MCell3 Quick Tutorial and Reference Guide

October 19, 2006

In this document, the main text is in a serif font. Command-line entries, MDL file commands, and code is in a `fixed-width` font. Values that must be supplied by the user are in an *italicized sans-serif* font.

This document was created using LyX 1.3.4.

1 Running MCell3

MCell3 runs on the command line. The format is

`mcell3` *options filename*

By default, MCell3 sends informational messages, such as simulation progress, to stdout (which will normally appear on the screen); error messages are sent to stderr (which will also normally appear on the screen). Results of simulations are written to files and do not appear as MCell3 is running.

A brief summary of MCell3 optional command-line arguments is given below.

Argument	**Explanation**
`-seed` *N*	Start with random number seed *N* instead of 1 (the default).
`-iterations` *N*	Run the simulation with *N* timesteps (overrides the value in the mdl file)
`-help`	Print out a basic help screen.
`-info`	Same as `-help`
`-logfile` *filename*	Send messages to *filename* instead of stdout/stderr
`-errfile` *filename*	Send error messages to *filename* instead of stderr
`-logfreq` *N*	Print out a message when every *N* iterations have finished.
`-checkpoint_infile` *filename*	Use *filename* as a checkpoint file for the current simulation (overrides any value in the MDL file).

2 Model Description Language overview

MCell3 runs simulations that are specified in *model description language* (MDL) format. These files typically have the extension .mdl, but are not required to. A MDL file is a text file with commands separated by whitespace. The nature and type of whitespace (space, tab, newline) is unimportant to MCell3. You are thus free to use whitespace to clarify the contents of the MDL file.

2.1 The structure of an MDL file

Commands fall into five general groups, which usually should be given in the order presented below. Although this is not always required, there are some commands (e.g. defining a molecule) that must be used before others (e.g. defining a reaction that uses that molecule). The order below should always be safe:

1. Initialization. These commands set global parameters such as the timestep, spatial partitioning, and duration of the simulation.
2. Molecule definitions. These commands specify the names and diffusion constants of molecules in the simulation.
3. Reaction definitions. These commands specify the reactions that can occur between molecules and the rate at which those reactions occur.
4. Geometry specification. These commands describe the membranes and other boundaries within which the simulation occurs, plus where in the world to place molecules initially.
5. Output specification. These commands specify what data should be output as the simulation is running; this can include graphical snapshots of the simulation in progress, as well as lists of numbers of molecules or reactions as a function of time.

In addition, there are utility commands–defining variables and including other MDL files–that can appear nearly anywhere.

2.2 How to use this document

This document gives a brief description of every valid MCell3 command. Commands can be specified one after another; it is often convenient to put commands on separate lines but this is not necessary.

Some commands have a scope, delimited by `{` and `}` (braces). Within these braces, a different set of commands become available. In this document, each set of commands is given a different title. For example, commands given within a `DEFINE_MOLECULE` block receive the title Define Molecule Commands.

3 MDL commands

3.1 Initialization commands

The following initialization commands are required in every MDL file.

Command	Explanation
`TIME_STEP` = *t*	Set the simulation time step to *t* seconds. `1e-6` is a common value. Later commands can change the time steps taken by individual molecules, but this time step is still used by all output statements.
`ITERATIONS` = *N*	Run the simulation for *N* iterations.

The following initialization commands are optional.

Command	Explanation
`TIME_STEP_MAX` = *t*	MCell3 will move longer than the specified simulation time step if it seems safe. This command makes sure that the longest possible time step is no longer than *t* seconds, even if MCell3 thinks a longer step would be safe. The default is no limit.
`SPACE_STEP` = *N*	Have all diffusing molecules take time steps of different duration, chosen so that the mean diffusion distance is *N* microns for each molecule. By default, all molecules move the same time step.
`CHECKPOINT_INFILE` = *"filename"*	Start the simulation using the conditions specified in the checkpoint file *filename*. This will start at the time that the saved simulation left off, and will use molecules stored in the specified file instead of surface molecule densities/numbers specified in the MDL file. Release sites can add new molecules if the release time is after the time the simulation starts.
`CHECKPOINT_OUTFILE` = *"filename"*	Save the state of the simulation when `CHECKPOINT_ITERATIONS` (described below) is reached, and stop.
`CHECKPOINT_ITERATIONS` = *N*	Used with `CHECKPOINT_OUTFILE`. This specifies how many iterations to run before stopping and writing the checkpoint file. If *N* is less than `ITERATIONS`, the simulation will terminate normally instead.
`SURFACE_GRID_DENSITY` = *N*	Tile all surfaces so that they can hold molecules at *N* different positions per square micron. The default is 10000. For backwards compatability, `EFFECTOR_GRID_DENSITY` works also.
`INTERACTION_RADIUS` = *N*	Diffusing molecules will interact with each other when they get within *N* microns of each other. The default is $1/\sqrt{\pi \cdot \sigma_s}$ where σ_s is the surface density (default or user-specified).
`PARTITION_`*D* = [*list*]	Subdivide the *D*'th axis of space, where *D* is `X`, `Y`, or `Z`, at the boundaries given in *list* (in microns). In future versions, MCell3 will further subdivide space if it is computationally advantageous. By default, each axis will be split into between five and fifteen equal partitions. If you do not explicitly partition all three axes, MCell3 is likely to ignore your request and perform automatic partitioning. The spacing between adjacent partitions must be larger than the `INTERACTION_RADIUS`.
`RADIAL_DIRECTIONS` = *N*	Specifies how many different directions to put in the lookup table. The default is sensible. Don't use this unless you know what you're doing. Instead of a number, you can specify `FULLY_RANDOM` to generate the directions directly from double precision numbers (but this is slower).
`RADIAL_SUBDIVISIONS` = *N*	Specifies how many distances to put in the diffusion lookup table. Again, the default is sensible. `FULLY_RANDOM` is not implemented here.

Command	Explanation
ACCURATE_3D_REACTIONS = *boolean*	Specifies which method to use for computing 3D molecule-molecule interactions. If *boolean* is TRUE, then molecules will look through partition boundaries for potential interacting partners–this is slower but more accurate. If *boolean* is FALSE, then molecule interaction disks will be clipped at partition boundaries and probabilities adjusted to get the correct rate–this is faster but can be less accurate. The default is TRUE.
CENTER_MOLECULES_ON_GRID = *boolean*	If *boolean* is set to TRUE, then all molecules on a surface will be located exactly at the center of their grid element. If FALSE, the molecules will be randomly located when placed, and reactions will take place at the location of the target (or the site of impact in the case of 3D molecule/surface reactions). The default is FALSE.
VACANCY_SEARCH_DISTANCE = *r*	Normally, a reaction will not proceed on a surface unless there is room to place all products on the single grid element where the reaction is initiated. By increasing *r* from its default value of 0, one can specify how far from the reaction's location, in microns, the reaction can place its products. To be useful, *r* must be larger than the longest axis of the grid element on the triangle in question. The reaction will then proceed if there is room to place its products within a radius *r*, and will place those products as close as possible to the place where the reaction occurs (deterministically, so small-scale directional bias is possible).

NOTIFICATIONS
{
notification commands
}
	This block of commands lets you set the informational messages that MCell3 generates. The block can appear multiple times and applies to all MDL below it in the file. It can appear anywhere at the top level (but not inside other blocks).

WARNINGS
{
warning policy commands
}
	This block of commands lets you control how MCell3 handles warnings—whether it generates a warning and continues, silently handles the condition, or generates an error and quits. The block can appear multiple times and applies to all MDL below it in the file. It can appear anywhere at the top level (but not inside other blocks).

The following commands can be given in a notifications block; in each case, setting the notification policy to OFF will prevent any informational output regarding that aspect of the simulation. This will not affect warnings.

Notification Command	Explanation
BOX_TRIANGULATION_REPORT = *policy*	If *policy* is ON, MCell3 will report how many triangles are generated from each box object. Default is OFF.
DIFFUSION_CONSTANT_REPORT = *policy*	If *policy* is ON, MCell3 will report four measures of the diffusion constant for each molecule. If *policy* is BRIEF, MCell3 will report just one measure (average diffusion distance per step) for each molecule. Default is BRIEF.
FILE_OUTPUT_REPORT = *policy*	If *policy* is ON, MCell3 will report every time reaction data is written to disk. Default is OFF.
FINAL_SUMMARY = *policy*	If *policy* is ON, MCell3 will give some information about the CPU time used and some of the internal events. Default is ON.
ITERATION_REPORT = *policy*	If *policy* is ON, MCell3 will provide a running report of how many iterations have completed, chosen based on the total number of iterations. If *policy* is an integer value, MCell3 will report each time that number of iterations have elapsed. Default is ON.
PARTITON_LOCATION_REPORT = *policy*	If *policy* is ON, MCell3 will print out the locations of the partitions used for the simulation. Default is OFF.

Notification Command	Explanation
PROBABILITY_REPORT = *policy*	If *policy* is ON, MCell3 will print out the reaction probabilities for each reaction (except special internal surface reactions such as absorptive surfaces). Default is ON. This will reset the reporting threshold to a probability of zero.
PROBABILITY_REPORT_THRESHOLD = *p*	MCell3 will print out the probabilities for every reaction with probability greater than or equal to *p*. This will override the policy for probability reports.
VARYING_PROBABILITY_REPORT = *policy*	If *policy* is ON, MCell3 will print out the reaction probabilities when a time-varying reaction updates its reaction rate (regardless of the old or new probability). Default is ON.
PROGRESS_REPORT = *policy*	If *policy* is ON, MCell3 will print out messages indicating which part of the simulation process is underway (initializing, running, etc.). Default is ON.
RELEASE_EVENT_REPORT = *policy*	If *policy* is ON, MCell3 will print out a message every time molecules are released through a release site (indicating how many molecules of which type were released and the iteration on which they were released). Default is ON.
ALL_NOTIFICATIONS = *policy*	Set all notification policies to the same value (ON or OFF). This overrides the existing probability report threshold, if there is one.

The following commands can be given in a warnings block. Setting the warning policy to IGNORED will prevent any output and the condition will be handled as best it can. WARNING will give a warning message, but the problem will be handled and the simulation will continue. Setting to ERROR will generate an error and the simulation will stop. This will not affect notification policies.

Warning Policy Command	Explanation
DEGENERATE_POLYGONS = *policy*	Degenerate polygons are polygons with zero area and must be removed for the simulation to run. The default policy is WARNING.
HIGH_REACTION_PROBABILITY = *policy*	Generate warnings or errors if reaction probabilities exceed a certain threshold. The default policy is IGNORED. The warnings or errors will be generated both at parse time and during runtime if there are time varying reaction rates that exceed the threshold.
HIGH_PROBABILITY_THRESHOLD = *p*	If the policy is to generate warnings or errors on high probability reactions, have them generated when the probability equals or exceeds *p*. The default value is $1.0 + 10^{-12}$.
LIFETIME_TOO_SHORT = *policy*	Generate warnings if molecules have short lifetimes (which could affect the accuracy of the simulation). This warning occurs after the simulation has ended, so ERROR. is not a valid option. The default policy is WARNING.
LIFETIME_THRESHOLD = *n*	If the policy is to generate a warning if molecules have short lifetimes, then generate warnings on molecules that have an average lifetime of less than *n* iterations. The default value is 50.
MISSED_REACTIONS = *policy*	Generate errors or warnings if there are missed reactions (which usually is a consequence of an overly high reaction probability). This warning occurs after the simulation has ended, so ERROR. is not a valid option. The default policy is WARNING.
MISSED_REACTION_THRESHOLD = *f*	If the policy is to generate a warning if there are missed reactions, then generate a warning for each reaction where a fraction of at least *f* of reactions were missed. The default value is 10^{-3}.
NEGATIVE_DIFFUSION_CONSTANT = *policy*	Diffusion constants cannot be negative, and will be set to zero if they are. The default policy is WARNING.

Warning Policy Command	Explanation
MISSING_SURFACE_ORIENTATION = *policy*	Generate errors or warnings if a molecule is placed on a surface or reactions occur at a surface without a specified orientation—the code will assume you mean that there is no orientation in the warning or silent cases. To avoid triggering this condition, if you want to have no orientation, you must specify it explicitly with `',` or `,'` or `;`. The default policy is ERROR.
NEGATIVE_REACTION_RATE = *policy*	Reaction rate constants cannot be negative, and will be set to zero if they are. The default policy is WARNING.
USELESS_VOLUME_ORIENTATION = *policy*	Generate errors or warnings if a molecule is placed in a volume or reactions occur in free space but an orientation is specified anyway—there is no way to impose orientation so the marks will be ignored. The default policy is WARNING.
ALL_WARNINGS = *policy*	Set all warning policies to the same value (IGNORED, WARNING or ERROR). If ERROR is not a valid choice, the policy will be set to WARNING instead.

3.2 Molecule definition commands

All molecules must be defined by name in a DEFINE_MOLECULES block. For users of MCell 2, note that there is no longer a distinction between a receptor and a ligand. Everything is a molecule, and every different bound state of a receptor must have a unique name (since it must be a unique molecule). The names must be unique in the entire simulation (that is, unique within their own MDL file and any included MDL files that make up the whole simulation).

A define molecule block can be one of the following:

Command	Explanation

DEFINE_MOLECULE *name*
{
define molecule commands
}
	Define a single molecule called *name*. The molecule's properties are specified by commands inside braces.

DEFINE_MOLECULES
{
nameA { *define molecule commands* }
nameB { *define molecule commands* }
...
}
	Define a series of molecules by name. Each molecule's properties are specified by commands inside braces.

Each molecule must have a diffusion constant set using one of the following commands:

Define Molecule Command	Explanation
DIFFUSION_CONSTANT = *D*	This molecule diffuses in space with diffusion constant *D*. *D* can be zero, in which case the molecule doesn't move. Synonyms for this command are DIFFUSION_CONSTANT_3D and D_3D. The units of *D* are cm^2/s.
DIFFUSION_CONSTANT_2D = *D*	This molecule is constrained to a surface and diffuses with diffusion constant *D*. D_2D is a synonym for this command.

The following optional commands can be applied to each molecule (and must appear in this order, and after the diffusion constant is set):

Define Molecule Command	Explanation
CUSTOM_TIME_STEP = *t*	This molecule should take timesteps of length *t* (in seconds). Use either this or CUSTOM_SPACE_STEP, not both.
CUSTOM_SPACE_STEP = *L*	This molecule should take steps of average length *L* (in microns). If you use this directive, do not set CUSTOM_TIME_STEP.
TARGET_ONLY	This molecule will not initiate reactions when it runs into other molecules. This setting can speed up simulations when applied to a molecule at high concentrations that reacts with a molecule at low concentrations (it is more efficient for the low-concentration molecule to trigger the reactions). This directive does not affect unimolecular reactions.

3.3 Reaction definition commands

All reactions must be defined inside a reaction definition block:

Command	Explanation

DEFINE_REACTIONS
{
reaction commands
}
	Define a series of reactions inside braces.

Reactions are specified using arrow notation:

Reaction Command	Explanation
reactants -> *products* [*rate*]	Define a reaction that occurs between one or two *reactants* (names of molecules, separated by +) and produces an arbitrary number of *products* (also separated by +), with a specified *rate*. If a molecule is in the *reactants* list and not in the *products* list, it is destroyed in the reaction. One reactant may be a surface type (see section 3.4.1). The rate can also be a filename, in quotes, that contains two columns: the second is the rate, while the first is the time at which that rate should start being used. This allows variable reaction rates. If you want no products, use the NULL keyword as a placeholder.
reactants -> *products* [*rate*] : *name*	As above, and call the reaction *name* so it can be referred to by count statements.

This notation is perhaps best explained through examples. In the most basic form, reactants and products are just the names of molecules, separated by +:

Example	Explanation
A -> B [100]	Molecule A changes into molecule B at a rate of $100\,s^{-1}$.
A -> A + B [100]	Molecule A emits molecules of B at a rate of $100\,s^{-1}$.
A -> NULL [100]	Molecule A is destroyed at a rate of $100\,s^{-1}$.
A + B -> A [1e6]	Molecule A destroys molecule B at a rate of $10^6\,M^{-1}\cdot s^{-1}$.
A + B -> A + C [1e6]	Molecule A catalytically converts B to C at a rate of $10^6\,M^{-1}\cdot s^{-1}$.
A+B -> A+B+C [1e6]	Collision of A and B catalytically generates C at a rate of $10^6\,M^{-1}\cdot s^{-1}$.

Orientation classes are a fundamentally new concept introduced in MCell3. They replace the MCell 2 idea of POSITIVE_POLE, NEGATIVE_POLE, and BOTH_POLES specifications for receptors.

Molecules in a surface (hereafter surface molecules) have an orientation: up if they point in the direction of the surface normal, down otherwise.. Molecules in 3D (hereafter, volume molecules) that strike or come off a surface also have an orientation: up if the surface normal points towards them, down if it points away.

Reactions normally do not specify absolute orientation. Instead, a reaction lists the required relative orientation of the reactants, and specifies the relative orientation of the products. This allows you to write general reactions that do not depend on the direction of insertion of a molecule into a membrane.

The two orientations are specified by ' and , (apostrophe and comma) after the molecule's name. For example, a surface-bound molecule B has the orientations B' and B,. For example:

Example	Explanation
B' -> B, [10]	Molecule B flips (changes its orientation) at a rate of $10 s^{-1}$.

Example	Explanation
`B' -> B' + A' + C, [10]`	Molecule `B` emits molecules of `A` on the side it's pointing to and emits `C` on the other side, at a rate of $10 s^{-1}$
`B, -> B, + A, + C' [10]`	This specifies exactly the same reaction as above. `B` and `A` end up with the same orientation, while `C` has opposite orientation.

The best way to keep the relationships straight is to draw a "before" picture with each reactant facing the direction of the tick mark, and an "after" picture with each product facing in the direction of the tick mark. Since you can always turn your picture upside-down and still have the same picture, you can also always flip all tick marks and get the same reaction. You can thus use tick marks that are consistent with your mental picture.

If `A` diffuses in 3D but `B` and `C` are on a surface:

Example	Explanation
`A' + B' -> C' [1e5]`	Molecule `A` binds to `B` if it is on the side that `B` is pointing to, producing a `C` facing the same way as `B`, at a rate of $10^5 M^{-1} \cdot s^{-1}$.
`A, + B, -> C, [1e5]`	The same reaction again—everything occurs on the same side, but we wrote it on the bottom this time.
`A' + B, -> C' [1e5]`	Molecule `A` binds when it hits the opposite side of `B`, producing a `C` facing the opposite way as `B` (i.e. towards the side `A` came from), at a rate of $10^5 M^{-1} \cdot s^{-1}$.
`A, + B' -> C, [1e5]`	Same as above.

So far, all examples have used the first orientation class, specified with `'` and `,`. The second orientation class is specified by `''` and `,,`. The third is `'''` and `,,,` and so on. A molecule can be in only one orientation class, and molecules in different classes do not pay attention to each others' orientation. In a reaction where orientation may be a factor, every molecule must be explicitly given an orientation class; by default it is an error to omit orientation (although this behavior can be adjusted to generate warnings or no messages instead; in this case, molecules without an orientation class act without regard to orientation). Therefore:

Example	Explanation
`A'' + B, -> C' [1e5]`	Molecule `A` binds to either side of `B` (since they are in different orientation classes); this produces a `C` facing the opposite way as `B`, at a rate of $10^5 M^{-1} \cdot s^{-1}$.
`A,, + B, -> C' [1e5]`	This is the same reaction—since `A` is the only molecule in the second orientation class, it doesn't matter which way we specify things.
`A,, + B' -> C, [1e5]`	Same again–`B` and `C` still have opposite orientations.
`A, + B' -> C,, [1e5]`	Molecule `A` hits the opposite side of `B` and produces `C` that is equally likely to point either way, at a rate of $10^5 M^{-1} \cdot s^{-1}$.
`A, + B' -> C'' [1e5]`	Same as above, since `C` is still not in the same orientation class as the others.
`A'+B'' -> A,+B''' [1e5]`	Molecule `A` hits molecule `B` on either side; `A` keeps traveling (goes to the other side) and `B` to tumbles to a random orientation, at a rate of $10^5 M^{-1} \cdot s^{-1}$.
`A'+B'' -> C'''+D'''' [1e5]`	`A` and `B` react in any orientation and produce `C` and `D` in random orientations. All orientation classes are different, so there are no geometrical constraints here.

There are examples of how one would use this syntax to model well-known biological reactions at the end of this file.

Tick marks add, so that `',` and `,'` mean no orientation (orientation class 0): reactions will occur from either orientation when given no orientation class, and products will orient randomly. A semicolon, `;`, can be used instead of two opposite

tick marks. Orientations can also be specified numerically inside `{}` after the molecule name. For example, `A{1}` and `A{-1}` are synonyms for `A'` and `A,` and `A{0}` is a synonym for `A;`

There are several variants of the normal reaction arrow `->`. One can use an arbitrary number of dashes in the arrow; `->` `-->` and `------>` all mean the same thing, for instance. In addition, the following arrows have different meanings:

Reaction Arrow	**Explanation**
`->`	A unidirectional reaction going from reactants (on the left) to products (on the right).
`<->`	A bidirectional reaction going in either direction; at most two molecule names can appear on each side. A rate must be given for each direction using the notation `[>`k_+ `, <`k_-`]`, where k_+ is the forward rate constant and k_- is the backward rate constant.
reactant `--` *catalyst* `->` *products*	This specifies a catalytic reaction where *reactant* is converted to *products* in the presence of *catalyst*. This is the same as the reaction *catalyst* + *reactant -> catalyst* + *products*. Presently, there can only be one reactant.
reactant `<-` *catalyst* `->` *product*	A bidirectional catalytic reaction. There can only be one reactant and one product.

3.4 Geometry definition commands

3.4.1 Surface properties

MCell3 allows the user to specify properties of the surfaces of objects. For example, one may wish to specify that a surface does not block the diffusion of molecules. Each type of surface is defined by name, and each surface name must be unique in the simulation and should not match any molecule names. Surface properties are specified inside a surface definition block:

Command	Explanation

DEFINE_SURFACE_CLASS *name*
{
surface property commands
}
	Define a single surface type called *name*. The properties are specified by zero or more commands inside braces.

DEFINE_SURFACE_CLASSES
{
nameA { *surface property commands* }
nameB { *surface property commands* }
...
}
	Define a series of surface types by name.

To define surface properties, use the following commands:

Surface Property Command	Explanation
REFLECTIVE = *name*	The molecule called *name* is reflected by this surface. This is the default behavior for volume molecules; it prevents surface molecules from crossing triangle boundaries. Tick marks on the name allow selective passage of molecules in one orientation relative to the surface.
TRANSPARENT = *name*	The molecule called *name* passes through this surface. This is only meaningful for volume molecules; surface molecules assume that their surface can be traveled in unless it is labeled REFLECTIVE. Tick marks allow the creation of one-way transparent surfaces.
ABSORPTIVE = *name*	The molecule called *name* is destroyed if it touches this surface. Tick marks allow destruction from only one side for volume molecules, or destruction of only one orientation of surface molecules.
CLAMP_CONCENTRATION *name* = *value*	The molecule called *name* is destroyed if it touches the surface (as if it had passed through), and new molecules are created at the surface, as if molecules had passed through from the other side at a concentration *value* (units = M). Orientation marks may be used; in this case, the other side of the surface is reflective. Note that this command is only used to set the effective concentration of a volume molecule at a surface; it is not valid to specify a surface molecule. This command can be abbreviated as CLAMP_CONC.

MOLECULE_DENSITY
{
name1 = *D1*
name2 = *D2*
...
}
	Add the named molecules at the specified densities *D1*, *D2*, ..., (units = μm^{-2}) to every surface with this surface class. Use orientation marks after the name to specify the direction relative to the surface normal. For example, A' specifies a molecule in the same orientation as the surface, while A, specifies the opposite orientation. Using both marks indicates that the molecule should be assigned an orientation randomly.

Surface Property Command	Explanation

MOLECULE_NUMBER
{
name1 = *N1*
name2 = *N2*
...
}
	Add the exact numbers *N1*, *N2*, ..., of molecules onto any region that is made out of this surface class. Note: this usage is not recommended; it is better to add exact numbers of molecules to the region. Orientation marks after the name must be used to specify the direction the molecules are facing.

Note that surface normals are defined by the right-hand rule applied to the vertices in order as listed (see section 3.4.2). Box objects are converted internally into triangles and the surface normals point outwards.

3.4.2 Geometrical objects

Two types of geometrical objects are supported in MCell3. Objects should not have coincident surfaces unless neither surface contains a molecule that any moving molecule can react with. Also, all coincident surfaces should agree on whether they are transparent, reflective, or absorptive to each molecule that might strike them. Geometrical objects can be defined using:

Command	Explanation

name BOX
{
box commands
region commands
transformation commands
}
	This defines a box object called *name*. The shape and position of the box is defined by . Optionally, additional commands can create regions and perform geometrical transformations on the box. Internally, a box is represented as a set of triangles.

name POLYGON_LIST
{
polygon commands
region commands
transformation commands
}
	This defines a polygon list object called *name*. Polygon list objects explicitly give their triangular surface elements.

A variety of optional commands can be used inside a geometrical object definition block, after corners or vertex list / element connections are specified, to modify the basic composition of the object and its surface properties. These are described below. Geometrical transformations are described later, in section 3.4.5.

Box Command	Explanation
CORNERS = [*x1*,*y1*,*z1*],[*x2*,*y2*,*z2*]	The box object has corners as specified. The first coordinates should be less than the second set of coordinates, although MCell3 may fix it if you do it incorrectly.
ASPECT_RATIO = *a*	Make sure that the ratio of the long to short side of each triangle making up the box is no more than *a*. The smallest allowed value is 2. The default is to not care about triangle shape.

Polygon Command	Explanation

VERTEX_LIST
{
[*x0*,*y0*,*z0*]
[*x1*,*y1*,*z1*]
...
}
	Specify the vertices of the triangles inside a polygon list object inside braces. Each vertex is given by its triple [*x*,*y*,*z*]. This command must be given before the ELEMENT_CONNECTIONS command.

Polygon Command	Explanation

ELEMENT_CONNECTIONS
{
[*a0*,*b0*,*c0*]
[*a1*,*b1*,*c1*]
...
}
	Specify the triangles by vertex indices. The vertices are numbered from 0 upwards in the order they were given in the vertex list. The direction of the surface normal is determined by the right-hand rule while following the vertices. Each triangle is given by a triple [*a*,*b*,*c*] of vertex numbers. This command must be given after the VERTEX_LIST command.

Region Command	Explanation

DEFINE_SURFACE_REGIONS
{
nameA {
element specifier commands
regional surface commands
}
name2 { *...* }
...
}
	Define regions on the object. The extent of a region is given by the element specifier commands (at least one is required). Molecules can be added and surface properties can be set with the optional regional surface commands. You can have an arbitrary number of regions on an object, and they may overlap if you wish. Molecules added to overlapping regions accumulate; surface properties are those of the last region applied.

REMOVE_ELEMENTS
{
element specifier commands
}
	Remove the portion of the object specified by the element specifiers. You can think of this as a special type of region that defines the removed portions of the object. No real region exists on any part of the object that has been removed. You can use a list of element numbers/names instead of element specifiers if you wish, but you cannot mix a list of element numbers/names with the element specifier syntax.

Element Specifier Command	Explanation
INCLUDE_ELEMENTS = [*list*]	Include the elements specified by number or name. For polygon objects, these refer to the triangles defined by the element connections, counting from zero upwards in the order given. For boxes, the side names LEFT, RIGHT, FRONT, BACK, BOTTOM, and TOP can be used to refer to the sides, where left/right corresponds to the x axis (left is lower x values), front/back to y, and bottom/top to z. ALL_ELEMENTS refers to the entire object. Numbers can be specified individually (separated by commas) or in ranges with the format *N* TO *M*. The two styles can be mixed (separated by commas).
EXCLUDE_ELEMENTS = [*list*]	Exclude the elements listed. If this is the first element specifier, assume that all elements not listed are included. If not, subtract from the existing list.
INCLUDE_REGION = *name*	Include the existing region on this object called *name* into this region, too.
EXCLUDE_REGION = *name*	Exclude the existing region on this object called *name* from this new region.
INCLUDE_PATCH=[*x1*,*y1*,*z1*],[*x2*,*y2*,*z2*]	This specifier is only valid on box objects, and the corners must define a rectangular patch that is on exactly one side of the box. The box will be divided into triangles in such a way that this patch consists of separate triangles and will form a region.
EXCLUDE_PATCH=[*x1*,*y1*,*z1*],[*x2*,*y2*,*z2*]	Exclude the patch from this region.

After element specifiers, regions can specify a surface type and add extra molecules using:

Regional Surface Command	Explanation
SURFACE_CLASS = *name*	Set the surface type of this region to the previously defined surface class called *name*.
MOLECULE_DENSITY {...}	This is the same as the Surface Property Command of the same name.
MOLECULE_NUMBER {...}	This is the same as the Surface Property Command of the same name. Its usage is recommended here, as a regional surface command, rather than as a surface property command, so that the number of molecules is specified in the same place as the geometry, thus making the density easier to figure out.

3.4.3 Release objects

Release objects place molecules into the world. Release objects provide the only means of placing molecules in a three dimensional space, but some release shapes can place molecules on surfaces as well. Release objects are defined using the following commands:

Command	Explanation

name RELEASE_SITE
{
release site commands
transformation commands
}
	Create a release site called *name*. The shape and method of release is specified by the release site commands. Optionally, geometrical transformations can be applied also.

name CUBIC_RELEASE_SITE {...}	Create a cubic release site called *name*. Molecules are released in a box as specified by the radius. (This is the same as using the SHAPE=CUBIC command inside RELEASE_SITE.)
name SPHERICAL_RELEASE_SITE {...}	Create a spherical release site called *name*. Molecules are released uniformly within the sphere depending on the defined radius of the object. (This is the same as using the SHAPE=SPHERICAL command inside RELEASE_SITE.)
name SPHERICAL_SHELL_SITE {...}	Create a spherical shell release site called *name*. Molecules are distributed on a spherical shell at the defined radius of the object. For now, you must specify the number to distribute, not a concentration. (This is the same as using the SHAPE=SPHERICAL_SHELL command inside RELEASE_SITE.)

DEFINE_RELEASE_PATTERN *name*
{
release pattern commands
}
	Define a new release pattern according to the commands given. A release pattern must be defined for anything other than release at the beginning of the simulation. Release patterns must be defined before they are used. Multiple release sites can use the same pattern.

The following commands define where, what, and when a release object releases molecules:

Release Site Command	Explanation
SHAPE = *geometry*	Release molecules in the specified shape. Valid shapes are CUBIC, SPHERICAL, SPHERICAL_SHELL, and LIST; or the name of region(s) on which to release. Each region must already be instantiated or be inside the same OBJECT as the release site (see OBJECT command). Region names can be combined with + to indicate release on both regions, - to indicate the release occurs on the first and not the second, and * to indicate the release occurs only where the two regions overlap. Parentheses may be used for grouping. Volume molecules will be released in the volume bounded by the regions (each region must be closed); surface molecules will be released on the surface (and regions need not be closed).
LOCATION = [*x*,*y*,*z*]	The release occurs centered at this location. Only used for geometrical shapes.
MOLECULE = *name*	The named molecule is the one that will be released. Not used for the LIST shape. You must specify an orientation if the molecule is a surface molecule.

MOLECULE_POSITIONS
{
name1 [*x1*,*y1*,*z1*]
name2 [*x2*,*y2*,*z2*]
...
}
	The named molecules are added in the locations given. The molecule names must be followed by orientation marks if they have a 2D diffusion constant. If a molecule has a 2D diffusion constant, it will be placed on the surface closest to the coordinate given. This command is used for the LIST shape only.

SITE_DIAMETER = *d* SITE_RADIUS = *r*	For a geometrical release site, this releases molecules uniformly within a diameter *d* or a radius *r*. Not used for releases on regions. With the LIST shape, this is the distance that surface molecules search for a surface before giving up; free molecules pay no attention to this value for the LIST shape.
SITE_DIAMETER = [*x*,*y*,*z*] SITE_RADIUS = [*x*,*y*,*z*]	Release is asymmetric with a different diameters in different directions, as indicated by the vector. Not used for releases on regions or with the LIST shape.
RELEASE_PROBABILITY = *p*	This release does not occur every time, but rather with probability *p*. (If omitted, the default is to release without fail.) Either the whole release occurs or none of it does; the probability does not apply molecule-by-molecule. *p* must be in the interval $[0,1]$.
NUMBER_TO_RELEASE = *n*	Release *n* molecules. For releases on regions, *n* can be negative, and the release will then remove molecules of that type from the region. To remove all molecules of a type, just make *n* large and negative. It is unwise to both add and remove molecules on the same timestep—the order of addition and removal is not defined in that case. This directive is not used for the LIST shape, as every molecule is specified.
CONCENTRATION = *c* DENSITY = *d*	Release molecules at concentration *c* molar for volumes and *d* molecules per square micron for surfaces. Neither can be used for the LIST shape; DENSITY is only valid for regions.

GAUSSIAN_RELEASE_NUMBER
{
MEAN_NUMBER = *n*
STANDARD_DEVIATION = *s*
}
	Release molecules according to a Gaussian distribution with the specified mean and standard deviation.

Release Site Command	Explanation
RELEASE_PATTERN = *name*	Use the named release pattern instead of the default. The default is to release the specified number of molecules at the beginning of the simulation. If *name* is the name of a reaction pathway, the release event will happen every time that reaction happens. The location will then be relative to the site of the reaction, and the z-axis will be rotated to align with the surface normal if the reaction was at a surface. This is much slower than creating products within a reaction, so only use it for special cases (e.g. synaptic vesicle release with a random or very large number of neurotransmitter molecules).

Release patterns are defined as follows.

Release Pattern Command	Explanation
DELAY = *t*	The release pattern will start at time *t*. (Default is to start at time zero.)
RELEASE_INTERVAL = *t*	During a train of releases, release molecules after every *t* seconds. Default is to release only once ($t = \infty$).
TRAIN_DURATION = *t*	The train of releases lasts for *t* seconds before turning off. Default is to never turn off ($t = \infty$).
TRAIN_INTERVAL = *t*	A new train of releases happens every *t* seconds. Default is to never have a new train ($t = \infty$). The train interval must not be shorter than the train duration.
NUMBER_OF_TRAINS = *n*	Repeat the release process for *n* trains of releases. Default is one train.
NUMBER_OF_TRAINS = UNLIMITED	Repeat trains forever.

3.4.4 Instantiation, grouping, and modification of objects

An object is a box, polygon, release site, or a metaobject which contains other objects. Metaobjects are defined and modified using

Command	Explanation

name OBJECT
{
object specifier commands
transformation commands
}
	Define a new object called *name*. Inside the braces, list other objects one at a time to be added (see below).

INSTANTIATE *name* OBJECT { ... }	Same as above, except we also insert the object into the world. A simulation must have at least one INSTANTIATEd object.

MODIFY_SURFACE_REGIONS
{
nameA[*regA1*] {
regional surface commands
}
nameB[*regB1*] { ... }
...
}
	This modifies surface regions on existing objects via their name and region name. Element lists may not be changed, but otherwise all regional surface commands are available. The full name must be given in the case of separate objects (using *name1*.*name2* to refer to objects inside metaobjects). If an object is included in a metaobject, then has a surface region modified, and is included in another metaobject, the surface regions will differ in those the two metaobjects.

You can define release sites, boxes, and polygon objects inside another object, as well as placing previously defined objects into existing ones:

Object Specifer Command	Explanation

newname `OBJECT` *oldname*
`{`
transformation commands
`}`
	Add the existing object called *oldname* into the existing object and label it *newname*. You can add extra commands (e.g. transformation) inside the braces. The old and new names can be the same thing. Thereafter, this object can be referred to in the world as *name*.*newname*.

name `BOX {...}`	Create a box inside the existing object (using the same syntax as previously defined).
name `POLYGON_LIST {...}`	Create a polygon list object inside the existing object (using the same syntax as previously defined).
name `RELEASE_SITE {...}`	Create a release site inside the existing object.
newname `OBJECT {...}`	Create an object inside the existing object.

3.4.5 Geometrical transformations

At the end of the definition of a release object or geometrical object, or in the block where an object is instantiated, it can be moved using the following transformation commands (placed at the end of the block before the closing brace).

Transformation Command	Explanation
`TRANSLATE = [`*x*`,`*y*`,`*z*`]`	Move the object by the specified vector.
`SCALE = [`*x*`,`*y*`,`*z*`]`	Scale the object by multiplying each coordinate by the corresponding value in the vector.
`ROTATE = [`*x*`,`*y*`,`*z*`] ,` *A*	Rotate *A* degrees about the axis defined by the supplied vector.

3.5 Output specification commands

There are two forms of output in MCell3, visualization output and count output. Visualization output typically contains the molecules and/or geometry of the model in a form suitable for visualization or analysis that requires knowledge of the precise location of particles. Count output reports running totals of summary statistics such as the total number of molecules of a certain type in the world, the number of times a reaction has occurred inside some object in the world, and so on. Count output can also be written when triggered by a specific event such as a reaction taking place.

3.5.1 Visualization Output

Command	Explanation

VIZ_OUTPUT
{
viz output commands
}
	Define a new visualization output block. MDL files can have multiple VIZ_OUTPUT blocks.

Each viz output block consists of the following commands:

Viz Output Command	Explanation
MODE = *viz_mode*	Specifies the mode of the visualization output. The mode defines the directory structure and number of files comprising the visualization output. The valid values for are DREAMM_V3, DREAMM_V3_GROUPED, and DX. The DX mode requests the old MCell2 style of output format for compatibility purposes. The default is DREAMM_V3.
FILENAME = "*filename_specifier*"	Name of the master header file containing all information for DREAMM and references to the multiple binary data files. If *filename_specifier* contains a path, the relevant directory structure pointed to has to be created by the user before the start of the simulation.

MESHES
{
data output block
}
	Defines meshes visualization data output block.

MOLECULES
{
data output block
}
	Defines molecules visualization data output block.

Each data output block consists of the following commands:

Data Output Block Commands	Explantation

NAME_LIST
{
name list commands
}
	Defines a valid name list. The valid values are either names separated by any type of whitespace, strings with wildcards (in quotes) that match names, or keywords defined below. All children of the named objects are included by default. If this statement occurs in a MESHES block, the names should be names of objects; in a MOLECULES block they should be names of molecules.

TIME_POINTS
{
data type @ *time_points_list*
}
	Defines what data should be output at what times. The data types are given below and valid notations for *time_points_list* are [*time1*], or [*time1*, *time2*, ..., *time_end*], or [*time1*, *time2*, [*time3* TO *time_end* STEP *delta_time*]], or ALL_TIMES. Mutually exclusive with ITERATION_NUMBERS.

Data Output Block Commands	**Explantation**

ITERATION_NUMBERS
{
data type @ *iterations_numbers_list*
}
	Defines what data should be output at what iterations. The data types are given below and valid notations for *iteration_numbers_list* are [*iteration1*], or [*iteration1*, *iteration2*, ..., *iteration_end*], or [*iteration1*, *iteration2*, [*iteration3* TO *iteration_end* STEP *delta_iteration*]], or ALL_ITERATIONS. Mutually exclusive with TIME_POINTS.

The following name list commands for MESHES and MOLECULES are available:

Name list Commands (MESHES)	**Explanation**
ALL_MESHES	All mesh object names should be included in the NAME_LIST sub-block inside a MESHES block. ALL_MESHES is equivalent to naming the top-level mesh object (assuming that only a single INSTANTIATE block is present).

Name list Commands (MOLECULES)	**Explanation**
ALL_MOLECULES	All molecule names should be included in the NAME_LIST sub-block inside MOLECULES block.

The following data type commands for MESHES and MOLECULES are available:

Data types (MESHES)	**Explanation**
GEOMETRY	Mesh vertex and connectivity information should be written at the specified time/iteration.
REGION_DATA	Mesh region information should be written at the specified time/iteration.
ALL_DATA	Equivalent to using both GEOMETRY and REGION_DATA.

Data types (MOLECULES)	**Explanation**
POSITIONS	Molecule position information should be written at the specified time/iteration.
ORIENTATIONS	Molecule orientation information should be written at the specified time/iteration.
ALL_DATA	Equivalent to using both POSITIONS and ORIENTATION.

There are two possible visualization output file formats. DREAMM_V3 mode is the default, and creates files in native DX format. This mode is optimized for speed of visualization, but creates many individual files. It has a directory structure with the top-level directory given by adding _viz_data to the filename, *i.e.* *filename*_viz_data. For example if FILENAME = "./viz_data/diffusion_box" then the directory diffusion_box_viz_data will be created inside the ./viz_data directory. Please note that any directory structure above diffusion_box_viz_data if specified has to be created manually by the user before the beginning of the simulation. Inside the *filename*_viz_data directory there is the data directory called frame_data and three files:

- *filename*.dx (the header file)
- *filename*.iteration_numbers.bin
- *filename*.time_values.bin

The directory frame_data contains a number of sub-directories named by combining the word iteration_ with the iteration number of the simulation, such as iteration_0, iteration_20, etc. Each of these iteration sub-directories by itself contains up to nine files:

- meshes.dx (header file for meshes)
- mesh_positions.bin
- mesh_states.bin (*optional*)
- region_indices.bin
- surface_molecules.dx (header file for surface molecules)
- surface_molecules_orientations.bin
- surface_molecules_positions.bin
- surface_molecules_states.bin (*optional*)
- volume_molecules.dx (header file for volume molecules)
- volume_molecules_orientations.bin
- volume_molecules_positions.bin
- volume_molecules_states.bin (*optional*)

Visualization data output for the DREAMM_V3_GROUPED mode is in native DX format and includes one master header file and seven binary data files, plus up to two optional data files if state values are specified in the NAME_LIST blocks:

- *filename*.dx (the master header file)
- *filename*.mesh_positions.bin
- *filename*.mesh_states.bin (*optional*)
- *filename*.region_indices.bin
- *filename*.molecule_positions.bin
- *filename*.molecule_orientations.bin
- *filename*.molecule_states.bin (*optional*)
- *filename*.iteration_numbers.bin
- *filename*.time_values.bin

Because the DREAMM_V3_GROUPED mode produces a small number of files, they each may become very large. Hence, reading the files may be slow, but this mode may be best for use on production (supercomputer) machines to avoid transfering large number of files.

All of the keywords in the VIZ_OUTPUT block are optional except FILENAME. If the user does not specify the FILENAME keyword an error message is printed and the simulation aborted. Some of the binary files for both formats may be empty. For example, if no regions are defined the file region_indices.bin will be empty. Similarly, if no meshes or molecules are defined the corresponding mesh_positions.bin or all molecules related binary files will be empty. This avoids unintentional mixing of pre-existing and new files that could result during several runs if incomplete file sets were to be generated with the same names. In DReAMM, the user will only need to point to the *filename*.dx file, and the data from the binary files will be imported as needed for different frames. While using checkpointing in case of the DREAMM_V3_GROUPED format the resulting visualization output files add the checkpoint sequence number to their names, like *filename*.mesh_positions.1.bin. In the case of the DREAMM_V3 format only three files - *filename*.dx, *filename*.iteration_numbers.bin and *filename*.time_values.bin will add the checkpoint sequence number to their names since all the data for each iteration is stored separately in the corresponding iteration_# subdirectory.

Examples of VIZ_OUTPUT statements are given below.

Short-hand #1 (time style):

```
VIZ_OUTPUT {
  FILENAME = "viz_data/output_example"
  MESHES {
    NAME_LIST { ALL_MESHES /* or list of object names */ }
    TIME_POINTS { ALL_DATA @ [0] }
  }
  MOLECULES {
    NAME_LIST { ALL_MOLECULES /* or list of molecule names */ }
    TIME_POINTS { ALL_DATA @ ALL_TIMES }
  }
}
```

Short-hand #2 (iterations style):

```
VIZ_OUTPUT {
  FILENAME = "viz_data/output_example"
  MESHES {
    NAME_LIST { ALL_MESHES /* or list of object names */ }
    ITERATION_NUMBERS { ALL_DATA @ [0] }
  }
  MOLECULES {
    NAME_LIST { ALL_MOLECULES /* or list of molecule names */ }
    ITERATION_NUMBERS { ALL_DATA @ ALL_ITERATIONS }
  }
}
```

Expanded case:

```
VIZ_OUTPUT {
  FILENAME = "viz_data/output_example"
  MESHES {
    NAME_LIST { ALL_MESHES /* or list of object names */ }
    TIME_POINTS {
      GEOMETRY @ [0]
      REGION_DATA @ [0]
    }
  }
  MOLECULES {
    NAME_LIST { ALL_MOLECULES /* or list of molecule names */ }
    TIME_POINTS {
      POSITIONS @ ALL_TIMES
      ORIENTATIONS @ ALL_TIMES
    }
  }
}
```

Usual UNIX-style wildcards like "*" and "?" are allowed in the *name_list* but must be enclosed in quotes. For example in the case of `MOLECULES` the following `NAME_LIST` statements are all valid:

```
NAME_LIST{A B C1 C2 C3}
NAME_LIST{A B "C*"}
NAME_LIST{A B "C?"}
```

Each `MESHES / NAME_LIST` statement may contain a single mesh object name or multiple mesh object names with optional state values. It is left to the user to avoid possible confusion arising from overlapping object trees within a single master header file and its associated data files.

3.5.2 Reaction Data Output

Command	Explanation

REACTION_DATA_OUTPUT
{
reaction output commands
}
	Define a new count data output block which contains the commands below. Each MDL file can have multiple reaction data output blocks.

Each reaction data output block consists of the following commands:

Reaction Output Command	Explanation
OUTPUT_BUFFER_SIZE = *N*	Write output to disk after every *N* lines. The default is *N*=10000. This command is optional, but must be first if it is used. The output will also always be written when the simulation terminates, regardless of *N*.
STEP = *t*	Output this block every *t* seconds. Exactly one of STEP or the following two commands should be used. Triggered output ignores the values specified, but some value must still be given.
TIME_LIST = [*list*]	Output this block at the times specified in the list.
ITERATION_LIST = [*list*]	Output this block at the iteration numbers specified in the list (i.e. after that number of timesteps).
HEADER = *setting*	Output blocks by default have no header but can optionally have a header line that states the output (name of molecule, reaction, etc.) in each column. This command can set the behavior of that header line; it applies to all output files until the next HEADER line. A *setting* of ON turns on the header line; OFF prevents any header. A string, in quotes, will turn the header on and prepend the string to the line; this is useful to add comment character(s). For example, "//" would add a C++-style comment prefix to the line. For TRIGGER statements (see below), the column label (plus comment character if specified) is appended to each line of output when headers are on.
{ *value* } => "*file*"	Output the value in braces to the filename in quotes. The first column will be the time (in seconds) of the iteration unless the ITERATION_LIST specifier is used, in which case the first column will be the iteration number. For count values, the second column will be the value listed; other possibilities appear later in this document. This command, and the variants listed below, can be repeated to send different output to many files. The output symbol => has several variants which are described below.
{ *value* : "*name*" } => "*file*"	Output the value in braces with the column header string *name* to the filename *file*. Not valid if *value* is found using wildcards. Trigger outputs put this header in the rightmost column on each line; count outputs put the name at the top of the appropriate column.
{ *value* , *value* , ... } => "*file*"	cFor counts, output the list of values in braces, one to a column, in the order listed. The first column will be the time/iteration number; successive columns will be the values in the order listed. If headers are on, each column header can be customized by specifying : "*name*" after the value. For triggers, all the specified events will be combined into one file.

The values specified in braces are either TRIGGER statements, COUNT statements, or mathematical operations applied to COUNT statements (e.g. you can add, subtract, etc. COUNT statements to each other and to constants and so on). Wildcards can be used to select multiple molecules or reactions by name, but in this case mathematical operations cannot be used. The wildcards ? and * can be used to match any single character and any sequence of characters,

respectively; internally, this will generate one count statement per matching name (and therefore will create multiple columns of output). Having headers on is convenient in this case, so one can tell which column corresponds to which name.

If a simulation starts from a checkpoint file, it will add to any existing output files. Otherwise, the output files will be overwritten if they already exist.

COUNT statements are either *instantaneous*, and give information about the state of the model at the instant the count is output—the number of molecules in a region, for example—or are *cumulative*, and count the number of events that have occurred since the beginning of the simulation. Alternatively, they can output the time and location of each reaction or molecular collision of the type specified.

The COUNT statements themselves have the following syntax:

Count Statement	Explanation
COUNT[*name* , WORLD]	Count molecules or reactions in the world. If *name* refers to a molecule, this is an instantaneous count of the number of copies that molecule in the world. If *name* refers to a reaction, count how many times that reaction has occurred since the beginning of the simulation. If "*name*" is in quotes, in this command or any of the following commands, the string in quotes can contain wildcards which will be matched to molecule and reaction names and will be listed in alphabetical order. It is usually a good idea when using wildcards to turn on headers so one can see which column is which.
COUNT[*name* , *object*]	Count molecules (reactions not implemented yet) inside the object called *object*. This must be an instantiated object. For example, if you have instantiated an object called my_world with a box called my_box inside it, *object* would be my_world.my_box. If you are counting surface molecules or reactions at a surface, only the ones that actually occur on *object* will be counted (not those inside). Molecules with a 3D diffusion constant will be counted inside the object, but the object must be closed. All counts are instantaneous.
COUNT[*name* , *region*]	Count molecules (reactions not implemented) inside the named region. It must be referenced fully. E.g. if my_box (from above) has a region called my_region, the name would be my_world.my_box[my_region]. The count is instantaneous.
COUNT[*name*,*region*,ALL_ENCLOSED]	Count all molecules (reactions not implemented) that occur inside this region (not counting those that occur on the surface of the region). This lets you count surface molecules contained on surfaces that lie within a box, for example. This will work with object names as well as region names, but the object or region must be closed. It is only useful for surface molecules. The count is instantaneous.
COUNT[*name*,*region*,ESTIMATE_CONC]	Estimate the concentration of the molecule at that region, averaged since the beginning of the simulation (output has units of μM). A single object can be used instead of a region. The region/object does not need to be closed. To find the average concentration during one count interval, let t_i be the time of the ith output, let t_j be some earlier output, and let $\bar{c}(t)$ be the concentration averaged up to time t. Then the average concentration between times t_j and t_i is $\bar{c}(t_j \rightarrow t_i) = \frac{t_i \bar{c}(t_i) - t_j \bar{c}(t_j)}{t_i - t_j}$. Note that this is the concentration all around the surface, so if the molecule can only reach one side, the concentration on that side will be twice what is reported here. The command can be given verbosely as ESTIMATE_CONCENTRATION. The estimate is based on a cumulative count.

Count Statement	Explanation
COUNT[*name*, *region*, *hits*]	Output the number of times the named molecule has hit the named region (or object). The *hits* specifier should be one of FRONT_HITS, BACK_HITS, ALL_HITS, FRONT_CROSSINGS, BACK_CROSSINGS, and ALL_CROSSINGS. The count is cumulative.
EXPRESSION[*expression*]	Evaluate and output a mathematical expression. This can be mixed with COUNT statements but not with TRIGGER statements.

Cumulative counts are reset when a simulation is started from a checkpoint. This breaks ESTIMATE_CONC, but the other cumulative counts can be recovered by adding the last report before the checkpoint to the first one after the checkpoint.

TRIGGER statements output the time and location each time the number of molecules changes or a reaction happens. TRIGGER statements are not compatible with the WORLD or the ESTIMATE_CONC directives. Within output statements pointing to the same output file, there can only be a single TRIGGER command, i.e., it can not be mixed with other TRIGGER or COUNT statements. Please note that trigger outputs are not guaranteed to specify a point on the boundary of a region when a molecule diffuses out. Surface molecules diffuse by hopping, not raytracing, so when they leave a region, the last point occupied is printed. Volume molecules that cross boundaries will typically have their positions reported as slightly inside the boundary. Locations of reactions and hits are exact, however.

TRIGGER statements obey the following syntax

Trigger Statement	Explanation
TRIGGER[*molecule*, *region*]	Generates output each time the number of molecules inside the specified region changes. The output has the format *iteration time exact time X Y Z orientation number [name].* Here, the first column gives the time of the iteration during which the event happened and the second column the exact time the event was scheduled. The third through fifth columns are the x, y, and z coordinates of the location where the event took place. The sixth column gives the molecule orientation, i.e., it is 0 for volume molecules and +/-1 for surface molecules according to their orientation with respect to the surface containing them. The seventh column can take on values of +/-1 depending on if the molecule was added or removed, respectively. If HEADER is on, the eighth column lists the molecule name.
TRIGGER[*reaction*, *region*, ALL_ENCLOSED]	Generates output each time the specifed reaction takes place inside the named closed region. In the case of reactions involving surface molecules, the region needs to enclose the surface containing the molecules and can not be the surface that contains the molecules itself. The output has the format *iteration time exact time X Y Z [name].* The fields present have the same meaning as explained above. Since reactions do not have an orientation and always occur one at a time the corresponding *orientation* and *number* fields are omitted. If HEADER is on, the sixth column lists the reaction name. For reactions with volume molecules, the ALL_ENCLOSED keyword is optional.
TRIGGER[*molecule*, *region*, *hits*]	Generates output each time the molecule hits or crosses the named region. The *hits* specifier should be one of FRONT_HITS, BACK_HITS, ALL_HITS, FRONT_CROSSINGS, BACK_CROSSINGS, and ALL_CROSSINGS. The output has the format *iteration time exact time X Y Z orientation [name].* All entries besides the *orientation* field have the same meaning as explained above. The *orientation* column can take on values of +/-1 depending on if the region was hit or crossed from the front or the back, respectively. The *number* column is omitted here. If HEADER is on, the seventh column lists the molecule name.

The following output symbols can be used in place of => and give the behaviors described below. All output symbols will create files if none exist. No output symbols will create directories—if the files that are referred to cannot be created as specified, MCell3 will quit with an error message. Output may create empty files if the simulation ends without producing output (either because of an error condition or because the simulation did not run long enough to reach the time/iteration of any reaction data output).

Output Symbol	Explanation
=>	If a checkpoint file is not used, overwrite the existing file (with headers if requested). If a checkpoint file is used, discard any of the output file that appears to be a later time than the start of the current run, and append to the file from that point. Headers are not written unless the file has to be created or is empty to begin with. This command generally does "what you expect"—after the simulation has run, it will contain data from earlier in the simulation that the current run, plus the data created in the current simulation. If you switch between `ITERATION_LIST` and other output time specifiers, this command won't know whether output is by time or by iteration number, so don't use this command if you switch from one to the other after checkpointing.
>	Always overwrite the file, whether or not a checkpoint is used. If headers are requested, they will appear at the beginning of the file.
+>	Always create a new file, whether or not a checkpoint is used. If a file of the given name already exists and is not empty, MCell3 will print an error message and exit. If headers are requested, they will appear at the beginning of the file.
>>	Always append to an existing file without removing any previous data. Headers are only written if the file starts out empty or has to be created.
>>>	Always append to an existing file without removing any previous data and if headers are requested, write them even into the middle of the file.

3.5.3 Other Output Commands

The MCell 2 style `VIZ_DATA_OUTPUT` block is also supported (a maximum of one per MDL file) for backwards compatability. It is no longer explicitly supported, however, so the format is not described here.

3.6 Utility commands

MCell3 understands the standard numeric operations `+ - * /` as well as the following standard numerical functions:

Numerical Command	Explanation
`SQRT(x)`	Return the square root of *x*
`EXP(x)`	Return the value of *e* raised to the x^{th} power
`LOG(x)`	Return the natural logarithm of *x*
`LOG10(x)`	Return the base 10 logarithm of *x*
`SIN(x)`	Return the sine of *x*
`COS(x)`	Return the cosine of *x*
`TAN(x)`	Return the tangent of *x*
`ASIN(x)`	Return the inverse sine of *x*
`ACOS(x)`	Return the inverse cosine of *x*
`ATAN(x)`	Return the inverse tangent of *x*
`ABS(x)`	Return the absolute value of *x*
`CEIL(x)`	Return the smallest integer at least as big as *x*
`FLOOR(x)`	Return the largest integer at no bigger than *x*
`MAX(x,y)`	Return the larger of *x* and *y*
`MIN(x,y)`	Return the smaller of *x* and *y*
`RAND_UNIFORM`	Return a random number uniformly distributed between 0 and 1
`RAND_GAUSSIAN`	Return a random number from a Gaussian distribution with mean 0 and standard deviation 1.
`PI`	The numeric value $\pi = 3.14159265358979323846$
`SEED`	The value of the random number generator seed

At any outer block in MCell3, one can define variables simply by assigning a value to the name of the variable. E.g. `my_lucky_number=13` would be a valid (if unusual) way to define a variable. Variables can take numeric, array, or string values. String values consist of text between double quotes. Array values are lists of numbers inside brackets separated by commas, or starting and ending values plus a step size, as exemplified below (note the double brackets):

```
my_lucky_number = 13
my_favorite_array = [1,3,5,7,11,17]
my_second_favorite_array = [[1.3 TO 2.75 STEP 0.331]]
my_boring_string = "la la la, la la la"
```

The C-style `printf` and `sprintf` commands work too, pretty much the way you'd expect them to. MCell3 comments are delimited by `/*` and `*/`.

MDL files can include other MDL files using the following syntax:

Command	Explanation
`INCLUDE_FILE = "`*filename*`"`	Parse the text in *filename* as if it were inserted into this MDL file at this point.

Paths are relative to the location that MCell was run from, not relative to the MDL file being parsed.

4 Technical details affecting simulation speed and accuracy

4.1 Partitioning

In future releases, MCell3 will automatically partition space to improve execution speed. Currently, however, this must be performed manually. In general, partitions should be chosen to avoid having too many surfaces and molecules in one subvolume defined by the partitions. Molecules that are specified as `TARGET_ONLY` or which do not interact with other molecules diffusing in 3D need only have relatively few surfaces in one subvolume.

If there are few surfaces and/or molecules in a subvolume, it is advantageous to have the subvolume as large as possible. Crossing partition boundaries takes a small amount of time, so it is rarely useful to have partitions more finely spaced than the average diffusion distance of the faster-moving molecules in the simulation.

In cases where the diffusing molecules do not interact with each other, they can safely take extended time-steps by measuring how far they are from things they could interact with. In this case, the partitions with no surfaces should be as large as possible. For example, a box works well with partitions just inside its outer walls.

Finally, note that partition placement is not exact. The model is divided into 16384 possible partition boundaries, so partitions may shift by up to about one part in twenty thousand of the size of the model. For instance, if the model has a structure that is 6μm long, partitions may vary by about 0.0003μm. Thus, do not place partitions too close to objects in your model or they may not appear on the side you expect them to appear.

4.2 Mean diffusion distance

Diffusion in MCell3 (and in earlier versions of MCell) is modeled as a series of motions in a straight line. This is a good approximation around geometry that is of a larger scale than the mean diffusion length for the timestep of the molecule in question. For accurate results around intricate geometry, it may be necessary to reduce the time step (or space step).

4.3 Reaction probabilities

MCell3 assigns a probability of reaction for each collision. These probabilities are chosen to match the bulk reaction rate specified in the MDL file. The match will not occur, however, if the probability goes above 1.0. Internal correction factors can also raise the actual probability above the typical probability specified at the beginning. Therefore, MCell3 will output a warning if the reaction probability goes above 0.8 for reactions where a volume molecule hits a surface, or if the probability goes above 0.3 for a collision between pairs of volume molecules. *!!* Is this even true? *!!*

If warnings are given (and possibly even if they are not), one should reduce the time step to lower the probabilities and see if the same results are generated. If not, simulations should be run with shorter time steps in order to avoid overly high probabilities.

Unimolecular reactions with half-lives of less than one time step are also not perfectly accurate. Although unimolecular transitions will always occur at the right rate, other molecules may not experience the right effective concentration of each state, since a short-lifetime species may not be converted to another species until the end of the time step after which many other molecules may have had a chance to interact with it. Thus, the shortest-lifetime species in a series of unimolecular transitions should not have a half-life of less than approximately one time step if other molecules can interact with that state.

4.4 Interaction radii

Bimolecular reactions occur within a distance specified by the `INTERACTION_RADIUS` command. In many cases, one may want to increase or decrease this value. In particular, in order to get the right probability of reaction, MCell3 increases the probability of reaction when near surfaces.

If ACCURATE_3D_REACTIONS is set to FALSE, MCell3 also treats partition boundaries as opaque and increases the probability of reaction rather than looking for molecules on the other side of the partition. This speeds execution time but can lead to error, the reaction rate has approximately 1-2% error if the average spacing between surfaces is at least 10 times the interaction radius, and the reaction probabilities are 0.3 or less. For example, if one has partitions spaced 0.02μm apart, simulation accuracy will be poor with the default interaction radius of 0.01μm. Thus, one might wish to specify INTERACTION_RADIUS=0.001.

4.5 Placing molecules in the world

There are two ways to place molecules on surfaces: with a release site on a region, and as part of the property of a surface or region. Release sites are more flexible but slower; if you do not need the flexibility of release site notation, you're better off defining a region and using the MOLECULE_DENSITY or MOLECULE_NUMBER commands to add molecules at initialization.

All placement of molecules in volumes is done with release sites. However, the geometrical release sites (CUBIC and SPHERICAL) require less computation to place each molecule. Thus, these should be used preferentially for simple geometry. To release many particles at a one point, use a cubic release site and set the diameter to 0. To release many particles at different points, use the LIST release type.

5 Example models

5.1 Ligand-gated ion channel

Below are a set of molecule definitions and reactions that specify an ion channel that is gated by the binding of a single ligand.

```
DEFINE_MOLECULES {
  channel_unbound { D_2D=0 }
  channel_bound   { D_2D=0 }
  channel_open    { D_2D=0 }
  ligand          { D_3D=2e-8 }
  ion             { D_3D=3e-8 }
}
DEFINE_REACTIONS {
  channel_unbound' + ligand' -> channel_bound'             [1e7]
  channel_bound'             -> channel_unbound' + ligand' [2e2]
  channel_bound'             -> channel_open'              [5e2]
  channel_open'              -> channel_open'    + ion,    [8e4]
}
```

We have defined a reaction where a ligand binds to one end of a channel (presumably the extracellular face), which causes the channel to be in its bound state. In that state it can either release the ligand or become open. While open, it will emit ions on the other end (presumably the intracellular face). This would be suitable if the ion concentration is much higher outside than inside, or the membrane potential makes it highly favorable for the ion to move inside, so that we don't have to worry about the reverse reaction. If there is no electrical driving force, we might have to model ions both inside and outside:

```
DEFINE_REACTIONS {
  channel_unbound' + ligand' -> channel_bound'             [1e7]
  channel_bound'             -> channel_unbound' + ligand' [2e2]
  channel_bound'             -> channel_open'              [5e2]
  channel_open'' + ion'      -> channel_open''     + ion,  [4e7]
}
```

Here, the ion travels in either direction just as easily since it pays no attention to the orientation of the channel. However, if there was a modest driving force, traveling in might be easier than traveling out, which would be reflected in the rates.

```
DEFINE_REACTIONS {
  channel_unbound' + ligand' -> channel_bound'             [1e7]
  channel_bound'             -> channel_unbound' + ligand' [2e2]
  channel_bound'             -> channel_open'              [5e2]
  channel_open' + ion'       -> channel_open'    + ion,    [4e8]
  channel_open' + ion,       -> channel_open'    + ion'    [1e8]
}
```

In this case, the ion is four times as likely to travel from outside to inside as inside to outside.

5.2 Example bimolecular reaction

Here's a complete MDL file that implements a simple bimolecular reaction that should achieve equilibrium at 482 molecules of each species.

```
time_step = 1.0e-6
TIME_STEP = time_step
TIME_STEP_MAX = time_step
ITERATIONS = 1e-2/time_step
EFFECTOR_GRID_DENSITY = 10000
INTERACTION_RADIUS = 0.001
PARTITION_X = [ [-0.1 TO 0.1 STEP 0.01] ]
PARTITION_Y = [ [-0.1 TO 0.1 STEP 0.01] ]
PARTITION_Z = [ [-0.1 TO 0.1 STEP 0.01] ]
DEFINE_MOLECULES
{
  A { D_3D = 100e-8 }
  B { D_3D = 100e-8 }
  C { D_3D = 100e-8 }
}
/* Your basic reversable binding reaction */
DEFINE_REACTIONS
{
  A + B -> C [1e7]
  C -> A + B [1e3]
}
small_box BOX
{
  CORNERS = [-0.1,-0.1,-0.1] , [0.1,0.1,0.1]
  /* REMOVE_ELEMENTS { TOP,LEFT } */  /* Could remove sides ... */
  /* REMOVE_ELEMENTS { INCLUDE_PATCH = [0.1,0,0] , [0.1,0.05,0.05] } /*... or patch*/
}
INSTANTIATE my_world OBJECT
{
  A_release CUBIC_RELEASE_SITE {
    LOCATION=[0,0,0]
    MOLECULE=A
    NUMBER_TO_RELEASE=482
    SITE_DIAMETER=0.196
  }
  B_release CUBIC_RELEASE_SITE {
    LOCATION=[0,0,0]
    MOLECULE=B
    NUMBER_TO_RELEASE=482
    SITE_DIAMETER=0.196
  }
  C_release CUBIC_RELEASE_SITE {
    LOCATION=[0,0,0]
    MOLECULE=C
    NUMBER_TO_RELEASE=482
    SITE_DIAMETER=0.196
  }
 my_box OBJECT small_box {}
}
```

```
REACTION_DATA_OUTPUT
{
  STEP = 1e-5
  { COUNT [A,WORLD] } => "eq_A.dat"
  { COUNT [B,WORLD] } => "eq_B.dat"
  { COUNT [C,WORLD] } => "eq_C.dat"
}
```

6 Authors

The following authors have contributed to this document, as described.

- Rex Kerr: Initial layout and design, first draft of document, editing of later drafts.
- Markus Dittrich: `VIZ_DATA` section.