

MCell3 Quick Tutorial and Reference Guide

Rex Kerr

5th April 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 L^AT_EX 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 <i>N</i>	Start with random number seed <i>N</i> instead of 1 (the default).
-iterations <i>N</i>	Run the simulation with <i>N</i> timesteps (overrides the value in the mdl file)
-help	Print out a basic help screen.
-info	Same as -help
-logfile <i>filename</i>	Send messages to <i>filename</i> instead of stdout/stderr
-logfreq <i>N</i>	Valid syntax, but not implemented yet.
-checkpoint_infile <i>filename</i>	Use <i>filename</i> 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.

Command	Explanation
SPACE_STEP = <i>N</i>	Have all diffusing molecules take time steps of different duration, chosen so that the mean diffusion distance is <i>N</i> microns for each molecule. By default, all molecules move the same time step.
EFFECTOR_GRID_DENSITY = <i>N</i>	Tile all surfaces so that they can hold molecules at <i>N</i> different positions per square micron. The default is 10000.
INTERACTION_RADIUS = <i>N</i>	Diffusing molecules will interact with each other when they get within <i>N</i> microns of each other. The default is 0.01.
PARTITION_ <i>D</i> = [<i>list</i>]	Subdivide the <i>D</i> 'th axis of space, where <i>D</i> is X, Y, or Z, at the boundaries given in <i>list</i> (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 = <i>N</i>	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 = <i>N</i>	Specifies how many distances to put in the diffusion lookup table. Again, the default is sensible. FULLY_RANDOM is not implemented here.
ACCURATE_3D_REACTIONS = <i>boolean</i>	Specifies which method to use for computing 3D molecule-molecule interactions. If <i>boolean</i> is TRUE, then molecules will look through partition boundaries for potential interacting partners—this is slower but more accurate. If <i>boolean</i> 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 = <i>boolean</i>	If <i>boolean</i> 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 currently TRUE, but this should probably be changed.
VACANCY_SEARCH_DISTANCE = <i>r</i>	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 <i>r</i> 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, <i>r</i> 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 <i>r</i> , and will place those products as close as possible to the place where the reaction occurs (deterministically, so small-scale directional bias is possible).

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
<pre>DEFINE_MOLECULE <i>name</i> { <i>define molecule commands</i> }</pre>	Define a single molecule called <i>name</i> . The molecule's properties are specified by commands inside braces.
<pre>DEFINE_MOLECULES { <i>nameA</i> { <i>define molecule commands</i> } <i>nameB</i> { <i>define molecule commands</i> } ... }</pre>	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 . Until surface diffusion is implemented, only 0 is a valid value for 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
<pre>DEFINE_REACTIONS { <i>reaction commands</i> }</pre>	Define a series of reactions inside braces.

Reactions are specified using arrow notation:

Reaction Command	Explanation
<i>reactants</i> -> <i>products</i> [<i>rate</i>]	Define a reaction that occurs between one or two <i>reactants</i> and produces an arbitrary number of <i>products</i> , with a specified <i>rate</i> . If a molecule is in the <i>reactants</i> list and not in the <i>products</i> list, it is destroyed in the reaction. One reactant may be a surface type (see section 3.4.1). <i>!!Need to describe rates from files!!</i>
<i>reactants</i> -> <i>products</i> [<i>rate</i>] : <i>name</i>	As above, and call the reaction <i>name</i> 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 100s^{-1} .
A -> A + B [100]	Molecule A emits molecules of B at a rate of 100s^{-1} .
A + B -> A [1e6]	Molecule A destroys molecule B at a rate of $10^6\text{M}^{-1}\cdot\text{s}^{-1}$.
A + B -> A + C [1e6]	Molecule A catalytically converts B to C at a rate of $10^6\text{M}^{-1}\cdot\text{s}^{-1}$.
A+B -> A+B+C [1e6]	Collision of A and B catalytically generates C at a rate of $10^6\text{M}^{-1}\cdot\text{s}^{-1}$.

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 have an orientation: up if they point in the direction of the surface normal, down otherwise.. Molecules in 3D 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 10s^{-1} .
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 10s^{-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 $10^5\text{M}^{-1}\cdot\text{s}^{-1}$.
A, + B, -> C, [1e5]	The same reaction again—everything occurs on the same side, but we wrote it on the bottom this time.

Example	Explanation
$A' + B, \rightarrow C' [1e5]$	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 $10^5 \text{ M}^{-1} \cdot \text{s}^{-1}$.
$A, + B' \rightarrow C, [1e5]$	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. Therefore:

Example	Explanation
$A'' + B, \rightarrow C' [1e5]$	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 $10^5 \text{ M}^{-1} \cdot \text{s}^{-1}$.
$A,, + B, \rightarrow C' [1e5]$	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.
$A,, + B' \rightarrow C, [1e5]$	Same again—B and C still have opposite orientations.
$A, + B' \rightarrow C,, [1e5]$	Molecule A hits the opposite side of B and produces C that is equally likely to point either way, at a rate of $10^5 \text{ M}^{-1} \cdot \text{s}^{-1}$.
$A, + B' \rightarrow C'' [1e5]$	Same as above, since C is still not in the same orientation class as the others.
$A'+B'' \rightarrow A,+B''' [1e5]$	Molecule A hits molecule B on either side; A keeps traveling (goes to the other side) and B tumbles to a random orientation, at a rate of $10^5 \text{ M}^{-1} \cdot \text{s}^{-1}$.
$A'+B'' \rightarrow C'''+D'''' [1e5]$	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.

Need to add code and syntax to specify a surface with an orientation also.

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
<pre> DEFINE_SURFACE_CLASS <i>name</i> { <i>surface property commands</i> } </pre>	Define a single surface type called <i>name</i> . The properties are specified by commands inside braces.

Command	Explanation
<pre> DEFINE_SURFACE_CLASSES { nameA { surface property commands } nameB { surface property commands } ... } </pre>	Define a series of surface types by name.

To define surface properties, use the following commands:

Surface Property Command	Explanation
REFLECTIVE = <i>name</i>	The molecule called <i>name</i> is reflected by this surface. This is the default behavior.
TRANSPARENT = <i>name</i>	The molecule called <i>name</i> passes through this surface.
ABSORPTIVE = <i>name</i>	The molecule called <i>name</i> is destroyed if it touches this surface.
<pre> MOLECULE_DENSITY { name1 = D1 name2 = D2 ... } </pre>	Add the named molecules at the specified densities <i>D1</i> , <i>D2</i> , ..., (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.
<pre> MOLECULE_NUMBER { name1 = N1 name2 = N2 ... } </pre>	Add the exact numbers <i>N1</i> , <i>N2</i> , ..., 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
<pre> name BOX { box commands region commands transformation commands } </pre>	This defines a box object called <i>name</i> . 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.
<pre> name POLYGON_LIST { polygon commands region commands transformation commands } </pre>	This defines a polygon list object called <i>name</i> . 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.
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 { <i>element specifier commands</i> <i>regional surface commands</i> } 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 { <i>element specifier commands</i> }	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 = [<i>list</i>]	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 <i>N TO M</i> . The two styles can be mixed (separated by commas).
EXCLUDE_ELEMENTS = [<i>list</i>]	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 = <i>name</i>	Include the existing region on this object called <i>name</i> into this region, too.
EXCLUDE_REGION = <i>name</i>	Exclude the existing region on this object called <i>name</i> from this new region.
INCLUDE_PATCH=[<i>x1</i> , <i>y1</i> , <i>z1</i>] , [<i>x2</i> , <i>y2</i> , <i>z2</i>]	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=[<i>x1</i> , <i>y1</i> , <i>z1</i>] , [<i>x2</i> , <i>y2</i> , <i>z2</i>]	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 = <i>name</i>	Set the surface type of this region to the previously defined surface class called <i>name</i> .
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.

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
<i>name</i> RELEASE_SITE { <i>release site commands</i> <i>transformation commands</i> }	Create a release site called <i>name</i> . The shape and method of release is specified by the release site commands. Optionally, geometrical transformations can be applied also.
<i>name</i> CUBIC_RELEASE_SITE { ... }	Create a cubic release site called <i>name</i> . Molecules are released in a box as specified by the radius. (This is the same as using the SHAPE=CUBIC command inside RELEASE_SITE.)
<i>name</i> SPHERICAL_RELEASE_SITE { ... }	Create a spherical release site called <i>name</i> . 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.)

Command	Explanation
<i>name</i> SPHERICAL_SHELL_SITE {...}	Create a spherical shell release site called <i>name</i> . 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 <i>name</i> { <i>release pattern commands</i> }	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 = <i>geometry</i>	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. If the molecule has a 3D diffusion constant, it will be released in the volume bounded by the regions and each region must be closed; otherwise, it will be released on the surface.
LOCATION = [<i>x</i> , <i>y</i> , <i>z</i>]	The release occurs centered at this location. Only used for geometrical shapes.
MOLECULE = <i>name</i>	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 { <i>name1</i> [<i>x1</i> , <i>y1</i> , <i>z1</i>] <i>name2</i> [<i>x2</i> , <i>y2</i> , <i>z2</i>] ... }	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 = <i>d</i>	For a geometrical release site, this releases molecules uniformly within a diameter <i>d</i> . 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 = [<i>x</i> , <i>y</i> , <i>z</i>]	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 = <i>p</i>	This release does not occur every time, but rather with probability <i>p</i> . (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.
NUMBER_TO_RELEASE = <i>n</i>	Release <i>n</i> molecules. For releases on regions, <i>n</i> 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 <i>n</i> 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.

Release Site Command	Explanation
CONCENTRATION = c	Release molecules at concentration c molar for volumes and c molecules per square micron for surfaces. Not used for the LIST shape.
GAUSSIAN_RELEASE_NUMBER { MEAN_NUMBER = n STANDARD_DEVIATION = s }	Release molecules according to a Gaussian distribution with the specified mean and standard deviation.
RELEASE_PATTERN = <i>name</i>	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.

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 of objects

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

Command	Explanation
<i>name</i> OBJECT { <i>object specifier commands</i> <i>transformation commands</i> }	Define a new object called <i>name</i> . Inside the braces, list other objects one at a time to be added (see below).
INstantiate <i>name</i> OBJECT { ... }	Same as above, except we also insert the object into the world. A simulation must have at least one INSTANTIATED object.

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

Object Specifier Command	Explanation
<i>newname</i> OBJECT <i>oldname</i> { <i>transformation commands</i> }	Add the existing object called <i>oldname</i> into the existing object and label it <i>newname</i> . 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 <i>name.newname</i> .
<i>name</i> BOX { ... }	Create a box inside the existing object (using the same syntax as previously defined).
<i>name</i> POLYGON_LIST { ... }	Create a polygon list object inside the existing object (using the same syntax as previously defined).

Object Specifier Command	Explanation
<i>name</i> RELEASE_SITE { ... }	Create a release site inside the existing object.
<i>newname</i> 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 = [<i>x</i> , <i>y</i> , <i>z</i>]	Move the object by the specified vector.
SCALE = [<i>x</i> , <i>y</i> , <i>z</i>]	Scale the object by multiplying each coordinate by the corresponding value in the vector.
ROTATE = [<i>x</i> , <i>y</i> , <i>z</i>] , <i>A</i>	Rotate <i>A</i> 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.

Command	Explanation
VIZ_DATA_OUTPUT { ... }	Define a new visualization data output block which contains the commands that are not yet described in this document. Each mdl file can have multiple visualization blocks, but they must all have the same mode.
REACTION_DATA_OUTPUT { <i>reaction output commands</i> }	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 = <i>N</i>	Write output to disk after every <i>N</i> lines. The default is <i>N</i> =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 <i>N</i> .
STEP = <i>t</i>	Output this block every <i>t</i> seconds. Exactly one of STEP or the following two commands should be used.
TIME_LIST = [<i>list</i>]	Output this block at the times specified in the list.
ITERATION_LIST = [<i>list</i>]	Output this block at the iteration numbers specified in the list (i.e. after that number of timesteps).
{ <i>value</i> } => " <i>file</i> "	Output the value in braces to the filename in quotes. The first column will be the timestep; the second will be the value listed. This command can be repeated to send different output to many files.
{ <i>value</i> , <i>value</i> , ... } => " <i>file</i> "	Output the list of values in braces, one to a column, in the order listed. The first column will be the timestep. This command can be repeated.

The values specified in braces are 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). The count statements themselves have the following syntax:

Count Statement	Explanation
COUNT[<i>name</i> , WORLD]	Count things in the world. If <i>name</i> refers to a molecule, count how many of that molecule are in the world. If it refers to a reaction, count how many times that reaction has occurred so far.
COUNT[<i>name</i> , <i>object</i>]	Count things inside the object called <i>object</i> . This must be an instantiated object. For example, if you have instantiated an object called <code>my_world</code> with a box called <code>my_box</code> inside it, <i>object</i> would be <code>my_world.my_box</code> . If you are counting surface molecules or reactions at a surface, only the ones that actually occur on <i>object</i> will be counted. Molecules with a 3D diffusion constant will be counted inside the object, but the object must be closed.
COUNT[<i>name</i> , <i>region</i>]	Count things inside the named region. It must be referenced fully. E.g. if <code>my_box</code> (from above) has a region called <code>my_region</code> , the name would be <code>my_world.my_box[my_region]</code>
COUNT[<i>name</i> , <i>region</i> , ALL_ENCLOSED]	Count all molecules or reactions that occur inside this region (not counting those that occur on the surface of the region). This lets you count surface molecules contained within a box, for example. This will work with objects as well as regions, but the object or region must be closed.
COUNT[<i>name</i> , <i>region</i> , CONCENTRATION]	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, but multiple objects cannot be used. 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 i th 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.
COUNT[<i>name</i> , <i>region</i> , <i>hits</i>]	Output the number of times the named molecule has hit the named region (or object). The <i>hits</i> specifier should be one of FRONT_HITS, BACK_HITS, ALL_HITS, FRONT_CROSSINGS, BACK_CROSSINGS, and ALL_CROSSINGS.

3.6 Utility commands

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

Numerical Command	Explanation
SQRT(<i>x</i>)	Return the square root of <i>x</i>
EXP(<i>x</i>)	Return the value of <i>e</i> raised to the x^{th} power
LOG(<i>x</i>)	Return the natural logarithm of <i>x</i>
LOG10(<i>x</i>)	Return the base 10 logarithm of <i>x</i>
SIN(<i>x</i>)	Return the sine of <i>x</i>
COS(<i>x</i>)	Return the cosine of <i>x</i>
TAN(<i>x</i>)	Return the tangent of <i>x</i>
ASIN(<i>x</i>)	Return the inverse sine of <i>x</i>
ACOS(<i>x</i>)	Return the inverse cosine of <i>x</i>
ATAN(<i>x</i>)	Return the inverse tangent of <i>x</i>
ABS(<i>x</i>)	Return the absolute value of <i>x</i>

Numerical Command	Explanation
CEIL(<i>x</i>)	Return the smallest integer at least as big as <i>x</i>
FLOOR(<i>x</i>)	Return the largest integer at no bigger than <i>x</i>
MAX(<i>x</i> , <i>y</i>)	Return the larger of <i>x</i> and <i>y</i>
MIN(<i>x</i> , <i>y</i>)	Return the smaller of <i>x</i> and <i>y</i>
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 `*/`.

`//`Include file syntax should be described here, but it is not!!

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 $6\mu\text{m}$ long, partitions may vary by about $0.0003\mu\text{m}$. 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 3D diffusing molecule hits a surface, or if the probability goes above 0.3 for a collision between pairs of 3D diffusing molecules.

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.

//Need to describe the messages that MCell3 spits out when it detects an error with probabilities (at the end of a run)!!

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 and partition boundaries. In general, 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 $0.02\mu\text{m}$ apart, simulation accuracy will be poor with the default interaction radius of $0.01\mu\text{m}$. 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 reversible 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"
}

```

!!That's all, folks!!