**Thermostat None Region r**”, where **r** is the name or number of the region.

Please note that all three implemented thermostat types will show no effect (or unexpected effects) if the system’s temperature is close to 0 K, as they all work by multiplying the velocities with a (*more or less complicated*) factor.

Timestep			
Mandatory Arguments:	<i>dt</i>	Real	[time]
Optional Arguments:	–		
Modifiers:	–		

Sets the simulation time step Δt used to integrate the equations of motion for all following runs to *dt*. If your system contains hydrogen atoms, a time step not above 0.5 fs is recommended. If only heavier atoms are present, a larger time step may be chosen. A good estimate for a time step that still allows for an accurate simulation is $\Delta t = \sqrt{m} \cdot 0.5$ fs, where *m* is the mass of the lightest atom in the system (in a.m.u.). This is one reason why some scientists perform simulations with fully deuterated compounds: It allows to increase the time step by a factor of ≈ 1.4 :-)

If this command is not invoked before a **Run** call, a default time step of 0.5 fs will be set before starting the run.

9.40.5 Scientific Background

In this section, some of the methods and algorithms used within ORCA's MD module are described in some more depth, with a focus on the scientific background.

9.40.5.1 Time Integration and Equations of Motion

The central concept of molecular dynamics simulations is to solve Newton's equations of motion (at least as long as the atom cores are treated classically). These read

$$\ddot{x}_i(t) = \frac{F_i(\vec{x}(t))}{m_i}, \quad i = 1 \dots N, \quad (9.3)$$

where $x_i(t)$ denotes the position of the i -th degree of freedom at time t , m the corresponding mass, and F_i the force acting upon this degree of freedom. As the force may depend on all positions, this is a coupled system of N ordinary differential equations (ODEs). In the general case, it is not possible to obtain an analytical solution of this system, and therefore numerical solution methods are applied. These are almost always based on discretizing the time variable and approximately solving the system by taking finite time steps.

Of all different methods to numerically solve coupled systems of ODEs, the *symplectic integration schemes* for Hamiltonian systems attained special attention in the field of molecular dynamics. They possess a very good conservation of energy. In contrast to many other methods, they show a reasonable behavior when investigating the long-term evolution of chaotic Hamiltonian systems (like, *e. g.*, MD simulations). Three popular such symplectic integration schemes are the *Leapfrog* algorithm, the *Verlet* method, and the *Velocity Verlet* integrator. Despite their different names, they are very similar. It can be easily seen that the Verlet and Velocity Verlet methods are algebraically equivalent (by eliminating the velocities from the Velocity Verlet algorithm), and it can be shown that, eventually, all three methods are identical.³ All three methods are explicit integration methods with a global error of order 2, and therefore one order better than the semi-implicit Euler method, which is also a symplectic integration scheme. As the Velocity Verlet algorithm is the only of these three methods which yields velocities and positions at the same point in time, many popular molecular dynamics packages (CP2k, CPMD, LAMMPS) use this scheme. For the same reasons, the ORCA MD module uses the Velocity Verlet algorithm as time integration method.

The general equations of the Velocity Verlet scheme read

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t) \Delta t + \frac{1}{2} \vec{a}(t) \Delta t^2, \quad (9.4)$$

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \frac{\vec{a}(t) + \vec{a}(t + \Delta t)}{2} \Delta t. \quad (9.5)$$

By inserting

$$\vec{a}_i(t) = \frac{\vec{F}_i(t)}{m_i}, \quad i = 1 \dots N, \quad (9.6)$$

³Hairer, Lubich, Wanner, "Geometric Numerical Integration", Springer 2006.

one arrives at the two-step method

$$\vec{x}_i(t + \Delta t) = \vec{x}_i(t) + \vec{v}_i(t) \Delta t + \frac{\vec{F}_i(t)}{2m_i} \Delta t^2, \quad i = 1 \dots N, \quad (9.7)$$

$$\vec{v}_i(t + \Delta t) = \vec{v}_i(t) + \frac{\vec{F}_i(t) + \vec{F}_i(t + \Delta t)}{2m_i} \Delta t, \quad i = 1 \dots N, \quad (9.8)$$

which is implemented in ORCA's MD module.

9.40.5.2 Velocity Initialization

In the beginning of a MD simulation, it is often the case that only the initial positions of the atoms are known, but not the velocities. As MD simulations are performed at some finite temperature, it is a good idea to initialize the velocities in a way such that the desired simulation temperature is already present in the beginning. In statistical mechanics, it is often assumed that the velocity distribution of atoms is given by a Maxwell–Boltzmann distribution (which is strictly only the case in idealized gases). Therefore, it is a reasonable choice to initialize the atom's velocities according to the Maxwell–Boltzmann equation in the beginning of a MD simulation. The goal is to find an initial velocity distribution in which each degree of freedom possesses a similar amount of energy, such that the equipartition theorem is approximately fulfilled.

The scalar Maxwell–Boltzmann velocity distribution (leaving out the normalization factor) at temperature T is given by

$$f(v) = v^2 \exp\left(-\frac{mv^2}{2k_B T}\right). \quad (9.9)$$

To initialize the particle's velocities such that this distribution function is fulfilled, one starts with a series of normal-distributed random numbers with mean 0 and variance 1, denoted by $\mathcal{N}(0, 1)$. The Cartesian velocity components for each atom are then computed by

$$v_{i,\alpha} := \sqrt{\frac{k_B T}{m_i}} \mathcal{N}(0, 1), \quad \alpha \in \{x, y, z\}, \quad i = 1 \dots N. \quad (9.10)$$

As the C++98 standard does not offer a platform-independent way of obtaining normal-distributed random numbers, these are internally computed from uniformly distributed random numbers by applying the *Box–Muller transform* [13]: Assuming that u_1 and u_2 are two uniformly distributed random numbers from the interval $[0, 1]$, the equations

$$z_1 := \sqrt{-2 \log(u_1)} \cos(2\pi u_2), \quad (9.11)$$

$$z_2 := \sqrt{-2 \log(u_1)} \sin(2\pi u_2) \quad (9.12)$$

yield two new random numbers z_1 and z_2 which obey a normal distribution with mean 0 and variance 1.

After the velocities have been initialized, the total linear momentum of the system will probably have some finite value other than zero. As the linear momentum is (approximately) conserved within a molecular dynamics simulation, this would result in the system drifting away into one direction during the course of the

simulation, which is probably not desired. Therefore, the total momentum is explicitly set to zero after the Maxwell–Boltzmann initialization:

$$\vec{P}_{\text{tot}} := \sum_{i=1}^N m_i \vec{v}_{i,\text{old}}, \quad (9.13)$$

$$\vec{v}_{i,\text{new}} := \vec{v}_{i,\text{old}} - \frac{\vec{P}_{\text{tot}}}{m_i N}, \quad i = 1 \dots N. \quad (9.14)$$

This, of course, might change the initial temperature. Therefore, a final step is performed, in which all velocity vectors are multiplied with a factor that is determined such that the initial temperature exactly matches the target value.

9.40.5.3 Thermostats

After the initial velocities have been initialized to some finite temperature, it might be assumed that one can simply start the time integration of the dynamical system (equivalent to the *NVE ensemble*), and the starting temperature would be approximately preserved. In a real system, however, there are (at least) two reasons why the temperature will strongly deviate from the initial value already after a few steps. First, the initial velocity distribution only considers the kinetic energy of the particles, but some amount of energy will be exchanged with the potential energy contribution (*e. g.*, bond stretching) immediately, altering the temperature. Secondly, the numerical errors introduced due to the finite time step (and in case of *ab initio* MD, also due to the approximate forces) will lead to a drift in energy and therefore in temperature. To counter these effects, it is often desirable to have a temperature control during the course of the simulation (which then runs in the *NVT ensemble*), which is called a thermostat.

There exist many different kinds of thermostats, ranging from simple expressions up to highly complex dynamical systems on their own. But all of them share a common issue: If the thermostat is coupled only weakly to the system, the temperature will change anyway. However, if the thermostat is coupled more strongly to the system (*i. e.*, intervenes stronger), then the dynamics of the simulation will change, no longer resembling the undisturbed original dynamics which one wants to investigate. Therefore, it is always a tradeoff between temperature stability and disturbed dynamics to decide how strong a thermostat should be coupled to the system.

In ORCA, currently three thermostats are implemented: The Berendsen thermostat [12], the Nosé–Hoover chain thermostat (NHC) [6, 7], and the “Canonical Sampling through Velocity Rescaling” thermostat (CSVR) [8].

Berendsen Thermostat

The Berendsen thermostat [12] is similar to the simple velocity rescaling scheme, but enhanced by a time constant τ to control the coupling strength. Let T_0 be the desired target temperature and T the current temperature of the system. Then the temperature gradient caused by the thermostat can be expressed as

$$\frac{dT}{dt} = \frac{T_0 - T}{\tau}. \quad (9.15)$$

Considering the fact that discrete time steps Δt are used, the correction factor for the velocities in each time step is determined by

$$f := \sqrt{1 + \frac{\Delta t (T_0 - T)}{T\tau}} \quad (9.16)$$

The new velocities are then easily obtained as

$$\vec{v}_{i,\text{new}} := f \cdot \vec{v}_{i,\text{old}}, \quad i = 1 \dots N. \quad (9.17)$$

Let's consider some special cases. If $\tau = \Delta t$, the whole temperature deviation from T_0 is corrected immediately, such that the temperature is always exactly kept at the target value. This is identical to simple velocity rescaling (without any time constant), which is known to work poorly for most systems (a single harmonic oscillator would, *e. g.*, simply explode). With a larger time constant $\tau > \Delta T$, the coupling strength is reduced, leading to reasonable results. Typically, a value of τ in the range of $20 \dots 200 \cdot \Delta T$ will be applied. For $\tau \rightarrow \infty$, the coupling strength goes to zero, such that the thermostat is no longer active. Values of $\tau < \Delta T$ are not allowed.

From the formula, it becomes clear that a Berendsen thermostat will have no effect if the system has a temperature of 0 K (or in the “massive” case: if the considered degree of freedom has 0 K), because it is based on multiplying the velocities by a factor to modify the temperature. Therefore, this type of thermostat can't be used to heat a system up starting from 0 K.

9.40.5.4 Constraints

Unlike restraints, constraints are geometric relations which are strictly enforced at every time (*i. e.*, they do not fluctuate around their target value). Many molecular dynamics techniques make use of geometric constraints (*e. g.*, to keep water molecules rigid, or to fix some reaction coordinate). Standard BOMD describes the nuclei as point charges in space, such that the motion of the atoms is governed by the laws of classical mechanics. Systems in classical mechanics can be described by the Lagrange formalism, which contains a well established sub-formalism for holonomic constraints, namely the method of Lagrange multipliers.

However, molecular dynamics discretizes time to solve the equations of motions with finite time steps, often using a Verlet integrator. With discretized time, it is slightly more involved to enforce and keep exact constraints. Within the last decades, algorithms have been developed to do so. One famous among them is the SHAKE algorithm. However, it comes with the disadvantage of only enforcing the constraints in the positions, not in the velocities. This may lead to problems such as artificially high temperature values due to “hidden” velocities along the constrained directions. An extension of SHAKE which also enforces the constraints for the velocities is the RATTLE algorithm, which is implemented in the AIMD module of ORCA.

The RATTLE scheme is a generalization of the Velocity Verlet integrator to allow for constraints. This means that RATTLE is not applied in addition to the Velocity Verlet integrator, but replaces it. In case of no active constraints, both methods are identical. A system of coupled constraints cannot be solved exactly in one step, and RATTLE uses an iterative approach to enforce all constraints simultaneously. This is often a matter of concern with respect to performance. However, in AIMD, the energy and gradient calculations typically take seconds or even minutes per step, such that the additional computation time for iteratively solving the constraints can be totally neglected.

As an iterative procedure, RATTLE is not able to give exact solutions, but only converged up to a given tolerance. In the ORCA MD module, the tolerance is currently set to 10^{-2} pm for distances, and 10^{-4} degree for angles and dihedral angles. This tolerance is typically reached within a few dozen iterations. In some cases, it might happen that the RATTLE iterations do not converge to the required tolerance. This is typically the case if the set of constraints is over-determined or contradictory.

The mathematical and technical details of RATTLE are not described here, they can be found in the literature. The general concept of RATTLE was suggested by Andersen [14]. The original article only covered distance constraints. A follow-up work describes how to handle any holonomic constraints, in particular how to constrain angles and dihedral angles [15]. The Wilson vectors (*i. e.*, derivatives of angles and dihedral angles with respect to Cartesian atom positions) are taken from Wilson’s original work [16].

Bibliography

- [1] Laio, A.; Parrinello, M. (2002) *Proc. Natl. Acad. Sci. U.S.A.*, 99(20), 12562.
- [2] Iannuzzi, M.; Laio, A.; Parrinello, M. (2003) *Phys. Rev. Lett.*, 90, 238302.
- [3] Tummanapelli, A. K.; Vasudevan, S. (2014) *J. Phys. Chem. B*, 118(47), 13651.
- [4] Tummanapelli, A. K.; Vasudevan, S. (2015) *Phys. Chem. Chem. Phys.*, 17, 6383.
- [5] Barducci, A.; Bussi, G.; Parrinello, M. (2008) *Phys. Rev. Lett.*, 100, 020603.
- [6] Martyna, G. J.; Klein, M. L.; Tuckerman, M. (1992) *J. Chem. Phys.*, 97(4), 2635.
- [7] Martyna, G. J.; Tuckerman, M. E.; Tobias, D. J.; Klein, M. L. (1996) *Mol. Phys.*, 87(5), 1117.
- [8] Bussi, G.; Donadio, D.; Parrinello, M. (2007) *J. Chem. Phys.*, 126(1), 014101.
- [9] Kumar, S.; Rosenberg, J. M.; Bouzida, D.; Swendsen, R. H.; Kollman, P. A. (1992) *J. Comput. Chem.*, 13(8), 1011.
- [10] Kästner, J.; Senn, H. M.; Thiel, S.; Otte, N.; Thiel, W. (2006) *J. Chem. Theory Comput.*, 2(2), 452.
- [11] Grimme, S. (2019) *J. Chem. Theory Comput.*, 15(5), 2847.
- [12] Berendsen, H. J. C.; Postma, J. P. M.; van Gunsteren, W. F.; DiNola, A.; Haak, J. R. (1984) *J. Chem. Phys.*, 81(8), 3684.
- [13] Box, G. E. P.; Muller, M. E. (1958) *Ann. Math. Statist.*, 29(2), 610.
- [14] Andersen, H. C. (1983) *J. Comput. Phys.*, 52(1), 24 .
- [15] Kutteh, R. (1998) *CCP5 Newsllett.*, 9.
- [16] Wilson, E. B.; Decius, J. C.; Cross, P. C. (1955) *Molecular Vibrations – The Theory of Infrared and Raman Vibrational Spectra*.