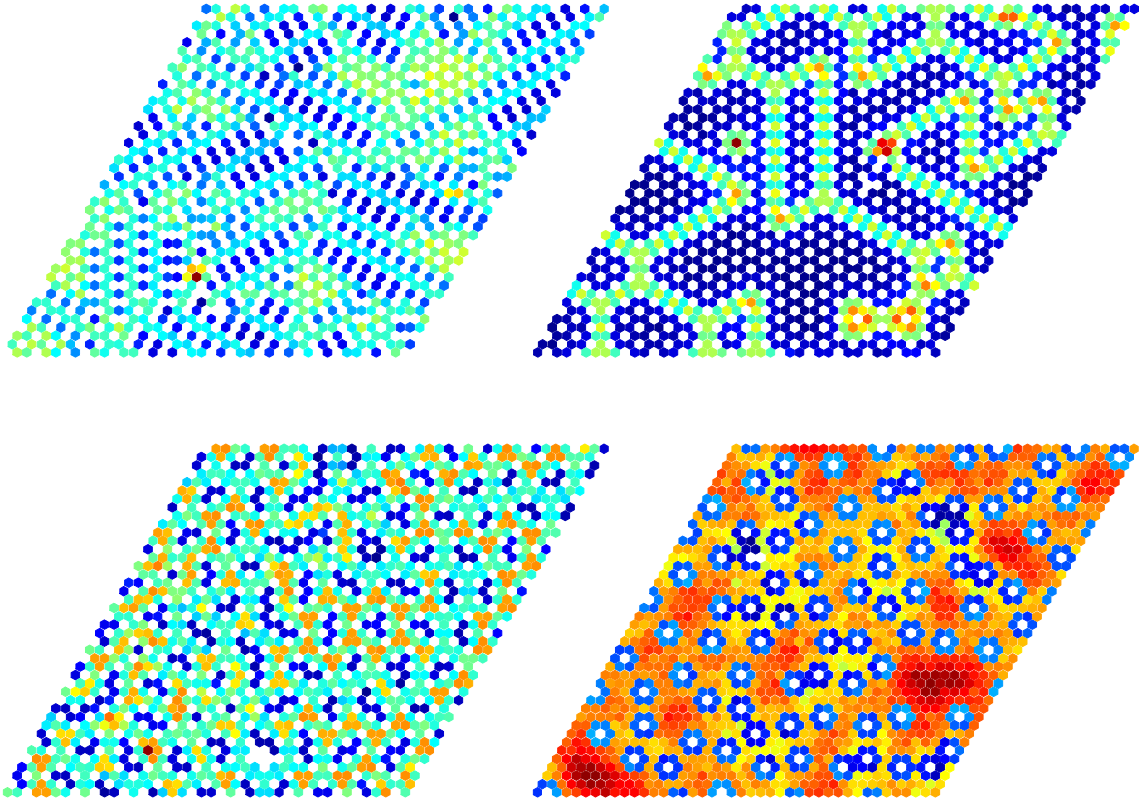
Zacros 1.02

Advanced Lattice-KMC Simulation Made Easy



User Guide

September 4, 2014

Dear Colleague,

I would like to thank you for downloading the Zacros package and I hope you will find this tool useful in your research. Zacros is an advanced kinetic Monte Carlo package for the simulation of molecular phenomena, such as adsorption and catalytic reactions, on surfaces. The package employs the Graph-Theoretical KMC methodology coupled with cluster expansion Hamiltonians for the adlayer energetics, which can naturally capture:

- steric exclusion effects for species that bind in more than one catalytic sites,
- complex reaction steps involving adsorbates in specific binding configurations and neighboring patterns,
- spatial correlations and ordering arising from adsorbate lateral interactions that may involve many-body contributions.

In addition to these, the code features an easy-to-learn keyword-based language for defining a simulation, and can be run in "debugging" mode, thereby generating detailed output that can be used to efficiently troubleshoot a KMC simulation.

Zacros is distributed free of charge to the academic community in the hope that it will benefit researchers worldwide. If you decide to use this software for a scientific article, I kindly ask you to include the following citations in your work:

> Stamatakis, M. and D. G. Vlachos (2011). "A Graph-Theoretical Kinetic Monte Carlo Framework for on-Lattice Chemical Kinetics." Journal of Chemical Physics **134**(21): 214115.
>
> Nielsen, J., M. d'Avezac, J. Hetherington and M. Stamatakis (2013). "Parallel Kinetic Monte Carlo Simulation Framework Incorporating Accurate Models of Adsorbate Lateral Interactions." Journal of Chemical Physics **139**(22): 224706.

I would be glad to receive feedback about Zacros, and if you would like to contribute to the development thereof, please don't hesitate to get in touch.

Kind regards,

Michail Stamatakis
Lecturer in Chemical Engineering
University College London
Torrington Place
London, WC1E 7JE
United Kingdom

Phone: 020 3108 1128
Fax: 020 7383 2348
e-mail: m.stamatakis@ucl.ac.uk
url: www.homepages.ucl.ac.uk/~ucecmst/

Contents

Introduction

Zacros is an advanced kinetic Monte Carlo (KMC) package for the simulation of molecular phenomena, such as adsorption and catalytic reactions, on structures that can be represented by static lattices. The package employs the Graph-Theoretical KMC methodology[1] coupled with cluster expansion Hamiltonians for the adlayer energetics,[2] allowing it to tackle:

- binding configurations on more than one sites, and the steric exclusion effect resulting therefrom,
- complex surface reactions in which several species and spectators can be involved in specific neighbouring patterns,
- adsorbate lateral interactions involving long-range and many body terms, and the spatial correlation and ordering effects resulting therefrom.

Moreover, OpenMP parallelization is implemented for the efficient simulation of systems with energetic models involving long-range interactions.

This user guide provides information about the syntax of input/output files and the options available in Zacros. For information on KMC simulation and the underlying methods implemented in the package, the user is referred to:

> Stamatakis, M. and D. G. Vlachos (2012). "Unraveling the Complexity of Catalytic Reactions via Kinetic Monte Carlo Simulation: Current Status and Frontiers." ACS Catalysis **2**(12): 2648-2663.
>
> Stamatakis, M. and D. G. Vlachos (2011). "A Graph-Theoretical Kinetic Monte Carlo Framework for on-Lattice Chemical Kinetics." Journal of Chemical Physics **134**(21): 214115.
>
> Nielsen, J., M. d'Avezac, J. Hetherington and M. Stamatakis (2013). "Parallel Kinetic Monte Carlo Simulation Framework Incorporating Accurate Models of Adsorbate Lateral Interactions." Journal of Chemical Physics **139**(22): 224706.

Compiling Zacros

The following instructions are for compiling Zacros with GFortran, the GNU Fortran compiler, which can be downloaded for free and works in Unix and Windows systems. Alternatively one can use CMake; for information on how to do this, please refer to the document *CompilingZacrosWithCMake.pdf*.

Compilation on Unix/Linux

This section refers to Unix and Linux operating systems. If your system does not come with the GFortran compiler you will have to install it. For instance, in Ubuntu Linux, you can do so by running the following command on a terminal: `sudo apt-get install gfortran`.

Compiling should be done using a terminal. It simply comes down to the following:

```
cd path/to/source/of/Zacros
mkdir build # If it does not exist yet
cd build
```

```
cp path/to/makefiles/makefile-gfortran-parallel-unix makefile
make
```

The first line above moves to the directory where the source code can be found. Then we create a build directory and move to that directory. All the build files will be located here, rather than "polluting" the source. We copy one of the makefiles provided to the build directory (renaming it to `makefile` at the same time) and compile the code. At the end of this process, there should be an executable called `zacros.x` in the build directory.

Finally, it is possible to compile a serial version of the code. This is of interest when running on a computer with a single core, or when simulating systems with no lateral interactions (please refer to section Energetics Input File). Simply replace (or rerun) line 4 with:

```
cp path/to/makefiles/makefile-gfortran-serial-unix makefile
```

Compilation on Mac OS/X

For Mac OS/X one can install GFortran from http://gcc.gnu.org/wiki/GFortranBinaries. If your system does not recognize the `make` command, you can install the XCode package making sure that the Command Line Tools are included in the installation. The rest of the compilation instructions are the same as in section Compilation on Unix/Linux.

Compilation on Windows

To compile Zacros in Windows you can install the GFortran compiler that can be downloaded from http://gcc.gnu.org/wiki/GFortranBinaries. The easiest option is probably the "Unofficial build of current development (4.8) source" under the section **MinGW build** ("native Windows" build).

Type `cmd.exe` and hit enter in the *"Search programs and files"* line, to bring up a command prompt. Then:

```
cd path\to\source\of\Zacros
md build # If it does not exist yet
cd build
copy path\to\makefiles\makefile-gfortran-parallel-windows makefile
make
```

The first line above moves to the directory where the source code can be found. Then we create a build directory and move to that directory. All the build files will be located here, rather than "polluting" the source. We copy one of the makefiles provided to the build directory (renaming it to `makefile` at the same time) and compile the code. At the end of this process, there should be an executable called `zacros.exe` in the build directory. One can also compile a serial version of the code by replacing line 4 with:

```
copy path\to\makefiles\makefile-gfortran-serial-windows makefile
```

Running Zacros

The simplest way to run Zacros is simply to launch the executable from the command-line. For Unix-like systems:

```
path/to/executable/zacros.x
```

or simply,

```
zacros.x
```

if `zacros.x` is in the user's path. For Windows systems `zacros.x` is replaced with `zacros.exe` in the above commands. Zacros expects all the right input files to be in the current directory. Please refer to section Setting up a KMC Simulation in Zacros for a description of these files.

When invoking the thread-capable executable (the default; see section Compiling Zacros for how to disable threads) Zacros will run with as many threads as there are cores. The number of threads can be manually defined in MS-DOS and UNIX by setting the appropriate environmental variables. In UNIX one needs to use the command `export`:

```
export OMP_NUM_THREADS=4
```

for 4 threads, whereas in MS-DOS this is done with command `set`:

```
set OMP_DYNAMIC=FALSE
set OMP_NUM_THREADS=4
```

Input/Output Files

The input to Zacros consists of 5 keyword-based files, out of which one is optional:

`simulation_input.dat` (non-optional)
`energetics_input.dat` (non-optional)
`lattice_input.dat` (non-optional)
`mechanism_input.dat` (non-optional)

`state_input.dat` (optional)

Moreover, the output of Zacros consists of the following files:

```
general_output.txt
lattice_output.txt
specnum_output.txt
history_output.txt
procstat_output.txt
```

If run in debugging mode (see debugging keywords in section Simulation Input File), Zacros also generates the following files which are useful for troubleshooting a simulation:

```
process_debug.txt
newton_debug.txt
globalenerg_debug.txt
```

All the aforementioned files are read from (or written to) the same directory.

In addition, `restart.inf` is an input/output file used to resume a simulation from the point it stopped. Even though it is a plain text file, it is not intended as human readable. For more details on how

the resume feature works, please refer to keywords `walltime` and `no_restart` in section Simulation Input File. Note that if the file `restart.inf` exists in the current working directory, Zacros will disregard the aforementioned input files, attempt to read `restart.inf` and resume the simulation.

Units and Constants

Zacros assumes that the input parameters are given in the following units:

Energy: electronvolt (eV) **Time**: second (s) **Length**: Ångstrom (Å)
Pressure: bar (bar) **Molecular mass**: atomic mass units (amu)* **Temperature**: Kelvin (K)

Moreover, the values of the constants used are as follows:

Pi constant: $\pi = 3.141592653589793$
Gas constant: $R_{gas} = 8.314472$ J/K/mol **Avogardro's number**: $N_A = 6.02214179 \cdot 10^{23}$ mol^{-1}
Boltzmann's constant: $k_B = R_{gas}/N_A$

For the conversion of J to eV the following constant is used:

EnergyConv = $6.24150974 \cdot 10^{18}$

One can use a different system of units for the time and pressure when providing preexponentials (see section Mechanism Input File), keeping in mind that the reported values will also have different units. However, using different units for energy and temperature will require changing the value of parameter `enrgconv` in file `constants_module.f90` and recompiling the program (see section Compiling Zacros for information on how to do this).

Setting up a KMC Simulation in Zacros

In the following we present the keywords and syntax used in each of the input files. Keywords are denoted with blue colored `Courier New` font, for instance `random_seed`. Numeric or string arguments to the keywords are denoted as follows:

int	an integer number
real	a real number
str	a string
keywrd	a keyword
expr	an expression which may consist of combinations of the above

If more than one arguments of the same kind follow a keyword, they appear numbered, for instance, `temperature ramp` *real1 real2*.

All input files support free-format; thus, blank lines and comments are permitted anywhere in the text

* This feature exists for future development and presently does not affect the input/output.

as long as the syntax is valid. The commenting character is #, for instance:

```
snapshots on time 1.50 # sampling every 1.5 time units
```

The keywords are not case sensitive and strings should be written free from quotation marks (unless for instance one wants to use quotation marks as part of a name). Spaces are not allowed in a string; one can use underscores _ instead. In general, the order of the keywords does not matter, but there are cases where a keyword must precede another one, for instance one has to first define the number of gas species and subsequently the names thereof, not the reverse. Moreover, keywords may not be repeated within the same scope. The parser will report an error is such cases.

Simulation Input File

The file `simulation_input.dat` contains information about the species involved in the chemistry, the conditions under which the chemistry is to be simulated, as well as parameters that specify the behavior of the program, namely when to take samples, what are the stopping criteria, etc. Common keywords and options are explained below.

`random_seed` *int*	The integer seed of the random number generator.
`tol_dx_newton` *real*	When simulating systems in which the temperature is not constant Zacros uses the Newton-Raphson method to solve a non-linear equation for the time of occurrence for each lattice process. The value of *real* gives the tolerance for the norm between subsequent approximations to a solution. If omitted, this tolerance is taken equal to the default value of 10^{-9}.
`tol_rhs_newton` *real*	See also keyword `tol_dx_newton` above. This keyword gives the tolerance for the right hand side for the Newton-Raphson method. If omitted, this tolerance is taken equal to the default value of 10^{-9}.
`max_newton_iter` *int*	See also keyword `tol_dx_newton` above. This keyword gives the maximum number of interactions for the Newton-Raphson method. If omitted, this number is taken equal to the default value of 150.
`n_gauss_pts` *int*	See also keyword `tol_dx_newton` above. The non-linear equation for the time of occurrence for each lattice process contains an integral of propensity over time. This integral is computed using the Gauss-Legendre quadrature for which this keyword gives the number of points to be used.
`temperature` *expr*	The temperature (K) under which the system is simulated. Expression *expr* can be one of the following:

real specifies a constant simulation temperature

ramp *real1 real2* specifies a temperature ramp where *real1* is the initial temperature (K) and *real2* is the rate of change (K/s). If *real2* is positive, temperature programmed desorption can be simulated. Negative values of *real2* can be used for simulated annealing calculations.

pressure *real* The pressure (bar) under which the system is simulated.

n_gas_species *int* The number of gas species in the chemistry.

gas_specs_names *str1 str2 …* The names of the gas species. There should be as many strings following the keyword as the number of gas species specified with keyword n_gas_species.

gas_energies *real1 real2 …* The total energies (eV) of the gas species. There should be as many reals following this keyword as the number of gas species specified with keyword n_gas_species. The ordering of these values should be consistent with the order used in keyword gas_specs_names.

gas_molec_weights *real1 real2 …* The molecular weights (amu) of the gas species. There should be as many reals following the keyword as the number of gas species specified with keyword n_gas_species. Note: at present these values are not used in the code. This feature is there for future development.

gas_molar_fracs *real1 real2 …* The molar fractions (dim/less) of the gas species in the gas phase. There should be as many reals following this keyword as the number of gas species specified with keyword n_gas_species. The ordering of these values should be consistent with the order used in keyword gas_specs_names.

n_surf_species *int* The number of surface species in the chemistry (without counting the empty sites as a species, even though it is considered as a pseudo-species internally in the code).

surf_specs_names *str1 str2 …* The names of the surface species. There should be as many strings following the keyword as the number of surface species specified with keyword n_surf_species. Note that the name "*" is reserved for the empty site pseudo-species.

`surf_specs_dent` *int1 int2 …* The number of dentates of the surface species, specifying the number of sites each species binds to. Thus, for a monodentate species (for instance O adatoms on fcc sites) this integer is 1, for a bidentate species (for instance O_2 on a top-fcc configuration) this integer is 2, etc. There should be as many integers following this keyword as the number of surface species specified with keyword `n_surf_species`. The ordering of these values should be consistent with the order used in keyword `surf_specs_names`.

`snapshots` *expr* Determines how often snapshots of the lattice state will be written to output file `history_output.txt`. Possible options for expression *expr* are:

- `off` switches off snapshots output. No configurations will be written.
- `on event` [*int*] specifies that a snapshot will be written at every *int* KMC steps. The integer following `on event` is optional, and assumes the value of 1 if omitted. In the latter case, the initial (KMC step 0) and all subsequent configurations will be written.
- `on time` *real* specifies that a snapshot will be written at linearly spaced time points, at every Δt = *real* time units (s): 0, Δt, $2\cdot\Delta t$, $3\cdot\Delta t$, …
- `on logtime` *real1 real2* specifies that a snapshot will be written at logarithmically spaced time points, starting at time t_0 = *real1* and progressing by multiplying by a = *real2*: t_0, $a\cdot t_0$, $a^2\cdot t_0$, $a^3\cdot t_0$, … This sampling scheme is particularly useful if one needs to investigate a vast range of timescales, as it overcomes the problem of generating huge output files.

`process_statistics` *expr* Determines how often statistical information about the occurrence of elementary events will be written to output file `procstat_output.txt`. Possible options for expression *expr* are:

- `off` switches off process statistics output. No information will be written.

`on event` [*int*] specifies that an entry to the output file `procstat_output.txt` will be generated at every *int* KMC steps. The integer following `on event` is optional, and assumes the value of 1 if omitted.

`on time` *real* specifies that an entry to the output file `procstat_output.txt` will be generated at linearly spaced time points, at every Δt = *real* time units (s): 0, Δt, $2\cdot\Delta t$, $3\cdot\Delta t$, …

`on logtime` *real1* *real2* specifies that an entry to the output file `procstat_output.txt` will be generated at logarithmically spaced time points, starting at time t_0 = *real1* and progressing by multiplying by a = *real2*: t_0, $a\cdot t_0$, $a^2\cdot t_0$, $a^3\cdot t_0$, … This sampling scheme is particularly useful if one needs to investigate a vast range of timescales, as it overcomes the problem of generating huge output files.

`species_numbers` *expr* Determines how often information about the number of gas and surface species, as well as the energy of the current lattice configuration) will be written to `specnum_output.txt`. Possible options for expression *expr* are:

`off` switches off species numbers output. No information will be written.

`on event` [*int*] specifies that an entry to the output file `specnum_output.txt` will be generated at every *int* KMC steps. The integer following `on event` is optional, and assumes the value of 1 if omitted.

`on time` *real* specifies that an entry to the output file `specnum_output.txt` will be generated at linearly spaced time points, at every Δt = *real* time units (s): 0, Δt, $2\cdot\Delta t$, $3\cdot\Delta t$, …

`on logtime` *real1* *real2* specifies that an entry to the output file `specnum_output.txt` will be generated at logarithmically spaced time

points, starting at time t_0 = *real1* and progressing by multiplying by a = *real2*: t_0, $a \cdot t_0$, $a^2 \cdot t_0$, $a^3 \cdot t_0$, … This sampling scheme is particularly useful if one needs to investigate a vast range of timescales, as it overcomes the problem of generating huge output files.

`event_report` *expr*

Controls event reporting behavior. Expression *expr* can be one of the following:

`off` switches off event reporting. No information will be written.

`on` switches on event reporting. At every KMC step, information about the elementary process that was just executed will be written to output file `general_output.txt`.

`max_steps` *expr*

The maximum number of KMC steps to be simulated. This keyword defines a stopping criterion. Expression *expr* can be one of the following:

`infinity` sets the maximum number of steps to the maximum value of a 4-byte integer number (2147483647 ≈ $2.15 \cdot 10^9$).

int specifies a number of maximum steps.

`max_time` *expr*

The maximum allowed simulated time interval (time ranges from 0.0 to the maximum time in a simulation). This keyword defines a stopping criterion. Expression *expr* can be one of the following:

`infinity` sets the maximum time to the maximum value of an 8-byte real number (about $1.8 \cdot 10^{308}$).

int specifies the maximum time.

`wall_time` *int*

The maximum allowed real-world time in seconds that the simulation can be left running. The code has an internal "stopwatch" and will exit normally once the walltime has been exceeded. Upon exit, the state of the program will be saved in file `restart.inf`, so that the simulation can resume at a later time. This feature is particularly useful when running in

computational clusters where a scheduler may enforce limits on the time a simulation can be run.

`no_restart` This keyword gives the option to override the default behavior of the program and not produce any `restart.inf` file upon exit. If this has been specified, the program will not be able to resume the simulation at a later time.

`finish` This keyword marks the end of input. Any subsequent information will not be parsed.

In addition to the aforementioned keywords there are some "debugging" keywords that may prove particularly useful for troubleshooting a KMC simulation. These keywords trigger internal checks or enable the output of detailed information in human-readable format, allowing the user to see "what is going on" during the simulation. Note that these debugging procedures will significantly slow down execution and/or result in large output files, so they should only be used in short runs (not for results production).

`debug_report_global_energetics` Triggers the output of information pertinent to the data-structures storing the energetic pattern contributions to the total lattice energy. The information is written to file `globalenerg_debug.txt` and includes: (i) the detection of new energetic clusters initialization of the simulation or whenever new species appear in the lattice after a KMC step, (ii) the deletion of energetic clusters whenever species disappear in the lattice after a KMC step, (iii) the re-indexing of an energetic cluster when another cluster in the "middle" of the stack is being deleted, so that no "holes" exist in the data-structures, (iv) the total lattice energy at every KMC step (for more details see section Energetics Debug Output File).

`debug_report_processes` Triggers the output of information pertinent to the queue of KMC lattice processes in debugging-output file `process_debug.txt`. Information written therein includes: (i) the detection of new elementary processes upon initialization of the simulation or whenever new species appear in the lattice after a KMC step, (ii) the deletion of elementary processes whenever species disappear in the lattice after a KMC step, (iii) the re-indexing of an elementary processes when another process in the "middle" of the queue is being deleted, so that no "holes" exist in the queuing data-structures, (iv) the update in the rates of elementary processes, as a result of energetic interactions emerging from

newly appearing species in the lattice (for more details see section Process Debug Output File).

`debug_newtons_method` Triggers the output of information pertinent to the Newton-Raphson method, when simulating systems in which the temperature is not constant. The subsequent approximations to the solution along with convergence information are written to file `newton_debug.txt`.

`debug_check_processes` Triggers an internal check that verifies the self-consistency of the data-structures pertinent to the KMC processes being queued and executed. If a problem is found, the program produces an error and terminates. **Please notify us immediately if you encounter such an error, as it means that there may be a bug in the simulator itself.**

`debug_check_lattice` Triggers an internal check that verifies the self-consistency of the data-structures representing the lattice state. If a problem is found, the program produces an error and terminates. **Please notify us immediately if you encounter such an error, as it means that there may be a bug in the simulator itself.**

Lattice Input File

The file `lattice_input.dat` defines the lattice structure on which species can bind, diffuse and react. There are 3 different ways of specifying a lattice structure as discussed in the following.

`lattice` *`expr`*

⋮

`end_lattice`

Defines a lattice specification block. Expression *`expr`* can be one of the following:

`default_choice` allows the user to specify one of the available default lattices (explained in section Default Lattices below).

`periodic_cell` allows the user to construct a lattice by giving information about the unit cell (explained in section Unit-Cell-Defined Periodic Lattices below).

`explicit` allows the user to import a custom (possibly non-periodic) lattice structure generated manually or by a different program (explained in section Explicitly Defined Custom Lattices below).

The permitted keywords for each of the aforementioned options are discussed in the following.

Default Lattices

Currently there are three possible default lattices, all of which are periodic with coordination numbers equal to 3, 4 and 6. In these lattices all sites are equivalent (single site type). The name of this site type is by default `StTp1`. Inside a default lattice block (`lattice default_choice` … `end_lattice`) the following keywords are allowed (Figure 1):

`triangular_periodic` *`real int1 int2`* Specifies a lattice with coordination number 3. The real number defines the lattice constant whereas the two integers give the number of copies of the unit cell in the horizontal and vertical directions, respectively for *`int1`* and *`int2`*. Note that the unit cell for this default lattice is not the primitive unit cell, and it contains 4 sites.

`rectangular_periodic` *`real int1 int2`* As above for a lattice with coordination number 4. For this lattice, the unit cell is the primitive cell.

`hexagonal_periodic` *`real int1 int2`* As above for a lattice with coordination number 6. For this lattice, the unit cell is not the primitive unit cell, and it contains 2 sites.

Unit-Cell-Defined Periodic Lattices

Zacros allows the user to define custom periodic lattices by providing information about the unit cell geometry, the sites contained therein, and the neighboring relations between sites in the same cell as well as neighboring cells. Inside a periodic lattice block (`lattice periodic_cell` … `end_lattice`) the following keywords are allowed:

`cell_vectors`
`real1 real2`
`real3 real4`

This keyword is followed by two pairs of reals in two subsequent lines, as shown in the left, which define the unit vectors. The two unit vectors are thus α = (*`real1`*, *`real2`*) and β = (*`real3`*, *`real4`*).

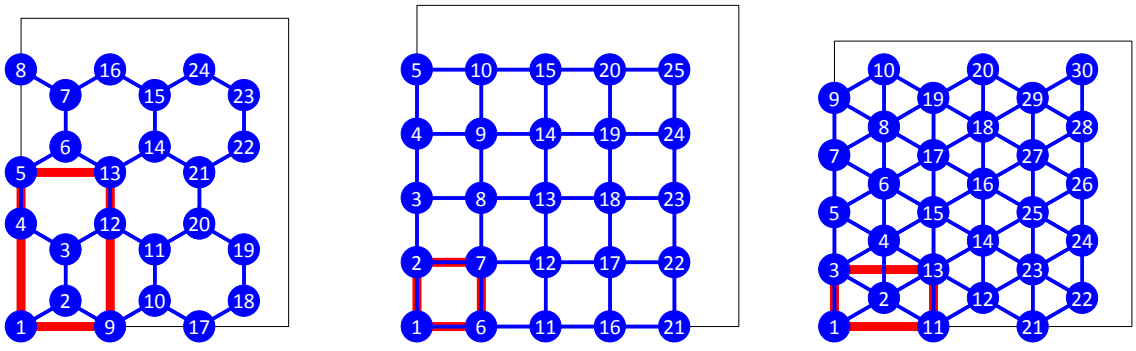
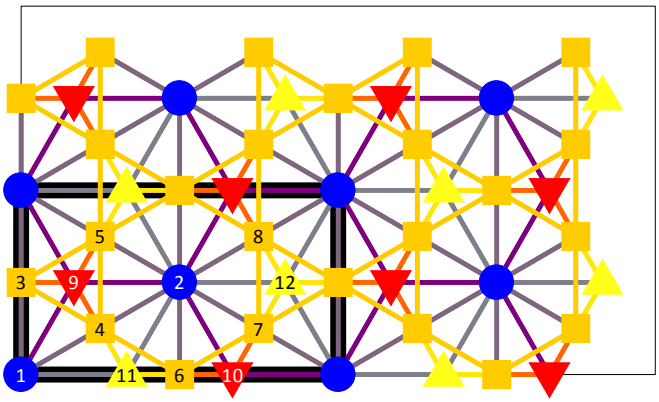


Figure 1: Default lattices in Zacros. Left: triangular (coordination number, CN = 3). Middle: rectangular (CN = 4). Right: hexagonal (CN = 6). The blue lines connect the 1st nearest neighbors of the lattice. The black lines denote the simulation box and the thick red lines the unit cell.



●	1	top
■	2	brg
▼	3	fcc
▲	4	hcp

```
top top brg brg brg brg brg brg fcc fcc hcp hcp
```

Figure 2: Lattice representing the (111) surface of an FCC metal, for instance Pt(111). Numbered are only the sites belonging to one unit cell, which is denoted by thick black lines. The table on the right shows the 4 different sites types, along with the index and name of each one. Given on the bottom are the two equivalent expressions that define the types of all sites within the unit cell.

`repeat_cell` *int1 int2*
: The number of repetitions of the unit cell in the directions of unit vectors α and β, respectively for *int1* and *int2*.

`n_cell_sites` *int*
: The total number of sites (of any site type) in the unit cell.

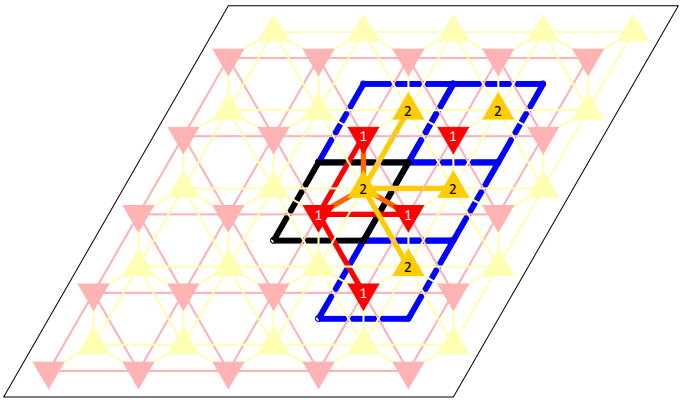
`n_site_types` *int*
: The number of different site types. For instance, if one needs to model a lattice for the Pt(111) surface taking into account top, bridge, fcc and hcp sites, then *int* should be 4 (see also Figure 2).

`site_type_names` *str1 str2 …*
: The names of the different site types. There should be as many strings following this keyword as the number of site types specified by `n_site_types`. In the example just mentioned for the Pt(111) surface lattice, the expression used can be: `site_type_names top brg fcc hcp` (see Figure 2).

`site_types` *expr*
: The site types for each of the different sites of the unit cell. Expression *expr* can consist of as many strings or integers as the number of site types specified by `site_type_names`. Thus, there are two options for *expr* (see Figure 2):

  `int1 int2 …` expresses the site types in terms of their indexes.

  `str1 str2 …` expresses the site types in terms of their names as specified by `site_type_names`.




```
1-2  self
1-1  north
1-1  east
1-1  southeast
2-1  north
2-2  north
2-1  east
2-2  east
2-2  southeast
```

Figure 3: Lattice representing the (111) surface of an FCC metal, with the fcc and hcp sites taken into account. Numbered are only the sites belonging to the central unit cell, as well as the north, northeast, east, and southeast neighboring unit cells (N, NE, E, SE, respectively). Within each cell, site 1 is the fcc, whereas the hcp is site 2. The links of sites of the central unit cell with sites in neighboring cells are depicted. The neighboring structure that gives rise to this lattice is shown on the right.

```
site_coordinates
real1      real2
real3      real4
   ⋮         ⋮
```

This keyword is followed by lines containing pairs of real numbers specifying the "fractional coordinates" of each site in the unit cell, as shown on the left. There should be as many lines as the number of sites in the unit cell, specified by `n_cell_sites`. The "fractional coordinates" are with respect to the unit cell vectors α and β defined by `cell_vectors`. Thus, the Cartesian coordinates of site 1 will be *real1*·α + *real2*·β, those of site 2 *real3*·α + *real4*·β, etc.

```
neighboring_structure

int1-int2 keyword
   ⋮

end_neighboring_structure
```

Defines a neighboring structure block containing an arbitrary number of expressions formed by two integers separated by a dash and a keyword (see Figure 3 for an example). The latter can be one of the following: `self`, `north`, `northeast`, `east`, `southeast`. Each of these expressions specifies a "link" between two sites, making them 1st nearest neighbors. For example, if sites 1 and 2 in the same unit cell need to be specified as neighbors, the expression to be used is: `1-2 self`. Note that the order in this case does not matter; thus the expression just noted is equivalent to `2-1 self`. Moreover, if site 1 neighbors with its own image on the unit cell above, the expression would be `1-1 north`. Note that if different sites are defined as neighbors across unit cells, the order matters, namely `1-2 northeast`, is not equivalent to `2-1 northeast`.

Explicitly Defined Custom Lattices

Zacros can also accept custom lattice structures which may be created manually. Inside an explicit lattice block (`lattice explicit` … `end_lattice`) the following keywords are allowed (see also Figure 4 for an example of a custom lattice):

`cell_vectors`
real1 *real2*
real3 *real4*

This keyword is followed by two pairs of reals in two subsequent lines, as shown in the left, which define the unit vectors. The two unit vectors are thus α = (*real1*, *real2*) and β = (*real3*, *real4*). Unlike the unit-cell-defined periodic lattice, the `cell_vectors` keyword here is optional, used only if the lattice defined is intended as periodic. Moreover, the site coordinates inside the `lattice_structure` block (see below) are always given as Cartesian coordinates for an explicit lattice (irrespective of periodicity).

`n_sites` *int*

The total number of sites in the entire lattice.

`max_coord` *int*

The maximum coordination number, namely the maximum number of 1st nearest neighbors, for any of the lattice sites.

`n_site_types` *int*

The number of different site types (as previously; see section Unit-Cell-Defined Periodic Lattices).

`site_type_names` *str1 str2* …

The names of the different site types. There should be as many strings following this keyword as the number of site types specified by `n_site_types` (as previously; see section Unit-Cell-Defined Periodic Lattices).

`lattice_structure`
expr1
expr2
⋮
`end_lattice_structure`

This keyword is followed by expressions *expr1*, *expr2*,… containing information about each and every site of the lattice. Thus, there should be as many expressions as the number of sites in the entire lattice, specified by `n_sites`. Each of these expressions has the following form (see also Figure 4 for an illustrative example):

int1 real1 real2 int2 int3 int4 int5 …

where:

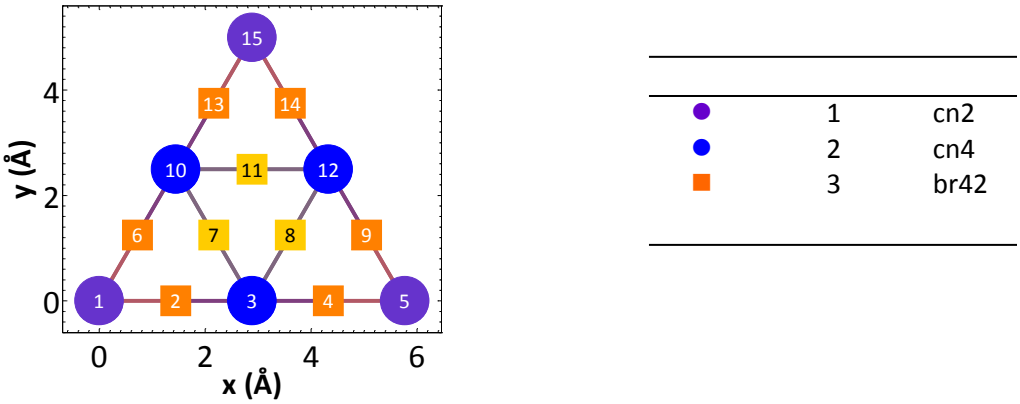
int1 is the index of the site (ranging from 1 to the number of sites specified by `n_sites`)

real1 and *real2* are the x and y Cartesian coordinates of the site with index *int1*.

int2 gives the site type of the site with index *int1*. In place of this integer, one can also use a string denoting a site type name as specified by keyword site_type_names as done in Figure 4 (see also the description of site_types in section Unit-Cell-Defined Periodic Lattices).

int3 gives the coordination number of the site with index *int1*.

int4 int5 … give the 1st nearest neighbors of the site with index *int1*. There should be as many neighbors listed as the value of *int3*. If there are more integers listed in this line, the extra entries will be ignored.




```
lattice_structure # Au6 nanocluster structure
  1    0.0000e+0  0.0000e+0    cn2     2     2     6
  2    1.4425e+0  0.0000e+0    br42    2     1     3
  3    2.8850e+0  0.0000e+0    cn4     4     2     4     7     8
  4    4.3275e+0  0.0000e+0    br42    2     3     5
  5    5.7700e+0  0.0000e+0    cn2     2     4     9
  6    7.2125e-1  1.2492e+0    br42    2     1    10
  7    2.1637e+0  1.2492e+0    br44    2     3    10
  8    3.6062e+0  1.2492e+0    br44    2     3    12
  9    5.0487e+0  1.2492e+0    br42    2     5    12
 10    1.4425e+0  2.4985e+0    cn4     4     6     7    11    13
 11    2.8850e+0  2.4985e+0    br44    2    10    12
 12    4.3275e+0  2.4985e+0    cn4     4     8     9    11    14
 13    2.1637e+0  3.7477e+0    br42    2    10    15
 14    3.6062e+0  3.7477e+0    br42    2    12    15
 15    2.8850e+0  4.9970e+0    cn2     2    13    14
end_lattice_structure
```

Figure 4: Lattice representing the surface of a Au_6 nanocluster, along with the corresponding lattice_structure *definition. For this example,* n_sites *would be set to 15 and* max_coord *to 4 (the maximum of column 5).*

How to Determine Lattice Connectivity

At this point, one might wonder: what is the main consideration when defining the neighboring structure for a lattice? For instance in the lattice of Figure 4, why did we not explicitly define the bridge sites (br42 and br44) to be neighbors with each other? The answer to that lies in the way species bind to the lattice sites and the types of reactions that can occur between these species. As rules of thumb:

- If multidentate species are present in the chemistry under investigation, the sites onto which each dentate binds have to be neighbors with each other. For instance, a carbonate species (CO_3) can bind on Au_6 in a top-bridge-top configuration involving sites cn2-br42-cn4 (see lattice of Figure 4).[3] Thus, these sites have been defined as neighboring in this lattice structure.

- For reactions that occur between adsorbed particles, there have to be one or more links between the sites occupied by the different reactant molecules. Thus, on Au_6 (Figure 4), representing a reaction between CO bound to cn2 and O_2 bound to cn4 in the absence of any adparticle on br42, necessitates a neighboring relation between cn2-br42-cn4. This reaction would give CO_2(gas) and a left-over O adatom at br42.[3]

Energetics Input File

The file `energetics_input.dat` defines a cluster expansion Hamiltonian[4] to be used for calculating the energy of a given lattice configuration. The syntax used in this file is discussed in the following.

```
energetics
   ⋮
 expr
   ⋮
end_energetics
```

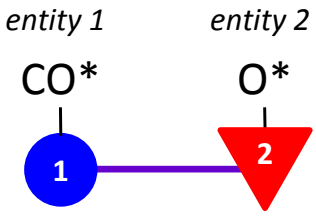
Defines an energetics specification block. Anything after the keyword `end_energetics` is ignored. The energetics specification block contains expressions consisting of one or several blocks structured as `cluster` … `end_cluster` (explained below). Each of the latter defines a cluster (also referred to as figure or pattern) in the cluster expansion Hamiltonian.

```
cluster str
   ⋮
 expr
   ⋮
end_cluster
```

Defines a cluster in the Hamiltonian. String *str* is a descriptive name of the cluster. There is no limitation to how many such "cluster definition" blocks can be contained in an energetics specification. Permitted keywords within this block will be presented shortly. Before doing so, however, let us briefly discuss how clusters are represented in Zacros.

Cluster Representation

Each cluster is represented as a graph pattern. The latter consists of a collection of connected sites onto which surface species can bind. In order to "translate" a pattern into input that Zacros can process, it is instructive to make drawings such as the ones in Figure 5. The pattern on the top left (CO-O interaction) can be used to model the repulsive interaction between an adsorbed CO molecule on a top site and an O adatom on an fcc site, for example on Pt(111) . This pattern involves two monodentate species bound to neighboring sites of different types. The pattern on the bottom left (Bidentate O_2) represents the



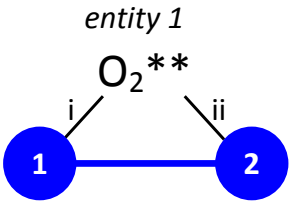
CO-O Interaction

Species

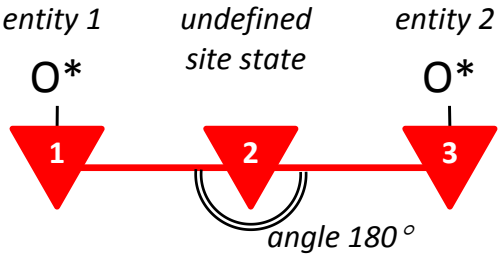
Name	Index	Dentates
CO*	1	1
O*	2	1
O_2**	3	2

Site Types

Marker	Index	Name
●	1	top
▼	2	fcc



Bidentate O_2



O-O Interaction 3rd NN

Figure 5: Schematics of various graph patterns representing energetic contribution terms (clusters) in a cluster expansion Hamiltonian. The white numbers represent the indexes of each site of the pattern. The lowercase roman numbers show the dentates of the bidentate oxygen.

bidentate binding configuration of O_2. Finally, the pattern on the bottom right (O-O Interaction 3rd NN) can model the interaction between O adatoms on Pt(111), for instance. It involves two monodentate adsorbates and three sites, the second of which can be empty or occupied by another species. Moreover, it involves a geometric criterion, as the angle between the links 1-2 and 2-3 has to be 180° for sites 1 and 3 to be 3rd nearest-neighbors (for 2nd nearest-neighbors this angle is 120°; see also Figure 3). To represent patterns such as these, Zacros provides a number of keywords discussed below.

`sites` *int1*

Specifies the number of sites in the graph pattern representing the cluster.

`neighboring` *int1*`-`*int2* …

Specifies the neighboring between sites, if more than one sites appear in the graph pattern. It is followed by expressions structured as *int1*`-`*int2* in the same line of input as the keyword. Each such expression denotes that the sites with indexes *int1* and *int2* are nearest neighbors. The values of *int1*, *int2*, … range from 1 up to the number of sites specified by the keyword `sites`. There can be as many such expressions as needed to fully define the neighboring structure of the pattern.

`lattice_state`

expr
⋮

Specifies the state of each site in the graph pattern representing the cluster. It is followed by as many lines as the number of sites specified by the keyword `sites`. Each one of these (non-blank) lines contains an expression specifying the

state of a site: the first line corresponds to the site indexed 1 in the pattern, the second line to site 2 etc. Note that there is no closing keyword for `lattice_state`; the program exits this input mode once the appropriate number of such lines has been parsed. Each expression *expr* conforms to one of the two following standards:

int1 str int2 The first argument *int1* is the number of the molecular entity bound to that site. Thus, if a bidentate species is bound to sites 1 and 3, both of these sites will have the same integers in the first column. The second argument *str* gives the name of the species bound to the site. Permitted surface species names are those defined previously with keyword `surf_specs_names` (see section Simulation Input File). Finally the third and last argument gives the dentate number with which the species is bound. For sites occupied by monodentate species, this number will always be equal to 1.

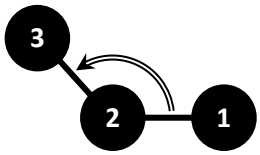
`& & &` For this specification all three columns contain the ampersand character. This denotes an unspecified state for that site.

`site_types` *str1 str2 …* The types of each and every site in the pattern. There should be as many strings following this keyword as the number of sites in the pattern specified by `sites`. This keyword is optional. If omitted, the pattern will be detected based on criteria pertinent to site occupancy, neighboring and geometry (if applicable) only.

`graph_multiplicity` *int* The multiplicity of the pattern, namely the number of times that the exact same pattern will be counted for a given lattice configuration. This keyword is followed by an integer and can be thought of as a symmetry number for the pattern (see also the description of keyword `cluster_eng` below). It is an optional keyword. Omitting the keyword is equivalent to specifying a value equal to 1 for *int*.

`cluster_eng` *real* The energy contribution of the pattern, given as a real number following the keyword. If `graph_multiplicity` is greater

than 1 for this pattern, the energy contribution is divided by the (integer) multiplicity.



`angles` *int1-int2-int3*`:`*real1* … Specifies a geometric criterion based on the angle between two links connecting pairs of three sites. There can be as many expressions following the keyword `angles` as needed, provided they appear on the same line. The integers *int1*, *int2* and *int3* denote three sites s_1, s_2 and s_3, out of which s_1 neighbors with s_2, and s_2 neighbors with s_3. Then, the value of *real* specifies the angle in degrees between vectors $s_2 \rightarrow s_1$ and $s_2 \rightarrow s_3$. Note that by default, mirror images of patterns are detected when Zacros calculates the total energy of a given lattice configuration. Thus, one does not need to explicitly define such mirror images unless the default behavior is overridden as discussed below.

`no_mirror_images` Overrides the default behavior of the program thereby preventing mirror image pattern detection. By default Zacros detects mirror images by looking for patterns which have angle values opposite from the ones specified in `angles` (for the same site indexes). For instance, if a pattern is specified with `angles 4-5-1:60`, Zacros will also search for patterns having `angles 4-5-1:-60` and all other properties the same as the original pattern. The presence of the keyword `no_mirror_images` restricts the search to the original pattern only.

`absl_orientation` *int1-int2*`:`*real* Specifies a geometric criterion based on the angle between the x-axis and a link between a pair of sites. The integers *int1* and *int2* denote the neighboring sites. The value of *real* specifies the angle in degrees between vector $s_2 \rightarrow s_1$ and the unit vector (1,0) in Cartesian coordinates. This keyword can be combined with keywords `angles` and `no_mirror_images` for a precise definition of the figures/patterns entering the cluster expansion Hamiltonian.

`variant` *str*
⋮
expr
⋮
`end_variant`

To reduce repetitions in the `energetics_input.dat` file, the `variant` blocks can be of particular use. Right after the `cluster` keyword one can thus define the number of sites (`sites`), neighboring structure (`neighboring`) and state of each site (`lattice_state`). Then, within that `cluster` …

end_cluster block, one can define several variants that will all share the same lattice structure and occupancies, but may vary in their geometry or site types. The name of the variant pattern consists of string *str* appended to the name of the parent pattern (*str* following the keyword cluster). In this respect, the following keywords are permitted within a variant block: site_types, graph_multiplicity, angles, no_mirror_images, absl_orientation, cluster_eng. If any of these keywords has been listed within a cluster block before a variant block has been opened, the keyword variant is no longer permitted within that cluster block.

Examples

As guiding examples, we finally give the Zacros input defining the clusters presented in Figure 5.

```
cluster CO-O_Interaction        # Opening a cluster block
  sites 2                       # There are two sites in the pattern…
  neighboring 1-2               #  that are neighbors
  lattice_state
    1 CO*   1                   # 1st site occupied by CO* (monodentate)
    2 O*    1                   # 2nd site occupied by O* (monodentate)
  site_types top fcc            # Specifying site types in the pattern…
  cluster_eng  0.140            #  and the energy contribution thereof
end_cluster                     # Closing the cluster block
```

```
cluster Bidentate_O2            # Opening a cluster block
  sites 2                       # There are two sites in the pattern…
  neighboring 1-2               # that are neighbors
  lattice_state
    1 O2**   1                  # 1st site occupied by 1st dentate of O2**
    1 O2**   2                  # 2nd site occupied by 2nd dentate of O2**
  site_types top top            # Specifying site types in the pattern…
  cluster_eng -0.203            # and the energy contribution thereof
end_cluster                     # Closing the cluster block
```

```
cluster O-O_Interaction         # Opening a cluster block
  sites 3                       # There are three sites in the pattern
  neighboring 1-2 2-3           # Site 2 neighbors with sites 1 and 3…
  lattice_state                 # (neighboring of 1-3 not yet precluded)
    1 O*   1                    # 1st site is occupied by O*
    & &    &                    # 2nd site's state is unspecified
    2 O*   1                    # 3rd site is occupied by O*
  variant 3rdNN                 # Defining variant O-O_Interaction_3rdNN
```

```
    site_types fcc fcc fcc     # Specifying site types in the pattern
    graph_multiplicity 2       # Multiplicity = 2 due to symmetry
    angles 1-2-3:180.0         # Geometric criterion for 3rd NN only:…
                               #   at this point the neighboring of 1-3
                               #  is ruled out!
    cluster_eng -0.016         # Defining cluster's energy contribution
  end_variant                  # Closing the variant block
end_cluster                    # Closing the cluster block
```

Mechanism Input File

The file `mechanism_input.dat` defines a reaction mechanism, consisting of one or more reversible and/or irreversible elementary steps. These steps can represent adsorption of molecules on surface sites, desorption therefrom, diffusion from one site to a neighboring site, or reactions between adsorbed particles and possibly gas species (Eley-Rideal reactions). The syntax used in this file is discussed in the following.

```
mechanism
   ⋮
 expr
   ⋮
end_mechanism
```

Defines a mechanism specification block. Anything after the keyword `end_mechanism` is ignored. The mechanism specification block contains expressions consisting of one or several blocks structured as either `step` … `end_step`, or `reversible_step` … `end_reversible_step` (more details follow). These blocks define irreversible or reversible elementary steps, respectively.

```
step str
   ⋮
 expr
   ⋮
end_step
```

Defines an irreversible elementary step in the mechanism. String *str* is a descriptive name of the step. There is no limitation to how many such "step definition" blocks can be contained in a mechanism specification.

```
reversible_step str
   ⋮
 expr
   ⋮
end_reversible_step
```

In a similar manner, this block defines a reversible elementary step in the mechanism. Thus, both the forward and reverse steps will be taken into account in the KMC simulation. String *str* is a descriptive name of this reversible step. The forward and backward steps are named by appending the strings "`_fwd`" and "`_rev`" to *str*. There is no limitation to how many such "step definition" blocks can be contained in a mechanism specification.

Permitted keywords within the two blocks just mentioned will be presented shortly. Before doing so, however, let us briefly discuss how elementary steps of a reaction mechanism are represented in Zacros.

Site Types

Marker	Index	Name
●	1	top
▼	2	fcc
■	3	brg

Species

Name	Index	Dentates
*	0	1
CO*	1	1
O*	2	1
O_2***	3	3

← *empty site regarded as a pseudo-species*

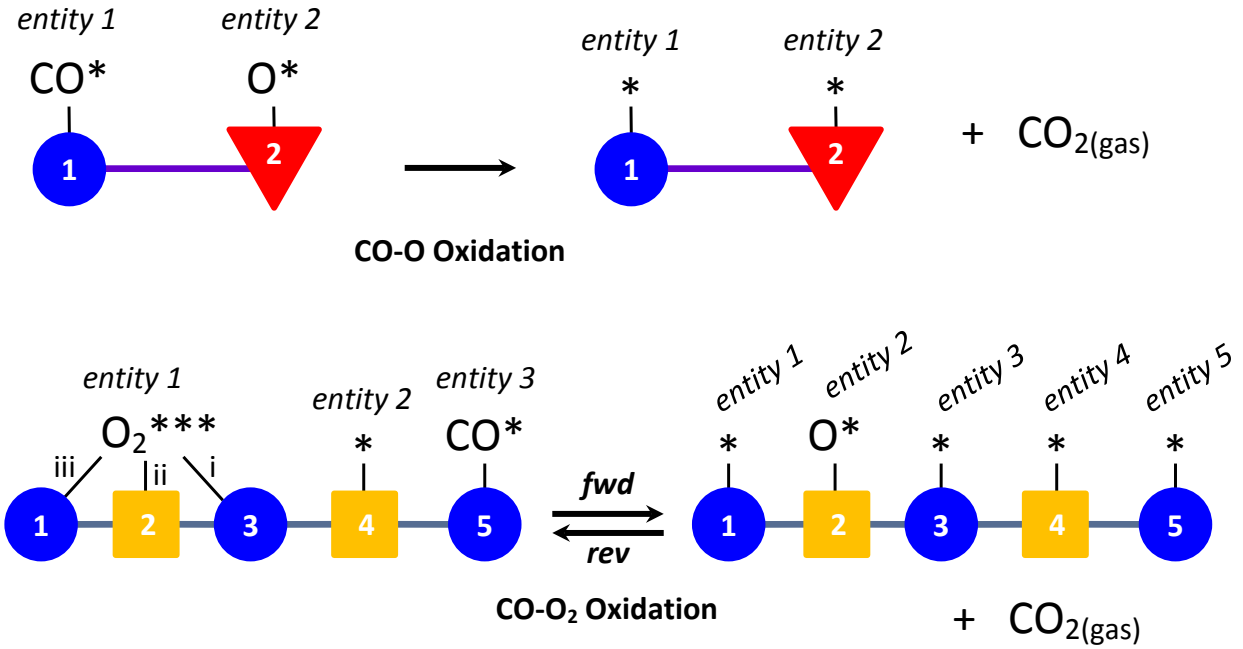


Figure 6: Schematics of various graph patterns representing elementary steps of a reaction mechanism. The white numbers represent the indexes of each site of the pattern. The lowercase roman numbers show the dentates of the tridentate oxygen.

Elementary Step Representation

As in the case of figures in a cluster expansion Hamiltonian, each elementary step is represented as a graph pattern, with specific initial and final states. In order to "translate" an elementary step into input that Zacros can process, it is instructive to make drawings such as the ones in Figure 6. Note that in these patterns the number of sites and the neighboring structure remain static (no reconstructions). For each pattern, the initial and final state is depicted along with two tables listing the site types and species participating in the reaction steps. Note that the first step (CO-O Oxidation) is an irreversible step whereas the second (CO-O_2 oxidation) is reversible. Moreover, both steps elicit a CO_2 molecule in the gas phase.

The rates of elementary reactions are calculated from Arrhenius relationships. For the forward step of a reversible process:

$$k_{fwd} = A_{fwd} \cdot \exp\left(-\frac{E^{\ddagger}_{fwd}(\boldsymbol{\sigma})}{k_B \cdot T}\right) \quad (1)$$

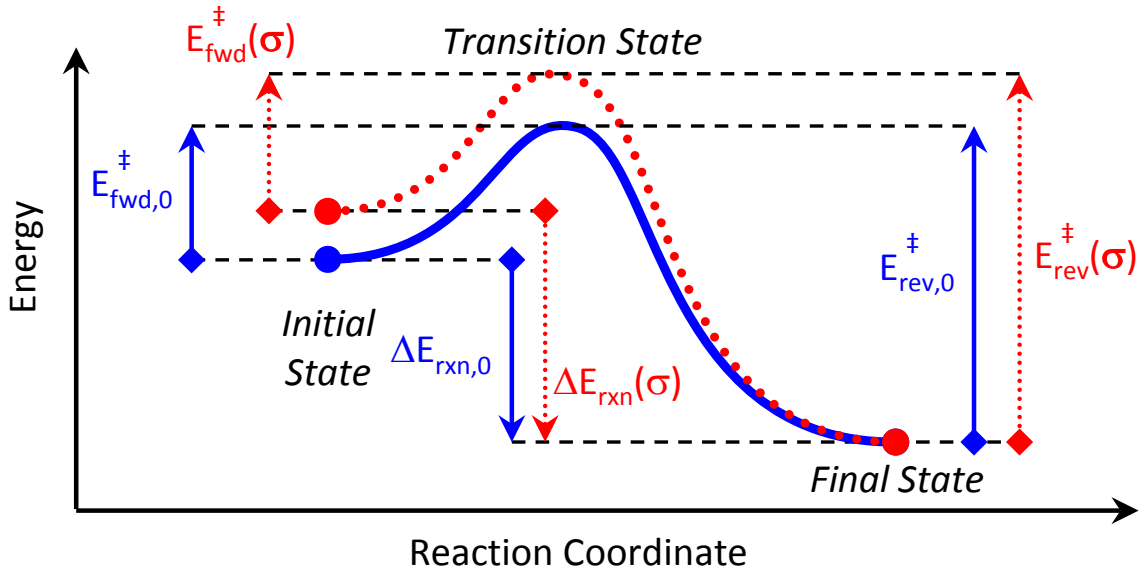


Figure 7: Energy profile of an elementary step. The quantities involved in the calculation of the forward and reverse activation energies are noted.

where A_{fwd} is the pre-exponential (also referred to as pre-factor), $E^{\ddagger}_{fwd}(\boldsymbol{\sigma})$ the activation energy for the given configuration of neighboring adsorbates, k_B Boltzmann's constant, and T the temperature. For the reverse step:

$$k_{rev} = A_{rev} \cdot \exp\left(-\frac{E^{\ddagger}_{rev}(\boldsymbol{\sigma})}{k_B \cdot T} \right) \tag{2}$$

Microscopic reversibility dictates that the difference between forward and reverse activation energy is equal to the reaction energy $\Delta E_{rxn}(\boldsymbol{\sigma})$:

$$\Delta E_{rxn}(\boldsymbol{\sigma}) = E^{\ddagger}_{fwd}(\boldsymbol{\sigma}) - E^{\ddagger}_{rev}(\boldsymbol{\sigma}) \tag{3}$$

In the expression above, $\Delta E_{rxn}(\boldsymbol{\sigma})$ can be calculated from the energetics model (cluster expansion Hamiltonian; see section Energetics Input File). Moreover, the forward activation energy can be parameterized in terms of a Brønsted-Evans-Polanyi (BEP) relationship:[2,5]

$$E^{\ddagger}_{fwd}(\boldsymbol{\sigma}) = \max\left(0,\ \Delta E_{rxn}(\boldsymbol{\sigma}),\ E^{\ddagger}_{fwd,0} + \omega \cdot \left(\Delta E_{rxn}(\boldsymbol{\sigma}) - \Delta E_{rxn,0}\right)\right) \tag{4}$$

where the max operator filters negative values, as well as values less than $\Delta E_{rxn}(\boldsymbol{\sigma})$, if the latter is positive. Moreover, $E^{\ddagger}_{fwd,0}$ and $\Delta E_{rxn,0}$ are the activation and reaction energies at the zero coverage limit (only the reactants existing on the surface), and ω is the so-called proximity factor ranging from 0.0 for an initial-state-like transition state, to 1.0 for a final-state-like transition state. The reverse activation energy expression that is in line with equations (3) and (4) is:

$$E^{\ddagger}_{rev}(\boldsymbol{\sigma}) = \max\left(-\Delta E_{rxn}(\boldsymbol{\sigma}),\ 0,\ E^{\ddagger}_{rev,0} - (1-\omega)\cdot\left(\Delta E_{rxn}(\boldsymbol{\sigma}) - \Delta E_{rxn,0}\right)\right) \quad (5)$$

where:

$$E^{\ddagger}_{rev,0} = E^{\ddagger}_{fwd,0} - \Delta E_{rxn,0} \quad (6)$$

To represent elementary steps constituting a mechanism, Zacros provides a number of keywords discussed below. Unless otherwise stated, these keywords are valid for defining both irreversible and reversible steps and thus can be used inside `step` … `end_step` or `reversible_step` … `end_reversible_step` blocks.

`gas_reacs_prods` *str1 int1* … Provides information about the gas species participating in the mechanism. The name of the first gas species is given in *str1* whereas the stoichiometry is given by integer *int1* (negative for reactants and positive for products). Permitted gas species names are those defined previously with keyword `gas_specs_names` (see section Simulation Input File). In principle, an arbitrary number of such *int str* pairs can appear, although in physically meaningful situations one would generally be limited to at most one reactant and one product.

`sites` *int1* Specifies the number of sites in the graph pattern representing the elementary step being defined.

`neighboring` *int1-int2* … Specifies the neighboring between sites, if more than one sites appear in the graph pattern representing the elementary step. It is followed by expressions structured as *int1-int2* in the same line of input as the keyword. Each such expression denotes that the sites with indexes *int1* and *int2* are nearest neighbors. The values of *int1*, *int2*, … range from 1 up to the number of sites specified by the keyword `sites`. There can be as many such expressions as needed to fully define the neighboring structure of the pattern. For patterns involving only one site, this keyword is omitted.

`initial`

int1 str int2
⋮ ⋮ ⋮

Specifies the initial state of each site in the graph pattern. It is followed by as many lines as the number of sites specified by the keyword `sites`. Each one of these (non-blank) lines contains an expression specifying the state of a site: the first line corresponds to the site indexed 1 in the pattern, the second line to site 2 etc. Note that there is no closing keyword for `initial`; the program exits this input mode once the appropriate number of such lines has been parsed. In each of

these lines, the first argument *int1* is the number of the molecular entity bound to that site. Thus, if a bidentate species is bound to sites 1 and 3, both of these sites will have the same integers in the first column. The second argument *str* gives the name of the surface species bound to the site. Permitted surface species names are those defined previously with keyword surf_specs_names (see section Simulation Input File). Finally, the third and last argument gives the dentate number with which the species is bound. For sites occupied by monodentate species, this number will always be 1.

final

int1 str int2
⋮ ⋮ ⋮

Specifies the final state of each site in the graph pattern. This keyword is subject to the exact same rules as the previously introduced keyword initial.

site_types *str1 str2 …*

The types of each and every site in the pattern. There should be as many strings following this keyword as the number of sites in the pattern specified by sites. This keyword is optional. If omitted, the pattern will be detected based on criteria pertinent to site occupancy and neighboring only.

pre_expon *expr*

Specifies the pre-exponential in the Arrhenius formula giving the rate constant of that elementary step. Possible options for expression *expr* are:

real if a single real number is given, the value of the pre-exponential is assumed to be constant (i.e. independent of temperature).

real1 real2 … real7 seven real numbers following keyword pre_expon, are interpreted as defining a temperature-dependent pre-exponential. The value of the latter is calculated from the following expression:

$$A_{fwd}(T) = \exp\left[-\left(\alpha_1 \cdot \log(T) + \frac{\alpha_2}{T} + \alpha_3 + \alpha_4 \cdot T + \alpha_5 \cdot T^2 + \alpha_6 \cdot T^3 + \alpha_7 \cdot T^4\right)\right] \quad (7)$$

where the values of α_1, α_2, …, α_7 are given by the reals *real1*, *real2*, …, *real7*, respectively, and log is the natural logarithm. This option is useful in simulating temperature programmed desorption or reaction spectra. Note that the equation (7) is

applied over the range between the initial and final temperatures in the simulation, namely $[T_{initial}, T_{initial} + Ramp \cdot t_{simulation}]$ (refer to keywords `temperature` and `max_time` in section Simulation Input File). For temperatures outside this range, the pre-exponential value at the endpoint of the interval is used, for instance if Ramp > 0, for $T < T_{initial}$ the pre-exponential will be taken equal to $A_{fwd}(T_{initial})$. Similarly, for $T > T_{initial} + Ramp \cdot t_{simulation}$ the pre-exponential will be taken equal to $A_{fwd}(T_{initial} + Ramp \cdot t_{simulation})$.

`pe_ratio` *real*

This keyword gives the ratio of forward over reverse pre-exponentials and is valid only inside a reversible elementary step specification block.

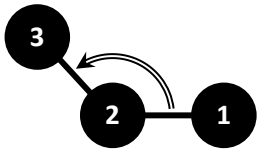
`activ_eng` *real*

The activation energy at the zero coverage limit. For a reversible step, *real* gives the forward activation energy at the zero coverage limit $E^{\ddagger}_{fwd,0}$. The forward activation energy for the given configuration, which enters the Arrhenius equation (1) is computed through the BEP relationship (4). The latter makes use of the reaction energy given by the energetics' model (cluster expansion Hamiltonian; see section Energetics Input File). The reverse activation energy entering the Arrhenius equation (2) is computed through equation (5), such that detailed balance is automatically satisfied.

`prox_factor` *real*

The proximity factor used in the BEP relationship to calculate the forward (and also reverse, if applicable) activation energy (see equations 4, 5). If this keyword is omitted, a default value of 0.5 is used for that elementary step.

`angles` *int1-int2-int3:real1 …*



Specifies a geometric criterion based on the angle between two links connecting pairs of three sites. There can be as many expressions following the keyword `angles` as needed, provided they appear on the same line. The integers *int1*, *int2* and *int3* denote three sites s_1, s_2 and s_3, out of which s_1 neighbors with s_2, and s_2 neighbors with s_3. Then, the value of *real* specifies the angle in degrees between vectors $s_2 \rightarrow s_1$ and $s_2 \rightarrow s_3$. Note that by default, mirror images of patterns are

detected when Zacros scans for the possible elementary processes for a given lattice configuration. Thus, one does not need to explicitly define such mirror images unless the default behavior is overridden as discussed below.

`no_mirror_images` — Overrides the default behavior of the program thereby preventing mirror image pattern detection. By default Zacros detects mirror images by looking for patterns which have angle values opposite from the ones specified in `angles` (for the same site indexes). For instance, if a pattern is specified with `angles 4-5-1:60`, Zacros will also search for patterns having `angles 4-5-1:-60` and all other properties the same as the original pattern. The presence of the keyword `no_mirror_images` restricts the search to the original pattern only.

`absl_orientation` *int1-int2*`:`*real* — Specifies a geometric criterion based on the angle between the x-axis and a link between a pair of sites. The integers *int1* and *int2* denote the neighboring sites. The value of *real* specifies the angle in degrees between vector $s_2 \rightarrow s_1$ and the unit vector (1,0) in Cartesian coordinates. This keyword can be combined with keywords `angles` and `no_mirror_images` for a precise definition of the figures/patterns representing the elementary steps of the reaction mechanism.

`variant` *str*
⋮
expr
⋮
`end_variant` — To reduce repetitions in the `mechanism_input.dat` file, the `variant` blocks can be of particular use. Thus, after the keywords `gas_reacs_prods`, `sites`, `neighboring`, `initial` and `final`, inside an elementary step definition block, one can define one or more variants that will all share the same gas reactants/products, lattice structure and initial/final occupancies, but may vary in their geometry or site types. The name of the variant pattern consists of string *str* appended to the name of the parent pattern (*str* following the keyword `step` or `reversible_step`). In this respect, the following keywords are permitted within a `variant` block: `site_types`, `pre_expon`, `pe_ratio` (for reversible step only), `active_eng`, `angles`, `no_mirror_images`, `absl_orientation`. If any of these keywords has been listed within a `step`/`reversible_step` block before a

variant block has been opened, the keyword variant is no longer permitted within that block.

Examples

As guiding examples, we finally give the Zacros input defining the elementary steps of Figure 6.

```
step CO-O_Oxidation            # Opening an irreversible step block
  gas_reacs_prods CO2 1        # One CO2 product molecule
  sites 2                      # There are two sites in the pattern…
  neighboring 1-2              #  that are neighbors
  initial                      # Initial state:
    1 CO*   1                  #  1st site occupied by CO* (monodentate)
    2 O*    1                  #  2nd site occupied by O* (monodentate)
  final                        # Initial state:
    1 *     1                  #  unoccupied 1st site (* is monodentate)
    2 *     1                  #  unoccupied 2nd site
  site_types top fcc           # Specifying site types in the pattern…
  pre_expon    1.000e+013      #  along with the pre-exponential…
  activ_eng    0.200           #  activation energy…
  prox_factor  0.500           #  and proximity factor
end_step                       # Closing the step block
```

```
reversible_step CO-O_Oxidation   # Opening a reversible step block
  gas_reacs_prods CO2 1        # One CO2 product molecule
  sites 5                      # There are five sites in the pattern…
  neighboring 1-2 2-3 3-4 4-5  #  that neighbor as specified
  initial                      # Initial state:
    1 O2*** 3                  #  1st site occupied by 3rd dentate of O2
    1 O2*** 2                  #  2nd site occupied by 2nd dentate of O2
    1 O2*** 1                  #  3rd site occupied by 1st dentate of O2
    2 *     1                  #  4th site unoccupied
    3 CO*   1                  #  5th site occupied by CO* (monodentate)
  final                        # Initial state:
    1 *     1                  #  1st site unoccupied
    2 O*    1                  #  2nd site occupied by O* (monodentate)
    3 *     1                  #  3rd site unoccupied
    4 *     1                  #  4th site unoccupied
    5 *     1                  #  5th site unoccupied
  site_types top brg top brg top # Specifying site types…
  pre_expon    1.000e+013      #  along with the pre-exponential…
  pe_ratio     1.800e+006      #  fwd/rev pre-exponential ratio…
  activ_eng    0.300           #  activation energy…
  prox_factor  0.500           #  and proximity factor
end_reversible_step            # Closing the reversible step block
```

Initial State Input File

By default a KMC simulation in Zacros is initialized with an empty lattice. However, there are cases in which one would need to explicitly specify an initial state, for instance, in simulations of temperature programmed desorption (TPD) or reaction (TPR), or in order to initialize the simulation from a representative equilibrium configuration. The user may thus override the default behavior by supplying a file named `state_input.dat`. This is an optional input file and, as noted before, Zacros will start from an empty lattice in the absence thereof. The syntax used in this file is discussed below.

```
initial_state
   ⋮
 expr
   ⋮
end_initial_state
```

Defines an initial state specification block. Anything after the keyword `end_initial_state` is ignored. The initial state specification block contains one or more "particle seeding" instructions *expr* (explained below), which allow Zacros to populate the lattice with the desired number of particles.

`seed_on_sites` *str int1 int2 …*

One or more such "individual seeding" instructions can appear in place of *expr* in an initial state specification block (see above). Each such instruction seeds one particle of the species with name *str* on sites specified by the integers *int1*, *int2*,… Permitted surface species names are those defined previously with keyword `surf_specs_names` and the number of integers following *str* must not exceed the number of dentates of that species, defined by `surf_specs_dent` (see section Simulation Input File). Finally, the *int1*, *int2*,… can range between 1 up to the number of sites that exist on the lattice.

```
seed_multiple str1 int1
 site_types str2 str3 …
 neighboring int2-int3 …
end_initial_state
```

One or more such "multiple seeding" blocks can appear in place of *expr* in an initial state specification block (see above). Each such instruction seeds multiple particles of the species with name *str1*. The number of particles is defined by the integer *int1*. The site types in which these particles will be seeded is given by *str2*, *str3*, … There should be as many such strings as the number of dentates of that species, defined by `surf_specs_dent` (see section Simulation Input File). If the species is monodentate, the `neighboring` keyword is omitted; otherwise a neighboring structure is specified using this keyword. This is done by using as many expressions of the form *int2-int3* as needed, in order to define the links between the sites occupied by that species. Note that if the neighboring structure thus defined cannot be found on the lattice, execution of the seeding instruction will fail.

Examples

As illustrative examples, consider the following cases:

Suppose we need to seed 2 carbonate (CO_3) particles randomly on the Au_6 lattice (Figure 4). CO_3 binds in a top-bridge-top configuration at sites cn2-brg42-cn4. Thus, we can use the following instructions in the file `state_input.txt`:

```
initial_state                    # Opening initial state block
   seed_multiple CO3*** 2        # Two CO3*** molecules to be seeded…
      site_types cn2 brg42 cn4   #  on the specified site types
      neighboring 1-2 2-3        # Defining dentates' neighboring
   end_seed_multiple             # Closing seed_multiple block
end_initial_state                # Closing initial_state block
```

Alternatively suppose we would like to seed two CO_3 molecules at specific sites on the Au_6 structure. We could then use the following syntax:

```
initial_state                    # Opening initial state block
   seed_on_sites CO3*** 1 6 10   # First CO3*** seeding instruction
   seed_on_sites CO3*** 15 14 12 # Second CO3*** seeding instruction
end_initial_state                # Closing initial_state block
```

Interpreting the Simulation Output Zacros

General Output File

The file `general_output.txt` contains general information about the KMC simulation. The file contents are broken down to the following sections, which are mostly self-explanatory:

Simulation Setup

This section repeats the information parsed while processing file `simulation_input.txt`. If everything is valid, this section ends with the message "`Finished reading simulation input`." otherwise an error is output to this file and execution is terminated.

Lattice Setup

In this section, information about the lattice structure is presented, namely the type of lattice specification (default, periodic, explicit; see section Lattice Input File), the area of the simulation box for periodic lattices, the site types and the number of sites per type, as well as the maximum coordination number in the lattice. If everything is valid, this section ends with the message "`Finished reading lattice input`." otherwise an error is output to this file and execution is terminated.

Energetics Setup

This section reports the number of clusters for the cluster expansion Hamiltonian parsed from the `energetics_input.dat` file, and the maximum number of sites involved in a cluster. A summary of the clusters defined is also given. If everything is valid, this section ends with the message "`Finished reading energetics input.`" otherwise an error is output to this file and execution is terminated.

Mechanism Setup

This section reports the number of elementary steps parsed from the `mechanism_input.dat` file, and the maximum number of sites involved in a step. A summary of the elementary steps contained in the mechanism is also given. If everything is valid, this section ends with the message "`Finished reading mechanism input.`" otherwise an error is output to this file and execution is terminated.

Initial State Setup

This section appears only if an initial state has been defined using file `state_input.dat` and summarizes all seeding instructions parsed therefrom. If everything is valid, this section ends with the message "`Finished reading initial state input.`" otherwise an error is output to this file and execution is terminated.

Threading Information

This section gives information about parallelization. If the program has been compiled as a serial application, the message "`NO THREADS`" will appear. If the compiler recognized the OpenMP directives, the message will read "`WITH THREADS` *`int`*" where *`int`* is the number of threads used during execution (please refer to section Running Zacros for more information about setting the number of threads).

Simulation Output

This section opens with the message "`Commencing simulation`" and closes with "`Simulation stopped`". If event reporting is turned on (see keyword `event_report` in section Simulation Input File) the occurrence of each lattice process is reported using the following format:

```
KMC step int1
   Elementary step str
   occurred at time t = real
   involving site(s):   int2 int3 …
```

where *`int1`* is the KMC step number, *`str`* is the name of the elementary step that just occurred, *`real`* is the time of occurrence thereof, and *`int2`*, *`int3`*,… are the indexes of the lattice sites on which the event took place. In the end of the simulation, right after the message "`Simulation stopped`", the KMC time, total number of elementary events simulated, and the event frequency are reported:

```
Current KMC time: real1
Events occurred:  int
Event frequency:  real2
```

Performance Facts

Metrics about the performance of the program are also reported right after message "`Performance facts`":

```
Elapsed CPU time:          real1 seconds
Elapsed clock time:        real2 seconds

Clock time per KMC event: real3 seconds
Clock time per KMC time:  real4 seconds/KMCTimeUnits

Events per clock hour:     int
KMC Dt per clock hour:     real5 KMCTimeUnits
```

In the above, *real1* gives the CPU time spent whereas *real2* is the real time, which we could for instance measure using a stopwatch. The `wall_time` constraint is imposed on real time (see section Simulation Input File). If the code was compiled as a serial application, *real1* and *real2* should be approximately the same.

The value of *real3* gives the real time needed on average to execute a single KMC step. This time is reported in seconds and is used to compute of how many events can be executed if the simulation was to be left running for one hour, as also reported in the value of *int*.

Moreover, *real4* gives the real time needed on average to propagate the system for 1 unit of KMC time. *real5* shows how far in KMC time the system will go in one hour of real time.

Note that the performance metrics are not aggregated if the simulation is run in multiple chunks by use of the restart feature. Thus, every time the simulation is restarted, Zacros resets the counters used to evaluate these performance metrics.

Newton's Method Statistics

Finally, if a temperature ramp has been specified, Zacros solves a non-linear equation to find the time of occurrence of each elementary event.[1] Statistics about the performance of the Newton-Raphson method are accrued during the simulation and will be aggregated if the simulation is restarted. The results are reported in this section and look like the following:

```
Total number of times run: int1
Number of times failed:    int2
Avg number of iterations:  real1
Maximum Dx error:          real2
Maximum RHS error:         real3
```

Note that the total number of times run (*int1*) is not equal to the number of KMC events simulated. This happens because in the course of the KMC simulation there are always processes that are detected but subsequently removed if any of the participating adsorbates "decides to do something else". The number of times failed *int2* should be zero. A non-zero value indicates that in one or several occasions (as many as *int2*) the Newton- Raphson loop went through the maximum number of iterations (150 by default) without converging, which may be cause for concern. The maximum errors are also reported:

real2 is the maximum norm of the difference between subsequent approximations of the solution, whereas *real3* is the maximum norm of the right hand side. Both tolerances are 10^{-9} by default. Refer to section Simulation Input File on how to override these default tolerances and the maximum number of iterations.

If the simulation has terminated successfully, the message "`> Normal termination <`" is written in the end of the file `general_output.txt`. In the case the simulation is being restarted, a short message appears providing information about how many times has the simulation been restarted previously, the last reported time and number of KMC events. The message "`> Normal termination <`" is also written in the end of every restart session.

Lattice Output File

After parsing the lattice setup information, Zacros writes the file `lattice_output.txt`, which summarizes the lattice structure. The first two lines of this file follow the format:

```
0 real1 real2 0 0 …
0 real3 real4 0 0 …
```

namely integer-type zeroes everywhere, except the 2nd and 3rd element of each row. These non-zero elements give the two vectors defining the entire simulation box in row format, namely α = (*real1*, *real2*) and β = (*real3*, *real4*). If the lattice has been defined using the keyword `explicit` these real numbers have values of zero.

The third and following lines give all the information pertinent to each site of the lattice, following the format:

int1 real1 real2 int2 int3 int4 int5 …

where:

int1 (1st column) is the index of the site (ranging from 1 to the total number of sites),

real1 and *real2* (2nd and 3rd columns) are the x and y Cartesian coordinates of site *int1*,

int2 (4th column) gives the site type of the site with index *int1*,

int3 (5th column) gives the coordination number of the site with index *int1*,

int4 int5 … (6th and following columns) give the 1st nearest neighbors of the site with index *int1*. Zacros always reports as many integers here as the maximum coordination number, writing zeroes after the last nearest neighbor.

History Output File

During the course of a simulation, Zacros takes snapshots of the lattice state and writes them in file `history_output.txt`, along with other pertinent information. The frequency at which snapshots are being taken is defined by keyword `snapshots` in file `simulation_input.txt` (see section Simulation Input File). The contents of the file `history_output.txt` are structured as follows.

The first few lines of the file constitute a header, with general information about the simulation:

```
Gas_Species:           str1 str2 …
Surface_Species:       str3 str4 …
Simulation_Box:
           real1            real2
           real3            real4
Site_Types:            str5 str6 …
```

These are mostly self-explanatory. Note, however that for explicitly defined lattices (see keyword `explicit` in section Explicitly Defined Custom Lattices), the Simulation box information does not appear.

The rest of the file consists of sections beginning with the word "configuration" followed by information structured as discussed below.

```
configuration  int1 int2 real1 real2 real3
         int3   int4   int5   int6
           ⋮      ⋮      ⋮      ⋮
           int7   int8    …
```

In the above, *int1* is a counter showing how many configurations have been written so far in file `history_output.txt`. Integer *int2* gives the number of KMC events that have happened up to that point. The next three reals *real1*, *real2* , *real3*, give the time, temperature and the energy of the current lattice configuration. The subsequent lines contain integers that encode the state of the lattice; there are as many such lines as the number of lattice sites. The information is presented as follows:

int3 (1st column) is the site number on the lattice,

int4 (2nd column) gives the entity/adsorbate number (each adsorbate on the lattice has a unique number/identifier, this is it),

int5 (3rd column) denotes the species number (zeroes are reported for empty sites),

int6 (4th column) gives the dentate number with which entity *int4* occupies site *int1*.

Finally, the last line (*int7*, *int8*, …) gives the number of molecules produced (or consumed if the corresponding value is negative) for each gas species in the chemistry. The order in which these numbers are reported is the same as the order with which gas species were defined (see `gas_specs_names` and pertinent keywords in section Simulation Input File) and also are mentioned

in the header of `history_output.txt`. Thus, *int7* refers to species *str1*, *int8* to species *str2* etc.

Process Statistics Output File

During the course of a simulation, Zacros collects statistical information about the occurrence of elementary events which is reported in file `procstat_output.txt`. This statistical information is always updated after every KMC event, whereas the frequency at which it is reported is defined by keyword `process_statistics` in file `simulation_input.txt` (see section Simulation Input File). The contents of the file `procstat_output.txt` are structured as follows.

The first line of the file constitutes a header following the format:

`Overall` *str1 str2 …*

The word "Overall" appears always first and is followed by strings that correspond to the names of all elementary events defined in file `mechanism_input.dat`.

The rest of the file consists of sections beginning with the word "configuration" followed by information structured as discussed below.

```
configuration  int1 int2 real1
        real2   real3   real4     …
         int3    int4    int5     …
```

In the above, *int1* is a counter showing how many configurations have been written so far in file `procstat_output.txt`. Integer *int2* gives the number of KMC events that have happened up to that point and *real1* the current time. The next two lines provide statistical information about each elementary step of the mechanism in the same order as mentioned in the header.

Thus, *real2*, *real3*, *real4*, … give the average waiting times $\bar{\tau}_k$ (also referred to as inter-arrival times) for each reaction event:

$$\bar{\tau}_k = \frac{1}{N_k^{occur}} \sum_{i \geq 1}^{N_k^{occur}} \tau_{k,i} \qquad (8)$$

where the averaging is done every time elementary event k occurs. Thus, N_k^{occur} is the number of times event k was executed so far in the KMC simulation, and $\tau_{k,i}$ the waiting time for that event to occur (the waiting time for event k is by definition the time that passed since the occurrence of the most recent event of any type). The value of *real2* refers to an "overall" average in which all events are considered.

Moreover, *int3*, *int4*, *int5*, … give the numbers of times each event was executed during the KMC simulation. The value of *int3* refers the total number of events and should be the same as the value of *int2* in the header of file `procstat_output.txt`. Moreover, the values of *int4*, *int5*, … should sum up to that of *int3*.

Species Numbers Output File

Zacros also reports the number of surface and gas species along with other pertinent information in file `specnum_output.txt`. The frequency at which this information is reported is defined by keyword `species_numbers` in file `simulation_input.txt` (see section Simulation Input File). The contents of this file are pretty self-explanatory and are summarized in the first line (header) of the file. the overall structure is as follows:

```
 Entry   Nevents    Time   Temperature  Energy  str1  str2 …   str3  str4 …
  int1     int2    real1     real2      real3   int3  int4 …   int5  int6 …
   ⋮        ⋮        ⋮         ⋮          ⋮      ⋮     ⋮        ⋮     ⋮
```

Entry refers to an integer counting how many lines have been written to this output file. The column marked as "`Nevents`" shows the total number of KMC events that happened up to that point, followed by a column that shows the (simulated) time passed. The next column gives the temperature which should be constant unless a temperature ramp has been specified (see keyword `temperature` in section Simulation Input File). The column marked as "`Energy`" gives the energy of the current lattice configuration. The following columns marked as *`str1`*, *`str2`*, … report the number of molecules of each species currently adsorbed on the lattice (the strings are the names of the surface species). Note that total numbers are reported; thus, if a species can bind to two different sites, this output does not provide any information as to how many particles are bound to sites of type 1 versus 2. Finally, the columns marked as *`str3`*, *`str4`*, … report the number of molecules of each gas species. For species that appear as products in the net reaction under consideration, one should expect to see positive numbers in this column. Negative numbers would be reported for reactant species.

Energetics Debug Output File

The output file `globalenerg_debug.txt` is generated if `simulation_input.dat` contains the keyword `debug_report_global_energetics` (see section Simulation Input File), and provides a full account of the bookkeeping related to energetics in the course of a KMC simulation. In particular, the file contains sections starting with the words "`Initialization`", and "`KMC step` *`int`*" and ending with the expression "`Current total lattice energy is` *`real`*". In these sections one or more of the following expressions can be contained:

```
Total empty-cluster energy constant = real
```

This constant will be zero, unless an "empty cluster" has been specified. The latter, is a cluster involving a single site with an unspecified state (using keyword `&`; see section Energetics Input File).

```
Global-cluster int1 identified:
    Cluster number: int2
    Cluster description: str
    Mapping of lattice to pattern sites: int3 int4 …
    Cluster graph-multiplicity: int5
    Its energy contribution is real
```

The expressions above are written when a pattern representing an energetic contribution has been detected in the current lattice configuration. During the course of the simulation, Zacros keeps a list of all such patterns, so that it can quickly compute changes in the lattice energy when adsorption/desorption diffusion and reaction events take place. Thus, *int1* is the index in this list of patterns, *str* is the name of the pattern just detected (one of the cluster names defined in energetics_input.dat; see section Energetics Input File); *int3 int4 …* give the location of this pattern on the lattice; *int5* and *real* just repeat the graph multiplicity and energy contribution values that were defined in energetics_input.dat using the keywords graph_multiplicity and cluster_eng, respectively.

Cluster *int* was removed.

The message above indicates that an energetic contribution was removed from the list, because the corresponding pattern ceased to exist.

Cluster *int1* was relabeled to *int2*.

Regarding this message, note that Zacros stores the list of patterns in a data-structure in which each pattern is indexed by an integer ranging from 1 to the total number of patterns N_{Tot}. To avoid generating gaps in this data-structure upon removal of a pattern N_{remv}, the last pattern with index N_{Tot} takes the index N_{remv}, so that the new set of indexes ranges from 1 to $N_{Tot} - 1$. The message above indicates that such a re-indexing took place, with N_{Tot} = *int1* and N_{remv} = *int2*.

Process Debug Output File

The output file process_debug.txt is generated if simulation_input.dat contains the keyword debug_report_processes (see section Simulation Input File), and provides a full account of the bookkeeping related to elementary event occurrence in the course of a KMC simulation. In particular, the file contains sections starting with the words "Initialization", and "KMC step *int*". In these sections the following expressions can be contained:

```
Process int1 identified:
    Elementary step number: int2
    Elementary step description: str
    Mapping of lattice to pattern sites:  int3 int4 …
    Its activation energy at the zero-coverage limit is real1
    Its activation energy for the given configuration is real2
    Its energy of reaction at the zero-coverage limit is real3
    Its energy of reaction for the given configuration is real4
    Its propensity at T0 is real5
    It will occur at t = real6 after Dt = real7
```

The expressions above are output when a pattern representing an elementary process has been detected in the current lattice configuration. During the course of the simulation, Zacros keeps a list of all such patterns in a heap data-structure, in order to be able to find in constant time the next event to take place. Thus, *int1* is a unique identifier in this heap, *str* is the name of the pattern just detected (one of the elementary event names defined in mechanism_input.dat; see section Mechanism

Input File); *int3 int4 …* gives the location of this pattern on the lattice; *real1* is the activation coverage at the zero coverage limit ($E^{\ddagger}_{fwd,0}$ or $E^{\ddagger}_{rev,0}$ in equations 4, 5); *real2* is the actual activation energy for the current configuration ($E^{\ddagger}_{fwd}(\boldsymbol{\sigma})$ or $E^{\ddagger}_{rev}(\boldsymbol{\sigma})$ in equations 4, 5); *real3* is the activation coverage at the zero coverage limit ($\Delta E_{rxn,0}$ in equations 4, 5); *real4* is the actual activation energy for the current configuration ($\Delta E_{rxn}(\boldsymbol{\sigma})$ in equations 4, 5). The value of *real5* gives the propensity (equations 1, 2) at the initial temperature of the simulation (which would be the same throughout the simulation if no temperature ramp has been defined). Finally, the random time for the occurrence of that event is reported in the last line: *real6* is the absolute time of occurrence and *real7* is the time increment (relative to the current time in which the process was detected).

`Process int was removed.`

The message above indicates that an elementary process was removed from the list, because the corresponding pattern ceased to exist.

`Process int1 was relabeled to int2.`

This message indicates that a process has been re-indexed to avoid generating gaps in the heap data-structure upon removal of a pattern. Thus, if pattern N_{remv} is being removed, the last pattern with index N_{Tot} takes the index N_{remv}, so that the new set of indexes ranges from 1 to $N_{Tot} - 1$. The message above indicates that such a re-indexing took place, with N_{Tot} = *int1* and N_{remv} = *int2*.

Notes on Troubleshooting

Zacros is able to identify syntax errors in the input files. If such an error is detected, the program will report an error with a detailed description of what the problem was and in which line of which file it was encountered.

In some cases though, the syntax may be perfectly valid but the specification might not be the one intended. The following notes provide some hopefully useful considerations and guidelines for troubleshooting.

1. **Numbering/Naming consistency**: make sure your numbering and naming is consistent throughout your input. For instance, the order in which the names of surface species appear after keyword `surf_specs_names` must match their dentate numbers after keyword `surf_specs_dent`. Similarly for the gas species definition.

2. **Pattern consistency**: make sure that the binding configurations of different species are used in consistent way in the following: the seeding instructions of `state_input.dat` (see section Initial State Input File), the energetic clusters of `energetics_input.dat` (see section Energetics Input File) and the elementary events of `mechanism_input.dat` (see section Mechanism Input File). For instance, if the binding configuration of a bidentate species has been defined with dentate 1 occupying a site of type “top” and with dentate 2 occupying a site of type “fcc”, this convention

should be followed throughout. If an individual seeding instruction (keyword `seed_on_sites`, section Initial State Input File) places this species on the wrong sites, Zacros will execute the instruction, but since no cluster contribution pattern will be detected, the lattice energy will remain the same after addition of this species.

3. **Quick checks**: it is worth checking the summary of energetics and mechanism specifications in file `general_output.txt` (see section General Output File). If the patterns that appear there are not the ones intended, there may be a problem with the input. In some cases the program will issue warnings which may not have catastrophic consequences by it is possible that they need to be addressed.

4. **Advanced checks**: one can frequently discover problems in the simulation setup by making use of the debugging keywords:

   ```
   debug_report_processes
   debug_report_global_energetics
   debug_newtons_method
   debug_check_processes
   debug_check_lattice
   ```

   in simulation_input.dat (see section Simulation Input File) along with the output information of the debugging files:

   ```
   globalenerg_debug.txt
   process_debug.txt
   ```

   For instance, to make sure that the energetics model was defined properly, one could work out an example problem, specify a configuration in file state_input.dat and see if the clusters are being detected properly.

Known Limitations

1. For input files, the maximum record length that can be parsed is 2^{13} = 8192 characters. A maximum of 3000 words can be parsed. The maximum allowed length for the names of species, site-types, clusters and mechanism-steps is 64 characters. These limits can be changed by redefining the appropriate constants in file `constants_module.f90` and recompiling Zacros.

2. The energy units in files `energetics_input.dat` and `mechanism_input.dat` are assumed to be in eV. If you need to use a different unit you can redefine parameter `enrgconv` in `constants_module.f90` and recompile the program (the values are provided so you only need to uncomment the appropriate line). See also section Input/Output Files.

3. Sites with unspecified states are not supported for elementary events. In most cases, sites that participate in the elementary event are occupied by reactants, products or transiently by the transition state. Thus, “extra” sites must usually be defined as empty rather than unspecified.

4. The calculation of energetics for lattices smaller than the maximum interaction length fails to provide accurate values. It is recommended that the size of the lattice be chosen as at least twice the length of the longest-range interaction pattern.

5. There is no explicit limitation in the number of surface and gas-phase species, the size of the lattice, and the number of cluster and elementary event patterns that can be defined, as the pertinent data-structures consist of allocatable objects. However, different compilers and operating systems may impose their limitations. Please refer to the documentation thereof.

References

[1] Stamatakis, M. and D.G. Vlachos, *A Graph-Theoretical Kinetic Monte Carlo Framework for on-Lattice Chemical Kinetics.* Journal of Chemical Physics, 2011. **134**(21): p. 214115.

[2] Nielsen, J., M. d'Avezac, J. Hetherington, and M. Stamatakis, *Parallel Kinetic Monte Carlo Simulation Framework Incorporating Accurate Models of Adsorbate Lateral Interactions.* Journal of Chemical Physics, 2013. **139**(22): p. 224706.

[3] Stamatakis, M., M. Christiansen, D.G. Vlachos, and G. Mpourmpakis, *Multiscale Modeling Reveals Poisoning Mechanisms of MgO-Supported Au Clusters in CO Oxidation.* Nano Letters, 2012. **12**(7): p. 3621-3626.

[4] Sanchez, J.M., F. Ducastelle, and D. Gratias, *Generalized Cluster Description of Multicomponent Systems.* Physica A: Statistical and Theoretical Physics, 1984. **128**(1-2): p. 334-350.

[5] Wu, C., D.J. Schmidt, C. Wolverton, and W.F. Schneider, *Accurate coverage-dependence incorporated into first-principles kinetic models: Catalytic NO oxidation on Pt(111).* Journal of Catalysis, 2012. **286**: p. 88-94.