Cellerator™ Command Reference

Copyright © 2005 California Institute of Technology.
All Rights Reserved. U.S. Government sponsorship acknowledged.

This listing was generated by createCelleratorReference
January 13, 2005 14:40:12 using Cellerator Version 1.5.0 (03-Jan-2005) in Mathematica 5.1 for Mac OS X (October 25, 2004)

addLeaf

addLeaf[tree, options] adds a leaf at the selected address in a tree. The selected address is split into two branches; whatever was previously at that leaf (which may be a subtree) is put on the left branch and the new data is put on the right branch.
Options are
address -> hash code into the tree (string of 1's and 2's for for left & right) specifying the location of the new node; for example if address-> {2,3,2} then node {2,3,2} will be split into two nodes. Whatever is at {2,3,2} becomes {2,3,2,1} and the new node becomes {2,3,2,2}.If address is an empty list {}, then the root node is split into two branches, with the old tree put on the left branch and the new leaf on the right.
data-> the data to be added to the new leaf. Default is an empty list {}

addLink

addLink[g,n1,n2] returns a graph with a new directed link between node n1 and n2 (integers); because the link is directed, only the spring potential and embedding odes for node 1 are changed and not for node 2. Thus node 2 exerts a force on node 1 and not vice-versa. See also addLinkBothways, addLinksBothWays.

addLinkBothWays

addLinkbothWays[g,n1,n2] adds two links: one from n1 to n2, and a second from n2 to n1. Both links are identical. See also addLink, addLinksBothways.

addLinks

addLinks[g,{{n1,n2},{n1,n2},...}}] returns a new graph with the specified links added. If any links that are requested already exist, those links are ignored. The spring force is NOT updated.

addLinksBothWays

addLinksBothWays[g, {{n1,m1},{n2,m2},{n3,m3},...}] adds two-way links between each of the specified node pairs. See also addLinkBothWays, addLinks (addLinks is the preferred method because it is more efficient thatn addLinksbothWays).

addNode

addNode[g,n] returns a graph with the node g added to it
addNode[g,n,parent->k] modifies the lineage of the graph to indicate the specified parent.

addSpecies

addSpecies[x1,x2,...] adds the species x1, x2,... to the global list $reactionSpecies, which is used during construction of SBML level 1 to save the names of the species that need to be defined.

andedString

andedString[x1,x2,...] returns the string "(x1&&x2&&...)"

andedStringMultisperseBefore

andedStringMultisperseBefore[{s1,s2,...,sn},{{ta1,ta2,...,tan},{tb1,tb2,..., tbn},...,{tz1,tz2,...,tzn}}] returns a list of strings, {"(s1ta1)&&(s2ta2)&&...&& (sntan)","(s1tb1)&&(s2tb2)&&..&&(sntbn)",...,"(s1tz1)&&(s2tz2)&&...&&(sntzn)"}.
For example, andedStringMultisperseBefore[{A,B,C},{{1,2,3},{4,5,6},{7,8, 9}}] returns {"(A1)&&(B2)&&(C3)","(A4)&&(B5)&&(C6)","(A7)&&(B8)&&(C9)"}

applyIndex

applyIndex[*var, index*] returns an indexed variable, *var*[*index*]; if *var* is already indexed, additional indices are added, e.g., applyIndex[x[5],{1,2}] returns x[5,1 2]

applySpringForce

applySpringForce[g,options] calculates the spring-force ODEs for every node in the graph g and regurns a new graph that applies those ODEs to the embedding variables.

Options are:
exception→"DeadCell": string name of nodeType that is not to have spring-force applied to it; e.g., for toy meristem, the string "Stem" might be used
fastForce→True used fastSpringForce, otherwise use springForce

argument

argument[f[x]] returns x
argument[f[x1,x2,...] returns {x1,x2,...}
argument[x] returns x when x is an atom.

arrow

arrow[type, arguments] is a text-command alternative to a cellerator arrow. All cellerator arrows can be specified in this way. In this way external programs that do not have access to Mathematica character sets can generate Cellerator command sequences without knowledge of the Mathematica escape sequences for Mathematica syntax represented by the Cellerator arrow.
The following formats are allowed (the argument k is optional and may contain more than argument, in the same order as the rate constants are normally specified in the call to interpret). Option sequences are of the form parameter->value, parameter->value,..., where "->" is the standard keyboard dash-greatethen.
arrow["Allosteric",...] is eqivalent to arrow["NHCA",...].
arrow["Annihilate",x,k] or arrow["Annihilation",x,k] are equivalent to {x→∅,k}
arrow["Bidirectional",x,y,k] and arrow["Reversible",x,y,k] are equivalent to {x⇄y, k}

arrow["BidirectionalEnzymatic",...] is equivalent to arrow["ReversibleCatalytic"...]

arrow["Catalytic",source,product,enzyme,k] is equivalent to {source $\overset{\text{enzyme}}{\rightleftarrows}$ product,k}
arrow["Conversion",x,y,k] or arrow["Reaction",x,y,k are equivalent to {x→y, k}
arrow["Create",x,k] or arrow["Creation",x,k] are equivalent to {∅→x,k}
arrow["Enzymatic",...] is equivalent to arrow["catalytic",...]

arrow[GMWC,src, prod, act, inh, enz, options] is equivalent to src $\underset{\{act,inh\}}{\overset{enz}{\Longrightarrow}}$ prod

arrow["GRN",x,y,options] is eqivalent to {x↦y, GRN[options]}; this is a transcriptional model.
arrow["Hill",x,y,options] or arrow["HillFunction",x,y,options] are eqivalent to {x↦y, hill[options]}; this is the transcriptional version of the Hill function
arrow["Hill",x,y,z, options], arrow["HillFunction",x,y,z, options], and arrow["SimplifiedSaturated",x,y,z, options] are eqivalent to {x $\overset{z}{\mapsto}$ y, hill[options]}; this is the saturating reaction version of the Hill function
arrow["MM",x,y,KD,V] is equivalent to {x⟹y, MM[KD,V]}
arrow["MM",x,y,enz,KD,V] is equivalent to {x $\overset{enz}{\Longrightarrow}$ y, MM[KD,V]}

arrow["MM",x,y,enz,k1,k2,k3] is equivalent to {x $\overset{enz}{\Longrightarrow}$ y, MM[k1,k2,k3]}
arrow["NHCA",x,y,options] is eqivalent to {x↦y, NHCA[options]}; this is non-heirarchical cooperative activation model that is based on the allosteric MWC model.
arrow["Reaction",x,y,k] or arrow["Conversion",x,y,k] are equivalent to {x→y, k}
arrow["Reversible",x,y,k] and arrow["Bidirectional",x,y,k] are equivalent to {x⇄y, k}
arrow["ReversibleCatalytic",source, product, forwardEnzyme,reverseEnzyme,k] is equivalent to source⇄product with overscript forwardEnzyme and underScript reverseEnzyme
arrow["ReversibleEnzymatic",...] is equivalent to arrow["ReversibleCatalytic"...]

arrow["SimpleCatalytic",x,y,enz,k] is equivalent to x $\overset{enz}{\rightarrow}$ y
arrow["SimplifiedSaturated",x,y,z, options] is the same as arrow["Hill", x,y,z, options] (catalytic, not transcriptional, Hill function)
arrow["SimplifiedNonSaturated",x,y,enz,k] is equivalent to arrow["SimpleCatalytic",x,y,enz, k] arrow["SSystem",x,y,tau,kp,km,cp,cm] is equivalent to {x↦y,SSystem[tau,kp,km,cp,cm]}
See also arrows for a list of the available arrows.

arrows

Arrows. Cellerator uses the following arrows:
→ or \[ShortRightArrow]
↦ or \[RightTeeArrow]
⇄ or \[RightArrowLeftArrow]
⟹ or \[DoubleLongRightArrow] or ESC==>ESC
⇌ or \[Equilibrium] or ESCequiESC (note: Equilibrium is just
the name used by Mathematica; this arrow does NOT denote Equilibrium!)
In addition, over- and underscripts are used to indicate catalytic (enzymatic) reactions.
The under/overscripts are over/under the entire reaction, not just the arrow. Arrows should
be entered using the Cellerator Palette to ensure the correct syntax.See also arrow.
The arrow used for reactions (→) is not the same symbol as the arrow used for option lists (→).
The arrow used for option lists can be entered using the keyboard -> character combination.
The following classes of cellerator arrows are defined:
S→P Conversion of S to P
S→∅ Annihilation of S
∅→P Creation of P
S⇄P Reversible conversion between S and P. Biochemical equivalent: {S→P, P→S}
$S \overset{X}{\rightleftarrows} P$ Catalytic reaction, coversion of S to P is faciliated by X with
the formation of an intermediate compound. Biochemical equivalent: S+X⇄SX→X+P
$S \overset{X}{\rightleftharpoons} P$ Catalytic reaction with two intermediate complexes. Biochemical equivalent is S+X⇄SX⇄PX⇄X+P
$S \underset{Y}{\overset{X}{\rightleftarrows}} P$ Reversible catalytic reaction, the conversion of S to P is facilated
by X and the conversion of P to S is facilitated by Y. Typically X is a kinase
and Y is a phosphatase. Biochemical equivalent:{S+X⇄SX→X+P, P+Y⇄PY→S+Y}
$S \overset{x}{\rightarrow} P$ Catalytic reaction without formation of intermediate
compout. Biochemical equivalent: S+X→P+X
$S \overset{X}{\mapsto} P$ the conversion of S to P is saturable in S and proceeds at a rate that
is proportional to X as $\frac{dP}{dt} = \frac{S^n\ (r + v\ X)}{K^n + S^n}$ where n, r, v, and K are constants.
S↦P Production of P occurs at a rate that depends upon S according
to one of the Right-Tee-Arrow functions:hill, GRN, NHCA, SSystem.
{S⟹P, MM[KD,V} Michaelis-Menten with $\frac{dP}{dt} = \frac{V * S}{KD + S}$
$\{S \overset{enz}{\Longrightarrow} P$, MM[KD,V} Michaelis-Menten with $\frac{dP}{dt} = \frac{V * S * enz}{KD + S}$
$\{S \overset{enz}{\Longrightarrow} P$, MM[k1,k2,k3} Michaelis-Menten with $\frac{dP}{dt} = \frac{k3 * S * enz}{\frac{k2+k3}{k1} + S}$
{S↦P,hill[...]} $\frac{dP}{dt} = k + \frac{r + v\ S^n}{K^n + S^n}$, where k, K, r, v, n are constants;
{S↦P,GRN[...]} $\frac{dP}{dt} = \frac{r}{1 + e^{-h - T\ S^n}}$, where h,T, n, r are constants(GRN);
{S↦P,NHCA[...]} $\frac{dP}{dt} = \frac{(1 + T\ S^n)^m}{k\ (1 + R\ S^n)^m + (1 + T\ S^n)^m}$, where R,
T, k, n, m are constants.(NHCA, non-heirarchical cooperative activation))
{S↦P,SSystem[...]} $\frac{dP}{dt} = \frac{1}{\tau}\left(k^+ \prod_i S_i^{c_i^+} - k^- \prod_i S_i^{c_i^-}\right)$ (SSystem)
$\{S1, S2, \ldots\} \overset{Enzyme}{\underset{\{\{A1,A2,\ldots\},\{I1,I2,\ldots\}\}}{\Longrightarrow}} \{P1, P2, \ldots\}$ $\frac{dP}{dt} = Enzyme * kcatGMWC * \left(\left(\Pi\ \frac{Si}{KSi}\right)\left(\Pi\left(1 + \frac{Si}{KSi}\right)^{n-1}\right)\left(\Pi\left(1 + \frac{Ai}{KAi}\right)^n\right) + L * \left(\Pi\ c * \frac{Si}{KSi}\right)\left(\Pi\left(1 + c * \frac{Si}{KSi}\right)^{n-1}\right)\left(\Pi\left(1 + \frac{Ii}{KIi}\right)^n\right)\right) / \left(\left(\Pi\left(1 + \frac{Si}{KSi}\right)^n\right)\left(\Pi\left(1 + \frac{Ai}{KAi}\right)^n\right) + L * \left(\Pi\left(1 + c * \frac{Si}{KSi}\right)^n\right)\left(\Pi\left(1 + \frac{Ii}{KIi}\right)^n\right)\right)$ where c, L, v, n are constants (GMWC, Generalized MWC).
$\underset{\{\{A1,A2,\ldots\},\{I1,I2,\ldots\},\{\{JS11,JS12,\ldots\},\{JS21,JS22,\ldots\},\ldots,\{JA11,JA12,\ldots\},\{JA21,\ldots\},\ldots\}\}}{\{S1, S2, \ldots\} \overset{Enzyme}{\Longrightarrow} \{P1, P2, \ldots\}}$ $\frac{dP}{dt} = Enzyme *$

$$\text{kcatGMWC}*\left(\left(\Pi\ \frac{Si}{KSi}\right)\left(\Pi\left(1+\frac{Si}{KSi}+\Sigma_p\ \frac{JS_{ip}}{KJS_{ip}}\right)^{n-1}\right)\left(\Pi\left(1+\frac{Ai}{KAi}+\Sigma_j\ \frac{J_{ij}}{KJ_{ij}}\right)^{n}\right)+L*\left(\Pi\ c*\frac{Si}{KSi}\right)\left(\Pi\left(1+c*\frac{Si}{KSi}+\Sigma_p\ \frac{JS_{ip}}{KJS_{ip}}\right)^{n-1}\right)\left(\Pi\left(1+\frac{Ii}{KIi}\right)^{n}\right)\right)/\left(\left(\Pi\left(1+\frac{Si}{KSi}+\Sigma_p\ \frac{JS_{ip}}{KJS_{ip}}\right)^{n}\right)\left(\Pi\left(1+\frac{Ai}{KAi}+\Sigma_j\ \frac{J_{ij}}{KJ_{ij}}\right)^{n}\right)+L*\left(\Pi\left(1+c*\frac{Si}{KSi}+\Sigma_p\ \frac{JS_{ip}}{KJS_{ip}}\right)^{n}\right)\left(\Pi\left(1+\frac{Ii}{KIi}\right)^{n}\right)\right)\text{(GMWC with competitive inhibition)}$$

See also arrow[...] for text equivalents to all arrows.

arrowType

arrowType[{reaction,rates}] returns an integer indicating the type of Cellerator Arrow
arrowType[reaction] will also return the same integer Except for transcriptional right-t arrows

See also interpretArrowType for a String instead of an integer.

The return values are:

1	$\emptyset\rightarrow x$ or $\{\emptyset\rightarrow x\}$ or $\{\emptyset\rightarrow x,k\}$
2	$x\rightarrow\emptyset$ or $\{x\rightarrow\emptyset\}$ or $\{x\rightarrow\emptyset,\ k\}$
3	$x\rightarrow y$ or $\{x\rightarrow y\}$ or $\{x\rightarrow y,\ k\}$
4	$x\rightleftarrows y$ or $\{x\rightleftarrows y\}$ or $\{x\rightleftarrows y,\ k,...\}$
5	$x\overset{z}{\rightarrow} y$ or $\{x\overset{z}{\rightarrow} y,\ k\}$
6	$x\overset{fwd}{\rightleftarrows} y$ or $\{x\overset{fwd}{\rightleftarrows} y,\ k,...\}$
7	$x\underset{rev}{\overset{fwd}{\rightleftarrows}} y$ or $\{x\underset{rev}{\overset{fwd}{\rightleftarrows}} y,\ k,...\}$
8	$x\overset{z}{\mapsto} y$ or $\{x\overset{z}{\mapsto} y,\ ...\}$
9	$x\underset{b}{\overset{a}{\Longrightarrow}} y$ or $\{x\underset{b}{\overset{a}{\Longrightarrow}} y,\ k,\ ...\}$
10	$\{x\mapsto y,\text{hill[stuff]}\}$ or $\{x\mapsto y,\text{type}\rightarrow\text{hill},\ ...\}$
11	$\{x\mapsto y,\text{GRN[stuff]}\}$ or $\{x\mapsto y,\text{type}\rightarrow\text{GRN},\ ...\}$
12	$\{x\mapsto y,\text{NHCA[stuff]}$ or $\{x\mapsto y,\text{type}\rightarrow\text{NHCA},\ ...\}\}$
13	$\{x\mapsto y,\text{SSystem[stuff]}\}$
14	$\{x\Longrightarrow y,\ \text{MM[KD,V]}\}$
15	$\{x\overset{z}{\Longrightarrow} y,\ \text{MM[K,V]}\}$ or $\{x\overset{z}{\Longrightarrow} y,\ \text{MM[k1,k2,3]}\}$
16	$\{x\overset{z}{\rightleftharpoons} y,\ k,...\}$
-1	anything else

assignToCompartments

assignToCompartments[var1[i1], var2[i2],var3[i3],...] returns a list {{var1, {i1,i2,...}}, {var2,{i1,i2,...}},...} where the list following each variable gives the numbers of the compartments (cells) that each variable is defined for.

autoRate

autoRate[options] returns a new rate constant of the form <prefix>nnn<suffix>, where nnn
  is an integer incremented by one each time autoRate is called, as determined by the
  following options. If a constant so generated has been previously defined, then nnn
  is incremented until a new one can be defined, unless reset is True. Options are::
digits->3, number of digits that the string nnn should be padded to
prefix→"k"
reset->False, if True nnn is reset to 1
suffix→"" (default is null string)
verbose->False, if True print message if anything out-of-the ordinary
  occurs, e.g., a previously defined constant is reused or a number is skipped.

binaryTreeQ

binaryTreeQ[x] returns True if x is a binary
  tree, and False when x is a non-binary tree or when x is not a tree.

blocks

blocks[options] returns True if a line segment defined by two points intersects a circle. The
  intersection must be between the two points, and not on the extended line. Options are:
center→{0,0}
radius→1
endPoints→{{x1,y1},{x2,y2}}
plot->False
See also: intersects, findIntersection

bracketedList

bracketedList[x1,x2,...] returns a string "[x1][x2]..."; the x's may be
  any heirarchy of lists and or atoms, includign the null list, which are all
  compressed to form a single sequence of brackets in the order presented, e.g.,
  bracketedList[{{a,b},c,{d,{e,f}}}] will return "[a][b][c][d][e][f]". Any null
  strings will be ignored, e.g., bracketedList[a,{},c] returns the string "[a][c]".

breaklinks

breaklinks is an option for run[graph,...] that determines if link breakage will be
  allowed. The default value is True. If breaklinks->True (Default) links will be
  removed when the distance between two nodes dynamically surpasses the value of dmax
  in an increasing direction. If breaklinks->False, no checking will be performed.

breakPoints

breakPoints[s,options] returns a list of the (integer) offsets in the string where line-wrapping is allowed, which are after any of the following characters: *,()[]+-/^>=&&||!?:.
Options are:
wrapAfter->{ch1,ch2,...} list of characters after which breaks are allowed; overrides defaults.
wrapAlso->{ch1, ch2,...} list of characters after which breaks are allowed; these characters are added to the allowed list
wrapExcept->{ch1,ch2,...} list of characters that are to be removed from the default character list
wrapAll-> False, if True, allow wrapping at all characters.

cellDivisionDomain

cellDivisionDomain[*options*] returns a modelDomain with two variables *x* and *y* such that *x*'==Θ(*A*-*T*) and *y*'=Θ(*x*(*t*)), where *A* is a molecule in another domain of the same cell and *T* is a constant.

Options are:
variables→{*x, y*} : gives the names of *x, y* (Defaults: splitVariable, timeSinceDivision}
cellDivisionvariable→A : gives the name of the molecule A (Default is "molecule2"
cellDivisionThreshold→T : value of *T* (Default is 0.6)
massVariable->mass: name of variable in domain that represents mass
step→Unit, type of step function to use on model;
default is Unit (use Mathematica UnitStep); step→Fuzzy (use fuzzyStep[x])

Cellerator

Cellerator™ is a Mathematica package designed to facilitate biological modeling via automated equation generation. Cellerator was designed with the intent of simulating at least the following essential biological processes: (1)signal transduction networks (STNs); (2) cells that are represented by interacting signal transduction networks; and (3) multi-cellular tissues that are represented by interacting networks of cells that may themselves contain internal STNs. These processes combine to form an obvious hierarchy that can be further subdivided for notational simplicity (e.g., STNs as elements of STNs, and so forth). In the past it has been necessary to manually translate chemical networks from cartoon-diagrams to chemical equations and thence to ordinary differential equations. This process is tedious and highly error prone, and impractical for all but the simplest of systems because of the combinatoric increase in the number of equations with the number of chemical species. Cellerator™ provides a framework for generating, translating, and numerically solving a potentially unlimited number of biochemical interactions.This is Cellerator ™ Version 1.5.0 (03-Jan-2005). To load Cellerator, include the statement
<<directory/cellerator.m
at the top of your Mathematica notebook.
©2001-2005 California Institute of Technology. U.S. Government Sponsorhip Acknowledged. All rights reserved. Patent Pending. Cellerator™ is not public domain software.
For more details see the Cellerator™ web site at http://www.cellerator.info

celleratorFileName

celleratorFileName[*name*, *type*, options] returns a string of the
formn"*directory*/*name*yyyymmddThhmm.*type*" which where the current date and time
are mm/dd/yyyy at hh:mm. *Type* defaults to 'dat'. *Name* defaults to 'CellFile'.

Options are:
directory→ directory name (defaults to the global varialbe $CelleratorOutputDirectory)

celleratorGraph

celleratorGraph[n] is the name of the global data structure to store a graph
referenced as pointer["celleratorGraph[n]"]; the number of graphs that have
been defined is given by graphCounter; to clear all graphs use ClearGraphs[].

celleratorGraphSolutionQ

celleratorGraphSolutionQ[s] returns True if s appears to be the output
of run[graph,...] and False otherwise. Compare with celleratorRunSolutionQ

celleratorODE

celleratorODE[n] is the name of the global data structure to store a differential
equation referenced as pointer["celleratorODE[n]"]; the number of ODEs that
have been defined is given by odeCounter; to clear all graphs use ClearODEs[].

celleratorRunSolutionQ

celleratorRunSolutionQ[s] returns True if s appears to be the output
of a run[STN,...] and False otherwise. Compare with celleratorGraphSolutionQ

celleratorSolution

celleratorSolution[n] is the name of the global data structure to
store a numerical solution (a list of interpolating functions) referenced
as pointer["celleratorSolution[n]"]; the number of Solutionss that have been
defined is given by solutionCounter; to clear all solutions use clearSolutions[].

celleratorSmassSgrowthSformula

celleratorSmassSgrowthSformula[m] is an uninstantiated function
that is replaced at run time with the selected formula for mass growth.

check4Event

check4Event[g,solution, tstart, tend, tsplit, dmax,options] determines what event should end the integration step.It is not intended for user execution, but is called by other functions.
Options:
mitosis->True (check for mitotic events)
breaklinks->True (check to see if links should be broken).
Return value: the list {*event-data, event-Time, event-description*}
event-data is the list if nodes if *event-description* is 'cell-divides'
event-data is the list of links that break if *event-description* is 'link-broken'
event-data is {} if event-descrition is 'normal-exit' (i.e., no event occurred)'
event-Time is the time of the first event that occurs.
Parameters:
g: name of graph;
solution: name of interpolating function
tstart, tend: time ranges to use in solution
tsplit: name of split variable
dmax: link length trigger for breaking.

checkDistance

checkDistance[l, tstart, tend, dmax] determines if link number l has crossed dmax in length between tstart and tend in the global set of interpolating functions current$solution. Not intended for end-user execution.

checkIfSplit

checkIfSplit[g, solution, tsplit, tend] determines if any variables in the graph g meet the cell division criterion between the beginning of the interpolation function split and the time tend, and if so, deterimines which node (or nodes) split first (its possible for multiple splits to occur simultaneously). The list {{n1, n2,n3,...}, t} is returned, where n1, n2, n3,.. are the numbers of the nodes that split at time t. The variable tsplit is the name of the split variable to check (the indexed variable that grows linearly with time after split conditions are met). .

circularTable

flatTable[r] generates a list of coordinates for cell centers for a two-dimensional hemispherical meristem, where r is the radius is cell diameters of the meristem. The cell centers are aligned along a hexagonal grid with spacing of one unit. The coordinates are embedded in a three-dimensional coordinate system, but all cell centers lie in the x-y plane, i.e., the third coordinate is always zero.

cloneNode

cloneNode[*n*,*i*] returns a new node copied from *n* but with the variable index replaced by *i*.
cloneNode[*n*,*i*,{*xoffset, yoffset, zoffset*}] moves the initial conditions of the new node by a vector {*xoffset, yoffset, zoffset*}

combineHillReactionsByProduct

```
combineHillReactionsByProduct[reaction,type] is called
  by prepReactionsForSBML, and is not meant to be called by the end user.
It modifies a list of reactions of the form {{A1↦B1,type[opts]},{A2↦B2,type[opts]},...}
  by combining all reactions with the same product into a single reaction of the form {A1⊕
  A2⊕A3...↦B,type[opts]}, and validates the combined option list within type[opts]. Every
  reaction passed to this function must have the same type. Valid types are hill, GRN, nHCA.
For example,combineHillReactionsByProduct[
    {{A↦B,hill[vmax→ v1,nhill→ n1,khalf→ K1, basalRate→ r1]},
    {C↦B,hill[vmax→ v2,nhill→ n2,khalf→ K2, basalRate→ r2]},
    {A↦C,hill[vmax→ v3,nhill→ n3,khalf→ K3, basalRate→ r3]}},hill]
returns
{{A⊕C↦B,hill[vmax→{v1,v2},nhill→n1,khalf→K1,basalRate→r1]},
    {A↦C,hill[vmax→{v3},nhill→n3,khalf→K3,basalRate→r3]}}
```

comma

```
comma is a global symbol used during Cellerator
  output translation to hold the desired text symbol used for a comma.
```

commaDelimitedString

```
commaDelimitedString[x1,x2,...] returns the string"x1,x2,...".
```

commaSeparatedString

```
commaSeparatedString[x1,x2,...] returns a
  string "x1▷comman▷x2▷commaь..."; see also commaDelimitedString
```

Comp

```
Comp[x] indicates the presence of an intermediate reactant,say xprime, that is not
  explicitly specified because of the implied conservation law that xprime[t]=1-x[t]]
Comp[x,xtotal] indicates that xprime[t]=total-x[t]

example: interpret[{{Comp[q]↦q,hill[vmax→ v,khalf→ K]}}] is interpreted as
{{q′[t]==v (1 - q[t]) / (1 + K - q[t])},{q}}
```

compartment

```
compartment is an option for writeSBML that specifies the name to used for the SBML compartment
  that represents a Cellerator graph node.If unspecified a unique name is generated automatically.
```

complex

```
The symbols complexLeft and complexRight are
  used to automatically generated new symbol names in the form
    <complexLeft>s1<dash>s2<complexRight>
See also complexLeft, complexRight, compoundName, intermediateCompound
```

complexLeft

```
complexLeft (default value=) is a globally defined string that is
  used to generate the name of an intermediateCompound. See intermediateCompound.
```

complexRight

```
complexRight (default value=) is a globally defined string that is
  used to generate the name of an intermediateCompound. See intermediateCompound.
```

compoundName

```
compoundName[x,y] returns a string representing the name of an intermediate
  compound formed by joining x and y. The form of the string returned is
    <complexLeft><x><dash><y><complexRight>
where <complexLeft>,<dash>, and<complexRight> are the values of the
  three strings by those names; and <x> and <y> are the strings for x and y. For
  example, if complexLeft="Complex$", dash="$", complexRight="$", then compoundName[
  fred,wilma] returns "Complex$fred$wilma$". See also intermediateCompound, dash.
```

connectionMatrix

```
connectionMatrix[g] returns a matrix of ones and zeros, where
  m[i,j] is one of there is a link from node i to node j and zero otherwise.
connectionMatrix[g,form-> MATRIX] returns it filtered by MatrixForm.
conectionMatrix[g,form-> PLOT] returns it as a density plot.
```

consolidateDomains

```
consolidateDomains[listOfDomains, options] consolidates the domains in
  a string that contains an SBML list of domains, in the sense that all symbols
  that represent the same domain are combined in a single domain definition.
```

consolidateOptions

consolidateOptions[options] returns an option list with only the first occurance of any particular option included. For example, consolidateOptions[A→ 96,A→ 1, B→ 17, C→ 12, B→ 93] returns the option list {A→96,B→17,C→12}

convertGK

convertGK generates a single differential equation by combining the terms generated by subconvert. It is not intended to be invoked directly. It is called by interpret when necessary.

countTerms

countTerms[x] counts the terms in a system of dfiferential equations produced by interpret. The argument of x must have the form [{{ode1, ode2, ode3,...}, {var1, var2,...}}] where odei is a diff eqn such as A'[t]== .... At the present time it is non-function (returns an incorrect value)

cpDomain

cpDomain[d1, d2] represents the cross product between two different domains. To instantiate, use expandDomain.

createCelleratorReference

createCelleratorReference[] generates a Mathematica-reference style description of all Cellerator commands in alphebetical order. All commands that have a "usage" string that have been loaded into memory will be included, even if these commands are not Cellerator commands.

createDistanceRules

createDistanceRules[g] creates rules of the form Distance[i,j]-> distance[g,{i,j}]; these rules are needed by run because the distances between nodes are stored in the uninstanted function Distance, which must be evaluated before a system of odes is passed off to NDSolve. distance[g,{i,j}] returns an actual formula in terms of the embedding variables. createDistanceRules is used by run when option run->True

createGardnerModel

createGardnerModel is a function called to modularize createMitoticOsillator; it is not intended to be used directly.

createGOLDBETERmodel

createGOLDBETERmodel is a function called to modularize
createMitoticOsillator; it is not intended to be used directly.

createGRNvSBML

createGRNvSBML[GRN$PROTEIN, compartment, options] creates SBML for a
function GRNv[celli,proteinj] which returns the value of the protein j in cell i.

createIndexedSpecieDefinitions

createIndexedSpecieDefinitions[options] returns {domain-definition-sbml,
specie-definition-SBML } for all the Cellerator Indexed Species in the model

createLambdaMatrix

createLambdaMatrix[g, options] creates a set of Mathematica rules that implement the
geometrical connectivity (upper case lambda, Λ(i,j)). These Mathematica rules are used
by integratedGraph to generate Λ(i,j) as a dynamic function of the graph variables.
This function is not intended for end user execution. It is used by integrateGraph.

createLAMBDASBML

createLAMBDASBML[compartment,options] creates an SBML level 2 function definition for the LAMBDA
function. LAMBDA[I,J] is the geometric connectivity between graph node [i] and graph
node [j]. The SBML is created from the global LAMBDA$Formula[I,J], which is normally
created automatically by the run[] function. createLAMBDASBML is called automatically by
writeSBML and is not normally intended to be user-invoked. Options are:variables, indent.

createMassGrowthSBML

createMassGrowthSBML[compartment, options] creates SBML
for a function that describes mass growth as a function of current mass.
The mass growth function name is dmdt.
Options are:
indent->""
Or any valid option for mass$formula.

createMitoticOscillator

createMitoticOscillator[options] returns a model domains for the specified oscillator. The model domain includes all the proteins for the mitotic oscillatoras well as the corresponding cell division domain variables splv and tspl.

Options are
name->Goldbeter. If it is one of the Library models (see mitoticOscillator) control is passed to the appropriate model (for specific options see mitoticOscillator), in which case equations and reactions are ignored,.
variables->{}, list of varialbe names in the model. Required for non-library models. May be superceded in some of the library models if an insufficient number of names is provided.
equations->{} list of differential equations for cell division, ignored for the Library models. Supercedes the *reactions* option.
reactions->{}, list of Cellerator Reactions to be sent to interpret for the cell division reactions, ignored for the library models. Will be ignored if *equations* option is used.
threshold->0.7 value that *thresholdVariable* must be to cause cell division to occur
ic->{} or initial conditions for each variable; may be a Held formula which is to be evaluated for new nodes after cell division. ReleaseHold will be applied to the differential equations just before integration. This way the ic can be applied in formula form to the daughter cells as well.
index->1 or index number to add to each variable, e.g.3 to turn {C,M,X} into {C[1],M[1],X[1]}; this should be the node number of the cell

Usage note: The first time this is called, the global variable $FirstMitoticOscillator should be set to True. the global variables$CelleratorMitoticIC and $CelleratorMitoticVariables will be defined the first time this is called, and then will be used on all subsequent calls. During the first call the global variable $FirstMitoticOscillator will be set to False, to ensure that all subsequent calls use the same $CelleratorMitoticIC and $CelleratorMitoticVariables. To begin with a new set of variables or initial condition rule set $FirstMitoticOscillator=True. ReleaseHold will be applied to $CelleratorMiticIC to define the actual initial conditions.

createNonIndexedSpecieDefinitions

createNonIndexedSpecieDefinitions[options] creates the level 2 sbml for non-indexed species, i.e., on those that have one instantiation per cell. Thus M[1], M[2], ... are interpreted as cell[1].M, cell[2].M, ...; variables like M[1,2,3][4] will be ignored!! Options are the same as createSpecieDefinitions

createNorelModel

createNorelModel is a function called to modularize createMitoticOsillator; it is not intended to be used directly.

createRef

createRef[context] creates a quick-reference manual of all functions in the given context. createRef[Global] produces the Cellerator Reference without the heading message. createRef[] produces the usual Cellerator reference. createRef[System] produces a quick-reference for all Mathematica objects with usage strings.

createSBML2forANewDomain

```
createSBML2forANewDomain[name, upper, lower, tabs] creates the level 2 SBML for a new domain.
name is a single SBML symbol name or a list of sbml symbol names for the same domain, and may be
  supplied as a single symbol or character string or a list of symbols or character strings;
upper is the upper limit o the do main;
lower is the lower limit of the domain (Default value is 1);
tabs is an optional string to add to the front of each line of text (default is two
  tab stops).Example: createSBML2forANewDomain[{fred,barney,wilma}, 10, 5,""] returns:
<domain upperbound="10" lowerbound="5">
    <listOfSymbols>
        <symbol name="fred"/>
        <symbol name="barney"/>
        <symbol name="wilma"/>
    </listOfSymbols>
</domain>
```

createSBML2forASparseDomain

```
createSBML2forASparseDomain[name, range
  tabs] creates the level 2 pseudo-SBML for a sparse domain.
name is a single SBML symbol name or a list of sbml symbol names for the same domain, and may
  be supplied as a single symbol or character string or a list of symbols or character strings;
range is an explicit list of integers over which the domain is defined.
tabs is an optional string to add to
  the front of each line of text (default is two tab stops).
WARNING: The code returned is not valid SBML.
Example: createSBML2forASparseDomain[{fred,barney},{4,9,23}] returns
        <domain range="[4,9,23]">
            <listOfSymbols>
                <symbol name="fred"/>
                <symbol name="barney"/>
            </listOfSymbols>
        </domain>
```

createSpecieDefinitions

```
createSpecieDefinitions[options] returns sbml for a list of indexed species.
Options are:
compartment->name of compartment
domain->name of domain
species->list of Mathematica variable names to be turned into SBML species
variables->list of Mathematica initial conditions for the variables
indent->optional string to add to the front of each line.
```

createTyson1991SixVariableModel

```
createTyson1991SixVariableModel is a function called to
  modularize createMitoticOsillator; it is not intended to be used directly.
```

createTyson1991TwoVariableModel

```
createTyson1991TwoVariableModel is a function called to
  modularize createMitoticOsillator; it is not intended to be used directly.
```

createUserModel

```
createUserModel is called by createMitoticOscillator
  to create a user defined mitotic oscillator.Options:
initialConditions→{value1, value2,...}, padded with zeroes to length of variables
equations→{ode1,ode2,...}, list of differential equations. First component of output
  of interpret. The equations will be ignored if the reactions option is used.
reactions→{reaction1,reaction2,...}, list of reactions to be sent to interpret.
  If reactions is used, then equations and variables are ignored; they will be
  determined automatically by interpret, which will process the reactions given.
threshold→1
thresholdVariable→variableName, variable to test against threshold for
  cell division; if not specified, defaults to first variable in variables option
variables→{var1,var2,....} list of variable names. The variables
  option will be ignored if the reactions option is used.
Example:
sys=interpret[{{A→B,k1}, {B+C⇄Q,k2,k3}}]
createUserModel[variables→ {B,A,C},equations→ First[sys],
  index→ 42,threshold→ 42,initialConditions→ {12,24},thresholdVariable→ C]
returns:
modelDomain[odes→{A[42]′[t]==-k1 A[42][t],B[42]′[t]==k3 Q[t]+k1 A[42][t]-k2 B[42][t] C[42][t],
  C[42]′[t]==k3 Q[t]-k2 B[42][t] C[42][t],Q′[t]==-k3 Q[t]+k2 B[42][t] C[42][t],splv[42]′[
  t]==UnitStep[-17.3`+C[42][t]] UnitStep[-cellerator$cellDivisionMassThreshold+mass[42][t]],
  tspl[42]′[t]==UnitStep[-1.`*^-8+splv[42][t]]},molecules→{B[42],A[42],C[42],splv[42],tspl[
  42]},ic→{B[42][0]==12,A[42][0]==24,C[42][0]==0,splv[42][0]==0,tspl[42][0]==0},time→0]
createUserModel[reactions→ {{PQ→QR,1}, {PC⇄PQ,2,3.7}}]
returns:
modelDomain[odes→{PC[1]′[t]==-2 PC[1][t]+3.7` PQ[1][t],PQ[1]′[t]==2 PC[1][t]-4.7` PQ[1][t],QR[1]′[
  t]==PQ[1][t],splv[1]′[t]==UnitStep[-cellerator$cellDivisionMassThreshold+mass[1][t]] UnitStep[
  -1+PC[1][t]],tspl[1]′[t]==UnitStep[-1.`*^-8+splv[1][t]]},molecules→{PC[1],PQ[1],QR[1],splv[
  1],tspl[1]},ic→{PC[1][0]==0,PQ[1][0]==0,QR[1][0]==0,splv[1][0]==0,tspl[1][0]==0},time→0]
```

crossProduct

```
crossProduct[a, b] gives the set cross product, i.e., the set of lists {ai, bj} where ai is an
  element of a and bj is an element of bj. Not to be confused with Cross (vector cross product).
```

crunchGRN

```
crunchGRN[options] is called by combineHillReactionsByProduct
  to generate a validated option list for grnExpFunction.
```

crunchHill

crunchHill[options] is called by combineHillReactionsByProduct
to generate a validated option list for hillFunction.

crunchNHCA

crunchNHCA[options] is called by combineHillReactionsByProduct
to generate a validated option list for NCHAFunction.

dash

The symbol dash contains the character used to create names of molecular complexes. The default value is '__'. It is used as follows: the complex formed by joining X and Y is called X__Y. A new symbol can be selected by redefining the value of dash. The default value has been changed to "\[UnderBracket]" because of difficulties displaying the Sharp character on some operating systems (rev. 4/1/02). See also intermediateCompound, compoundName

debugPrint

debugPrint[*debug, m1, m2, m3*] prints the message '*m1*: *m2*: *m3*' if the flag *debug* is True

dependentVariable

dependentvariable[*differentialEquation*]
returns the variable on the left hand side of *differentialEquation*.

dhmsDate

dhmsDate[] returns a string of the form
"Feb. 15, 2002, 10:50:59"
for the current date and time; compare with Now

digest

```
digest[reactionList] returns a list {{specie1, list of reactions
  involving specie1}, {specie2, list of reactions involving specie2},...},
digest[reactionList,{f1,f2,...}] returns the same list assuming that species f1, f2,... are
  held constant, i.e., all of the species f1, f2 are ommitted from specie1, specie2, ...
For example, digest[xlateGK[{A+B→C,B+C→F,F→∅}],{A}] returns
 {{{B,{{A,B}→{C},{B,C}→{F}}},{C,{{A,B}→{C},
  {B,C}→{F}}},{F,{{F}→{∅},{B,C}→{F}}},{∅,{{F}→{∅}}}},{A,B,C,F,∅}}
The input to digest can be either cellerator arrow reactions or the ouput
  of xlateGK. There is normally no reason for the user to ever invoke
  digest directly. It is invoked automatically, when required, by interpret.
```

directory

```
directory is an option for writeSBML that specifies the name of the directory that the output
  file is to be placed in. If the directory is not specified than the directory returned by
  the Mathematica Directory[] function is used is used. This directory is system dependent.
```

disjointUnion

```
disjointUnion[x1,x2,...] forms a union of all the lists x1, x2, where
  each element is tagged with set it comes from. For example disjointUnion[
  Range[5], {A,B,C}] returns {{1,1},{1,2},{1,3},{1,4},{1,5},{2,A},{2,B},{2,C}}
```

displayRates

```
displayRates[] displays an annotated list of the rate constants
  in the global rateDB for an already-interepreted list of reactions.
displayRates[DB] displays the annotated list for a specific database of the same format
Options are
sort->"Rate" - sort (alphabetically) by rate constant (default);
sort->"Reaction" - sort alphabetically by reaction.
interpret->True(default)/False, if True, the names of the parameters
  are annotated and returned as strings; if False, they are returned as symbols.
  When interpret→True, all parameters are returned, included ones that are hard-
  coded as constants; when interpret→False, only symbolic parameters are returned.
table->True(default)/False, if True, print in TableForm, if False, return as a list.
values-> True (default)/Falase, show values of rate constants; requires
  interpret->True; if no value is specified, the string "Unassigned" is used
Normal usage would be:
interpret[... a list of reactions ... ];displayRates[];

example
c={{X→ XSTAR,kx}, {XSTAR↦Y,hill[vmax→vx,khalf→K]}};
i=interpret[c];rateDB[];

The following would be displayed (but in TableForm so the columns are lined up) to the screen:

Rate constants        Reaction
K (hill khalf)        XSTAR ↦ Y
kx                    tX → XSTAR
Non-symbolic Value = 0 (hill basalRate)        XSTAR ↦ Y
Non-symbolic Value = 1 (hill nhill)        XSTAR ↦ Y
Non-symbolic Value = 1 (hill Thill)        XSTAR ↦ Y
vx (hill vmax)        XSTAR ↦ Y
```

distance

```
distance[g,{n1,n2}]  or distance[g, n1,n2] gives the distance (a formula) between two nodes
  in a graph, even if they are not linked. See also linkDistance, linkLength,icDistance.
```

Distance

```
Distance[i,j] is an uninstantiated function that is used by Cellerator to hold a place
  for the distance between nodes i and j in a graph. When instantiated at run time
  Distance[i,j] is replaced by distance[g,i,j], where g is the name of the graph,
  which evaluates to Sqrt[(x[i][t]-x[j][t])^2+(y[i][t]-y[j][t])^2+(z[i][t]-z[j][t])^2]
```

dmax

```
dmax is an option for run[graph,...] is the distance at which links are broken.
  Links will only be broken if they grow from less than dmax to greater than dmax
  during an integration step, i.e., if they are already > dmax initially, the link
  will remain intact. To inhibit checking for link breakage, use breaklinks-> False
```

domainDescriptionQ

domainDescriptionQ[expression] returns True if expression is a correctly formatted
  Cellerator for SBML domain description, which has the form {name, {value1,value2,...}}

dotHolder

dot is a global symbol used during Cellerator
  output translation to hold the desired text symbol used for a period.

embeddingDomain

embeddingDomain[*options*] represents a cell's embedding
embeddingDomain[*emb*, *options*] where *emb*
  is an embeddingDomain[] returns a modified embeddingDomain
embeddingDomain[nodeDomain[...]] extracts the embeddingDomain from the nodeDomain
embeddingDomain[graphDomain[...],n] extracts the embedding of the nth node of the Graph

Options are:
position→{x,y,z}: variable names for the coordinates (must be precisely 3 dimensions; 2
  dimensional objects have all z-coordinates set to zero but still retain the variable)
ic→{*value, value, value*} or {*x*[*t0*]==*value*, *y*[*t0*]==*value*,...} :
  list of initial conditions, all default to zero
time→0: time at which to apply initial conditions

embeddingDomains

embeddingDomains[graphDomain[...]] returns a list of all the embeddings in the requested graph

embeddingIC

embeddingIC[embeddingDomain[...] retrieves the list of initial
  conditions for the specified embeddingDomain in the form {x[t0]==xvalue,...}
embeddingIC[graphDomain[...],n] retrieves the list of initial
  conditions from the nth node of the specified graph
see also: embeddingICvalue, embeddingICs, embeddingICEquations

embeddingICEquations

embeddingICEquations[graphDomain[...]] retrieves the initial conditions from every embedding in
  the graph as a single list of equations; see also embeddingICs, embeddingICvalue, embeddingIC

embeddingICs

embeddingICs[graphDomain[...]] retrieves the list of initial conditions from every node of the specified graph as a list of list of values (not equations): {{x1value,y1value,z1value},{x2value,y2value,z2value},...} . See also: embeddingIC, embeddingICvalue, embeddingICEquation

embeddingICvalue

embeddingICvalue[embeddingDomain[...]] retrives the initial conditions from the specified embedding domain as a list {xvalue, yvalue, zvalue}
embeddingICvalue[graphDomain[...],n] retrieves the same list from the embedding of the nth node of the graph See also: embeddingIC, embeddingICs, embeddingICEquation

embeddingODEs

embeddingODEs[embeddingDomain[...] retrieves the list of differential equations for the specified embeddingDomain
embeddingODEs[graphDomain[...]] retrieves the embedding differential equations from all embeddingDomains in the specified graph
embeddingODEs[graphDomain[...],n] retrieves the embedding differential equations from the nth node of the specified graph

embeddingPOSITION

embeddingPOSITION[embeddingDomain[...] retrieves the list of position variable names for the specified embeddingDomain
embeddingPOSITION[graphDomain[...]] retrieves the embedding position variable names from all embeddingDomains in the specified graph
embeddingPOSITION[graphDomain[...],n] retrieves the embedding position variable names from the nth node of the specified graph

embeddingTime

embeddingTime[embeddingDomain[...]] retrives the time from an embeddingDomain
embeddingTime[graphDomain[...]] retrieves the time from the embedding of the first node in the graph.

EmptySet

EmptySet is a variable that is used to represent the symbol ∅ when SBML is generated.

eq2Rule

```
eq2Rule[expression] will convert an equation of the form a==b into
  a rule a→b; no additional processing is done on either a or b. If expression
  is not of the form a==b then an errorAbort will occur. See also equal2Rule
```

equalToRule

```
equalToRule[expr] converts an expression of the form "x==y" to an expression of the form "x→y".
  If x or y is not an atom, e.g, it has a form like x[3][17,2] only the Head of x will
  be returned. If the argument is not of the form  "x==y"  it is returned unchanged. For
  example equalToRule[fred==wilma]  and equalToRule[fred[barney,dino]==wilma] both return
  fred→wilma, while equalToRule[(a+b)/2] returns (a+b)/2.equalToRule is used by sprint
  to pre-process initial conditions into the form required by run. See also equalToRule
```

equationListQ

```
equationListQ[x] returns True if x is a list of the form {eq1, eq2, ...}
  where all the eqi are equations, i.e., equationQ[eqi] is true for all i.
```

equationQ

```
equationQ[x] returns True of x is an equation, i.e., an
  expression of the form Equal[...], such as a==b, and returns False otherwise.
```

errorAbort

```
errorAbort[]: prints 'ERROR:Further Processing Aborted' and aborts.
errorAbort[test, options], aborts if test test is True.
errorAbort[options], options are:
    id→name of calling module (used for message)
    message→message to print in case of abort
```

euclideanDistance

```
euclideanDistance[p1, p2] calculates the euclidean distance
  between two poitns. Coordinates are zero-filled to have the same dimenstion.
  Scalars are converted to vectors with the scalar value in the x-coordinate.
```

evaluateEmbedding

```
evaluateEmbedding[embeddingDomain, solution, t] returns a new embeddingDomain with
  updated initial conditons drawn from the interpolating function solution at time t
```

evaluateGraph

evaluateGraph[*graphDomain*, *solution*, t] returns a new graph with updated
initial conditons drawn from the interpolating function *solution* at time t

evaluateModel

evaluateModel[*modelDomain*, *solution*, t] returns a new modelDomain with
updated initial conditons drawn from the interpolating function *solution* at time t

evaluateNode

evaluateNode[*nodeDomain*, *solution*, t] returns a new nodeDomain
with updated initial conditons (for all model domains and the embedding
domain in the node) drawn from the interpolating function *solution* at time t

evaluateVariable

evaluateVariable[*variable*, t, *solution*] returns the
value of *variable* at time t in the interpolating function *solution*.

exclude

exclude is an option to run[graph,options] and writeSBML[graph,options] that specifies which
variables are to be excluded from the grn equations. All proteins in the graph are included
except for those listed in exclude. If the option include is specified, then the exclude
option is ignored and only those listed in include are included in the grn equations.

excludeOption

excludeOption[string, option-list]
excludeOption[{string1,string2,string3,...}, option-list]
This function returns an option list with the options string1->
value,string2->value,... removed from the original list (if they are there).

exclusionList

```
exclusionList[options] generates a list of variables to exclude from processing; used by run.
exclusionList[graph,options] adds all graphVariables[graph] to the variables specified by the
  variables option, making the use of the variables option typically unnecessary in this case.
Options are:
     exclude->{var1,var2,...} if this list is non-
  empty, it is immediately returned unless the include options is used
     include->{var1,var2,...} if this list is non-empty, the exclude option is ignored. A list
  of variables to specifically include, to the exclussion of all others specified in variables
     variables->{var1, var2, ...} list of all the variables in the system.
```

expandAllLHSBrackets

```
expandAllLHSBrackets[circuit] returns a modified circuit with all
  brackets of the form {a,b,c,...} ↦ d broken down into separate reactions with
  the same rate constants. circuit is any circuit that is compatible with interpret
```

expandGraph

```
expandGraph[g] expands any pointers within the graph g; g may itself be a pointer
  to a graph domain. A Cellerator pointer is the expression pointer[s] where s is
  a string giving the name of a global object that contains the true data structure
```

expandGraphVariables

```
expandGraphVariables[v,s] returns a list of possible variablesfor plotting when
  s is a run[graph,...] solution and v is a variable name or a list of variables.
  For example if the solution has 10 cells each of which have variables aMPF
  and pMPF in them, expandVariables[{aMPF[3], pMPF},s] returns {aMPF[3],pMPF[1],
  pMPF[2],pMPF[3],pMPF[4],pMPF[5],pMPF[6],pMPF[7],pMPF[8],   pMPF[9],pMPF[10]}.
Options are:
verbose→ False, if True, will print a message
  to the screen for each variable that is expanded by adding indices.
```

expandLHSBrackets

```
expandLHSBrackets[{reaction,rates}] expands reactions of the form {a+b+c... ↦
  d, opts} into {{a↦d, opts}, {b↦d,opts},...}. See also expandallLHSBrackets
```

expandRateConstant

```
expandRateConstant[k] returns a (possibly annotated) list of rate
  constants as strings. The main purpose is to produced interpreted listings for
  transcriptional reactions, whose rate constants are stored as unevaluated functions.
expandRateConstants[k23] returns {"k23"}
expandRateConstant[hill[vmax -> v1, khalf -> K, nhill ->
  n, Thill -> T, basalRate -> r]] returns the list  { "K (hill khalf)","n (
  hill nhill)","r (hill basalRate)","T (hill Thill)","v1 (hill vmax)"}
Numerical values are annotated as Non-symbolic values, missing values as Not Specified, e.g.,
expandRateConstant[GRN[nGRN→ n1, hGRN→ h42, RGRN→ 17]] returns
  {"h42 (hGRN)","n1 (nGRN)","Non-symbolic Value = 17 (RGRN)","TGRN (TGRN)"}.
This function is used by displayRates to display the table of rate constants in an (
  already-interpreted) set of reactions, and is not normally called by the end user.
```

expandSTNRunVariables

```
expandSTNRunVariables[v,r] returns a list of variables that match v in the the run[
  STN,...] output r. v may be a single variable or a list of variables.  A variable
  is said to match v if it has the same name or the same name with an index. For
  example, expandSTNRunVariables[{m,MEKP,RAFPhosphatase,K},r] may return {MEKP,
  RAFPhosphatase,K[1,0],K[1,1],K[1,2],K[2,0],K[2,1],K[2,2],K[3,0],K[3,1]} indicating
  that m is not allowed, while K expands to a number of indexed variables.If v=All,
  a list of all variables will be returned. If v=None, an empty list will be returned.
```

expandSum

```
expandSum[sumOverDomain[expression]] expands the actual
  sum represented by expression, e.g., expandSum[sumOverDomain[x[j],{{j,
  Range[5]}}]] returns x[1]+x[2]+x[3]+x[4]+x[5]. See also expandSums.
```

expandSums

```
expandSums[expression] expands all of the sumOverDomain functions
  in expression, i.e., replaces every occurance of sumOverDomain[...] with
  the actual expanded sum. See also: expandSum,sumOverDomain. Example:
s1=sumOverDomain[x[j],{{j,Range[5]}}];
s2=sumOverDomain[x[j]y[j]+y[k],{{j,{1,2,3}},{k,{2,4}}}];expandSums[{s1+s2,s2}]
returns
{x[1]+x[2]+x[3]+x[4]+x[5]+2 x[1] y[1]+3 y[2]+2 x[2] y[
  2]+2 x[3] y[3]+3 y[4],2 x[1] y[1]+3 y[2]+2 x[2] y[2]+2 x[3] y[3]+3 y[4]}
```

expandTree

```
expandTree[x] expands a tree structure in terms of its basic implementation.
```

exportODEs

```
exportODEs[s,options] will export a system of differential
  equations to a file with the selected format; s is the output of interpret.

Options are:

exportPath→"filename" (default is odes.dat)
exportFormat→format where format is a one of the following strings:
"mathematica" (default): save as a mathematica list
"list": same as "mathematica"
"text": a text listing, one equation per line
"html": an html file
"mathml": write equations using mathml
"html+mathml": an html file with equations embedded as mathml rather than graphics files
"C": write equations in C. Note: pointer/array declarations need to be edited manually.
"FORTRAN": write equations in FORTRAN. Note: Dimension states may need to be edited manually.
```

extractRateConstant

```
extractRateConstant[dbentry] returns a symbolic rate constant from
  a database entry in rateDB. The format of a database entry is {reaction,
  rate constants}.extractRateConstant is called by displayRates to display an (
  already interpreted) database, and is not typically needed by the end-user.
```

extractReaction

```
extractReaction[dbentry] returns a reaction from a database
  entry in rateDB. The format of a database entry is {{a,b,...}→{A,B,C,...},
  rate constants}.  extractReaction is called by displayRates to display an (
  already interpreted) database, and is not typically needed by the end-user.
```

fastSpringForce

```
fastSpringForce[g,n] calculates the total spring force on node n in graph g
fastSpringForce[g] returns a list {force on node1, force on node2, ...} for the entire graph
fastSpringForce does not use sumOverNeighbors because this is grossly inefficient. See also
  springForce. fastSpringForce is turned on in organism[] and meristem[] using option fastForce.
```

fccTable

```
fccTable[r] generates a hemispherical array of 3 dimensional points on a face-centered-cubic
  grid. The argument r is the radius of the grid in cell diameters and hence should be an integer.
```

field

field[function,domain] is a domain that represents the effect of a function on a
domain. To instantiate the results, use expandDomain. For example, expandDomain[
field[Sqrt,intDomain[100,250,75]]] returns {10,5 $\sqrt{7}$,5 $\sqrt{10}$ }. See also threadfield

file

file is an option for writeSBML that specifies the name of the output file.
If the directory option is also specified, the file is named directory/file. If
file is not specified a file name is generated based on the current system clock.

findBlockedLinks

findBlockedLinks[options] returns a list
of links that are blocked in the given graph. Options are: graph

findIntersection

findIntersection[{x0,y0},r, {x1,y1}, {x2,y2}] finds the points of intersection of
the line from {x1,y1} to (x2,y2} with the circle of radius r centered at {x0,y0}]
findIntersection[options] is an alternate form with the following options:
center→{0,0}
radius→1
endPoints→{{x1,y1},{x2,y2}}
plot->False, if True, draws a plot of
the points, the circle and the line segment connecting them
findIntersection returns a null list {} if there are no points of intersection.

findParameterValues

findParameterValues[x1,x2,...,xn,optionlist] returns a list {value1, value2, ..., valuen},
where the values are derived from the optionlist, or are zero if they are not found
Example: findParameterValues[f,g,h,x→ 23,g→ 14,k→ 27] returns {0,14,0}.

findSteadyState

findSteadyState[STN, options] returns the steady steady state of a system of reactions. STN is
the same format as either the input to or the output from interpret or may be a list of
equations. Any option for interpret may be used. The steady state(s) are returned as rule
lists, and some (or all) steady states may not be found since this function uses Solve.
Example: findSteadyState[{{TF↦P, type→ hill,nhill→ 2},{P→
∅,k1}}] returns {{P -> TF^2/(k1*(1 + TF^2))}}

flatTable

```
flatTable[r] generates a list of coordinates for cell centers for a two-dimensional
  hemispherical meristem, where r is the radius is cell diameters of the meristem.
  The cell centers are aligned along a hexagonal grid with spacing of one unit.
  The coordinates are embedded in a three-dimensional coordinate system, but all
  cell centers lie in the x-y plane, i.e., the third coordinate is always zero.
```

fuzzyStep

```
fuzzStep[x,r] approximates a step function at x=0 with a Tanh of slope r (default 1000)
```

generateColorTable

```
generateColorTable[graph, options] returns a list of the form
  {{nodetype, color}, {nodetype, color},..} to be used to plot the graphs.
Options are:
colorChart->AllColors (default) or a list of colors to use to represent all
  node types. If fewer colors are provided than there are node types in the graph
  then additional colors are applied cyclically from the global list AllColors.
nodeColors-> Automatic (Default) or {{nodetype,color},
  {nodetype, color}...} to override this function.
NOTE: Not intended for end user use; this is called by showGraph, showFlatGraph, and makeMovie.
```

generateICList4SBML

```
generateICList4SBML[{x1,x2,x3,...},opt] returns a list of
  initial condition assignments of the form {x1->val2,x2->val2,...}. Initial
  conditions may be passed in the form of initialConditions->{y1->val1,y2->
  val2,...}. If an initial condition is not specified it is set to zero.
```

generateMeristemLinks

```
generateMeristemLinks[{{x1,y1,z1},{x2,y2,z2},...}, options] generates a
  table of link pairs {{n1,m1},{n2,m2},{n3,m3},...} indicating that each cell n(
  i) should be connected to cell n(j).  The numbers n(i) are assigned based on
  the order in which the coordinates are presented to the function, i.e., node 1
  is the node is the first triple {x1,y1,z1}, node 2 is the second triple, etc.
Options are:
grid->fcc, hcp, flat, how are the cells arranged.
```

generateOutputFileName

```
generateOutputFileName[options] returns an output file name specification. No file is created or
  opened and no error checking is performed to see if the file or directory actually exists.
  All this function does is return a string with the desired file name specification.
If no options are specified, the file name is "dir/runYYYYMMDDThhmmss.
  sbml" where dir is the current working directory given by Directory[];
  to change the workining directory, use SetDirectory["newDir"];
Options are
file->file name
directory->directory name
filetype->file type
```

generateReactionSBML

```
generateReactionSBML[r, options] returns a string containing
  a complete level-1 SBML specification of the reactions in r. Options are:
model→ nameOfModel
note→ html to be included in <notes> field of generated SBML
literalnote->additional html to be added that will be
  included literally, without any additional formatting such as line wrapping
compartment→name of compartment that all variables are included in
author->string, optional name of author to be appended to the note
indent→"", additional string to write at beginning of each line of SBML
In addition, values for rate constants may be specified as name→value,name→value,...
```

genericValidateRateConstant

```
genericValidateRateConstant[{expr,k},options] is called
  by validateRateConstants.  expr must be a Cellerator arrow  expression;
  k is the supplied rate constants (might be none); and options are:]
total→1, number of required rate constants
repeat→Infinity, number of rate constants to generate before cyclically repeating them
verbose→ False, if True, print the reaction and rate constant to the screen.
validate->True, if False, don't automatically generate rate constants.
```

getEnzyme

```
getEnzyme[reaction] returns the catalyst in a catalytic reaction such as A ⇄ B
                                                                          C
```

getEquationOrder

```
getEquationOrder[eq] returns the order of the derivative for an equation of
  the form lhs==rhs, where lhs is a derivative of the form x'[t], x''[t], D[x[t],
  t], D[x[t],{t,n}], etc.; if eq is not an equation (i.e., it does not have Head[
  eq]==Equal, a value of -1 is returned. Algebraic equations have an order of 0.
```

getGrowthRates

getGrowthRates[n, options] checks the option list for the parameters rates or periods and returns a formatted list to 'organism'. Not intended to be user-invoked, only called by 'organism'.

getInitialValue

getInitialValue[var, {var1[t0]==val1, var2[t0]==val2,...}] searchs the list of initial conditions for a string of the form var[t0]==value and returns that value;
getInitialValue[{v1,v2,...}, {var1[t0]==val1, var2[t0]==val2,...}] returns a list of initial values for the variables v1,v2,... in the list of initial conditions provided by calling getInitialValues.
getInitialValue[var, graph] finds the initial value for variable var in the graph.
getInitialValue[{var1,var2,...}, graph] returns a list of initial values for the variables var1, var2,... in the graph by calling getInitialValues.

getInitialValues

getInitialValues is used by getInitialValue[list,...] when the first argument is a list; there is typically no reason for the user to invoke getInitialValues directly because getInitialValue will figure out on its own when getInitialValues needs to be called.
getInitialValues[{v1,v2,...}, {var1[t0]==val1, var2[t0]==val2,...} returns a list of initial values for the variables v1,v2,... in the list of initial conditions provided.
getInitialValue[{var1,var2,...}, graph] returns a list of initial values for the variables var1, var2,... in the graph.
getInitialValue[var, graph] returns a list of initial values for the variables var[i1],var[i2],... in the graph.

getMassGrowthRateList

getMassGrowthRateList[mu, n] returns a list of mass growth rates of length n. The argument mu may be one of the following:
A single value, in which case a list {value, value, ..., value} of length n is returned;
A list of values, in which case the list mass is padded on the right to a length n with its first first element, i.e, getMassGrowthRateList[{0.03, 0.07, 0.065}, 5] returns {0.03, 0.07, 0.065, 0.03, 0.03}
A rule list of the form {mu[i1]→val1, mu[i2]→val2,...,mu[ij]→valj, default→*defaultvalue*} which sets specific values and pads the rest with *defaultvalue*.
getMassGrowthRateList is used by organism to determine the desired steady state masses that are placed in the steadyStateMass field of the nodeData field of each graph node.

getMassList

getMassList[mass, n] returns a list of masses of length n. mass may be one of the following:
A single value, in which case a list {mass, mass, ..., mass} of length n is returned;
A list of values, in which case the list mass is padded on the right to a length n with its first first element, i.e, getMassList[{3,7,12}, r] returns {3, 7, 2, 5, 5}
A rule list of the form {mass[i1]->val1, mass[i2]->val2,...,mass[ij]->valj, mass->*defaultvalue*} which sets specific values and pads the rest with *defaultvalue*.
getMassList is used by organism to determine the desired initial masses for the cells.

getMassRange

```
getMassRange[r], where r is the output of run[graph...] (or runFixedInterval) returns a
  rule "massRange→{min,max}" where min is the smallest mass and max is the largest mass listed
  in the initial conditions of the graphs. The function getMassRange is called by makeMovie and
  the information is passed to showGraph to determine the mass ranges for the plots. Options are
method→ START, only check the graph initial conditions, or ALL, check start and
  end conditions of each run step. (in-step interpolation is not performed; so long
  as mass is monotonic increasing, except at cell division, this is sufficient)
```

getMeristemGrowthRates

```
getMeristemGrowthRates[nlist, meristemGrowthRates-> {CZ->r1,PZ->
  R2,Rib->r3,Floor->r4}]      gives optional rates r1, r2, r3, r4 for a list
  of node types; nlist is a list of node types. getGrowthRates returns a list
  of the same length as nlist. Not intended for end-use. Used by 'meristem'
```

getNodeData

```
getNodeData[n] returns the nodeData from node n
getNodeData[g,n] returns the nodeData from the nth node of graph g.
getNodeData[graph] returns an ordered list of the nodeData field from all nodes in the graph.
```

getNodeType

```
getNodeType[n] returns the nodeType from node n (a nodeDomain)
getNodeType[g,n] returns the nodeType from the nth node of graph g. See also getNodeTypes
```

getNodeTypes

```
getNodeTypes[g] returns a list of the nodeTypes from all nodes in the graph g.See also getNodeType
```

getODE

```
getODE[g] returns a list of all the differential equations in graph g
getODE[g, variable→ x] returns a list of the
  differential equations for x'[t] and/or for x[j]'[t] where j is any index
getODE[g, variable→ x[i]] retursns the single differential eqution for x[i]'[t]
getODE[g, variable→ {v1, v2, ...}]
  returns the differential equations for the specified variables
```

getOption

```
getOption[function,optionName, optionList]
Searchs through optionList for 'option→value' and returns value.
```

getOptionSymbols

```
getOptionSymbols[] returns a list of the names of all
  the options that defined in a Global` context via the Options function.
```

getPointers

```
getPointers[nodeDomain] retursn a list of all the Cellerator pointers reference in the nodeDomain.
```

getSpecies

```
getSpecies[n,options] is used by organism to determine what species are wanted.
```

getSpeciesIC

```
getSpeciesIC[n,options] retrieves and formats the species initial
  conditions options for organism. Only minimal error checking is performed.
```

getSpeciesODEs

```
getSpeciesODEs[n,options] retrieves and formats the
  speciesodes options for organism. Only minimal error checking is performed.
```

getSpeciesReactions

```
getSpeciesReactions[n,options] retrieves and formats the species
  reactions option for organism. Only minimal error checking is performed.
```

getSteadyStateMass

```
getSteadyStateMass[n] returns the steadyStateMass option from the nodeData field of node n
getSteadyStateMass[g,n] returns the steadyStateMass from the nth node of graph g.
getSteadyStateMass[graph] returns an ordered
  list of the steadyStateMass fields from all nodes in the graph.
```

getSteadyStateMassList

getSteadyStateMassList[s, n] returns a list of
steady state masses of length n. s may be one of the following:
A single value, in which case a list {value, value, ..., value} of length n is returned;
A list of values, in which case the list mass is padded on the right to a length n with its
first first element, i.e, getSteadyStateMassList[{3,7,12}, r] returns {3, 7, 2, 5, 5}
A rule list of the form {mass[i1]→val1, mass[i2]→val2,...,mass[ij]→valj, default→
defaultvalue} which sets specific values and pads the rest with *defaultvalue*.
getSteadyStateMassList is used by organism to determine the desired steady state masses
that are placed in the steadyStateMass field of the nodeData field of each graph node.

getSymbols

getSymbols[expression] returns a list of all the symbols in expression whose context is Global`

getVariableRange

getVariableRange[r], where r is the output of run[graph...] (or runFixedInterval)
returns a rule "variableRange→{min,max}" where min is the smallest mass and max
is the largest mass listed in the initial conditions of the graphs. The function
getVariableRange is called by makeMovie and the information is passed to showGraph
to determine the variable ranges for colors of cells in plots. Options are
method→ START, only check the graph initial conditions, or ALL, check start and
end conditions of each run step. (in-step interpolation is not performed; so long
as mass is monotonic increasing, except at cell division, this is sufficient)
variable->mass, name of variable to use

globalGraphs

globalGraphs stores the most recently computed
list of graphs computed by runFixedInteval.See also recoverSolution.

globalSolutions

globalSolutiosn stores the most recently computed list
of solutions computed by runFixedInterval. See also recoverSolution.

globalSYSTEM

globalSystem stores the most recently generated list of differential equations.
It is sometimes useful for debugging early terminations in integrateGraph.

GMWC

GMWC is an unistantiated function that holds
  the options for Generalized Monod-Wyman-Changeux Reactions.
Use of the header "GMWC" is optional. A GMWC reaction may be specified
  as either {reaction, GMWC[options]} or {reaction, options}
The syntax for reaction is

$$\underset{\{\{A1,A2,...\},\{I1,I2,...\}\}}{\overset{\text{Enzyme}}{\{S1,\ S2,\ ...\}\Longrightarrow\{P1,\ P2,\ ...\}}}$$

where c, L, are constants; S is a list of substrates; A is a list of activating
  catalysts; I is a list of inhibiting catalysts; and KSi,KAi,KIi are constants.
  If there is only one product P then then curly brackets on {P} can be omitted.
The corresponding ODE is

$$\frac{dP}{dt}=\text{Enzyme}*\text{kcatGMWC}*\left(\left(\Pi\ \frac{Si}{KSi}\right)\left(\Pi\left(1+\frac{Si}{KSi}\right)^{n-1}\right)\left(\Pi\left(1+\frac{Ai}{KAi}\right)^{n}\right)+L*\left(\Pi\ c*\frac{Si}{KSi}\right)\left(\Pi\left(1+c*\frac{Si}{KSi}\right)^{n-1}\right)\left(\Pi\left(1+\frac{Ii}{KIi}\right)^{n}\right)\right)/\left(\left(\Pi\left(1+\frac{Si}{KSi}\right)^{n}\right)\left(\Pi\left(1+\frac{Ai}{KAi}\right)^{n}\right)+L\ *\left(\Pi\left(1+c*\frac{Si}{KSi}\right)^{n}\right)\left(\Pi\left(1+\frac{Ii}{KIi}\right)^{n}\right)\right)$$

Competitive inhibition can be added as

$$\underset{\{\{A1,A2,...\},\{I1,I2,...\},\{\{JS11,JS12,...\},\{JS21,JS22,...\},...,\{JA11,JA12,...\},\{JA21,...\},...\}}{\overset{\text{Enzyme}}{\{S1,\ S2,\ ...\}\Longrightarrow\{P1,\ P2,\ ...\}}}$$

Competitive inhibitors are specified as a list of sublists, in order: {CIS1,
  CIS2,...,CISN,CIA1,CIA2,...,CIAM}, where CISk and CIAj is a list of
  competitive inhibitors for the kth substrate and jth activator, respectively.
The ODE including competitive inhibition is

$$\frac{dP}{dt}=\text{Enzyme}*\text{kcatGMWC}*\left(\left(\Pi\ \frac{Si}{KSi}\right)\left(\Pi\left(1+\frac{Si}{KSi}+\Sigma_p\frac{JS_{ip}}{KJS_{ip}}\right)^{n-1}\right)\left(\Pi\left(1+\frac{Ai}{KAi}+\Sigma_j\frac{J_{ij}}{KJ_{ij}}\right)^{n}\right)+L*\left(\Pi\ c*\frac{Si}{KSi}\right)\left(\Pi\left(1+c*\frac{Si}{KSi}+\Sigma_p\frac{JS_{ip}}{KJS_{ip}}\right)^{n-1}\right)\left(\Pi\left(1+\frac{Ii}{KIi}\right)^{n}\right)\right)/\left(\left(\Pi\left(1+\frac{Si}{KSi}+\Sigma_p\frac{JS_{ip}}{KJS_{ip}}\right)^{n}\right)\left(\Pi\left(1+\frac{Ai}{KAi}+\Sigma_j\frac{J_{ij}}{KJ_{ij}}\right)^{n}\right)+L\ *\left(\Pi\left(1+c*\frac{Si}{KSi}+\Sigma_p\frac{JS_{ip}}{KJS_{ip}}\right)^{n}\right)\left(\Pi\left(1+\frac{Ii}{KIi}\right)^{n}\right)\right)$$

Defaults:
If there are no activators (inhibitors) this
  should be indicated as an empty list as in $\underset{\{\{a...\},\{\}\}}{\overset{e}{\{s\}\Longrightarrow P}}$ or $\underset{\{\{\},\{i...\}\}}{\overset{e}{\{s\}\Longrightarrow P}}$
A single substrate may be specified as an atom:

$$\underset{\{\{a,..\},\{i,...\}\}}{\overset{e}{s\Longrightarrow P}}\quad\text{is interpreted as}\quad\underset{\{\{a...\},\{i...\}\}}{\overset{e}{\{s\}\Longrightarrow P}}$$

If there is a single enhancer it is assumed to be an activator as in

$$\underset{\text{atom}}{\overset{e}{\{s\ ...\}\Longrightarrow P}}\ \text{is interpreted as}\ \underset{\{\{atom\},\{\}\}}{\overset{e}{\{s\ ...\}\Longrightarrow P}}$$

If all the enhancers are atoms they are assumed to be all activators:

$$\underset{\{atom1,atom2,...\}}{\overset{e}{\{s\ ...\}\Longrightarrow P}}\quad\text{is interpreted as}\quad\underset{\{\{atom1,atom2,...\},\{\}\}}{\overset{e}{\{s\ ...\}\Longrightarrow P}}$$

If there is a single activator or inhibitor they may be specified as atoms:

$$\underset{\{atom,list\}}{\overset{e}{\{s\ ...\}\Longrightarrow P}}\ \text{is interpreted as}\ \underset{\{\{atom\},list\}}{\overset{e}{\{s\ ...\}\Longrightarrow P}}$$

$$\underset{\{list,atom\}}{\overset{e}{\{s\ ...\}\Longrightarrow P}}\ \text{is interpreted as}\ \underset{\{list,\{atom\}\}}{\overset{e}{\{s\ ...\}\Longrightarrow P}}$$

See also GMWCRateFunction.
Options are
cGMWC→1
LGMWC→1
nGMWC→1
KGMWC→1, or KGMWC→{KS1, KS2,..., KA1,KA2,...,KJ1,KJ2,...}, listed in order:
  substrates, then activators, then inhibitors, then competitive inhitibors of

substrates, then competitive inhibitors of activators; if the list is too short, remaining values are set to 1. Only KJ's for non-null J lists need to be specified.
kcatGMWC→1, the entire reaction is multiplied by kcatGMWC. This option is included for similarity with other reactions.
Example:

$$\underset{\{\{A1,A2,A3\},\{I1,I2\},\{\{JS\},\{\},\{J2A,J2B\},J3\}\}}{\{S\} \overset{En}{\Longrightarrow} \{P\}}$$

means: A single substrate, single product, three activators, two inhibitors; Substrate S has a single comptitive inhibitor; activator A2 has two competitive inhibitors J2A and J2B; activator A3 has one competitive inhibitor J3; and A1 has no competitive inhibitors, The KD values woudl be specified as GMWC[KGMWC→ {KS,KA1,KA2,KA3,KJS, KI1,KI2, KJ2A,KJ2B,KJ3}].

GMWCRateFunction

GMWCRateFunction[Enzyme, A, B, C, GMWC[...]] returns

$$\frac{dP}{dt} = Enzyme * kcatGMWC * \left(\left(\Pi \frac{Si}{KSi}\right)\left(\Pi\left(1+\frac{Si}{KSi}\right)^{n-1}\right)\left(\Pi\left(1+\frac{Ai}{KAi}\right)^{n}\right)+L*\left(\Pi\, c*\frac{Si}{KSi}\right)\left(\Pi\left(1+c*\frac{Si}{KSi}\right)^{n-1}\right)\left(\Pi\left(1+\frac{Ii}{KIi}\right)^{n}\right)\right) / \left(\left(\Pi\left(1+\frac{Si}{KSi}\right)^{n}\right)\left(\Pi\left(1+\frac{Ai}{KAi}\right)^{n}\right)+L*\left(\Pi\left(1+c*\frac{Si}{KSi}\right)^{n}\right)\left(\Pi\left(1+\frac{Ii}{KIi}\right)^{n}\right)\right)$$

where S is a list of substrates; A is a list of activating catalysts; I is a list of inhibiting catalysts; and each J is a list of competitive inhibitors corresponding to the A's. The standard canonical form is

$$\underset{\{\{A1,A2,...\},\{I1,I2,...\}\}}{\{S1,\ S2,\ ...\} \overset{Enzyme}{\Longrightarrow} \{P1,\ P2,\ ...\}}$$

Values of the constants are specified through GMWC[] options. See also GMWC.

goldbeterMinimalSystem

goldbeterMinimalSystem[options] returns a modelDomain with a three-variable mitotic oscillator as described by A.Goldbeter,*Proc.Natl.Acad.Sci.USA*,88:9107-9111 (1991)
Options are:
goldbeterVariables-> {C,M,X} - alternate names for Goldbeter's three variables
goldbeterICs-> {value, value, value}; the keyword
Random generates an IC between 0 and 0.3 with a uniform distribution
rate-> 1, rate factor to multiply the equations by; with
goldbeterParameters->goldbeter, the period is apprxomately 20/rate
goldbeterParameters->goldbeter,keener,slow,stopped. If goldbeter, the parameters are chosen as per figure 10.6 of Goldbeter's book; If keener, as per table 13.3 in Keener and Sneyd, Mathematical Physiology; If Slow. a set that gives a slower oscillation is used; if Stopped, all are set to zero.

graphCounter

graphCounter counts the number Cellerator graph pointers that have been allocated

graphDomain

graphDomain[nodes→*nodelist*, links→ *linklist*, lineage→ *lineagetree*]
graphDomain[graphDomain[...],*options*]
nodelist = {nodeDomain[...],nodeDomain[...],...}
linklist = {linkDomain[...],linkDomain[...],...}
lineageTree = tree[...]

graphelator

```
Graphelator™ is a generic term for the parts of Cellerator™ that involve
  graph manipulation. There is no separate file for function called Graphelator.
```

Graphelator

```
Graphelator™ is a generic term for the parts of Cellerator™ that involve
  graph manipulation. There is no separate file for function called Graphelator.
```

graphIC

```
graphIC[g] returns a list of the initial conditions for all variables in the graph g.
graphIC[g,exclude->{var1,var2,...}, returns
  the initial conditions with the specified variables excluded
```

graphICrules

```
graphICrules[g] returns a list of the initial
  conditions for all variables in the graph g as a rule list.
```

graphLinks

```
graphLinks[g] retursn a list of the link Domains in g.
```

graphNodes

```
graphNodes[g] returns a list of the node Domains in g.
```

graphODEs

```
graphODEs[g] returns a list of the differential equations in the graph g
Options:
evaluate->True, expand pointers
exclude->{}, list of variables to exclude from list
```

graphParameters

```
graphParameters[g] makes a list of all
  the symbols used in g that are not listed in graphVariables[g]
```

graphQ

graphQ[x] returns True if x is a graphDomain (or a pointer to a graphDomain), and false otherwise.

graphRunSolution2gridMultiPlotSolution

graphRunSolution2gridMultiPlotSolution[s] converts s,
the output of run[graph...] to the format necessary for gridMultiPlot.

graphVariables

graphVariables[g] returns a list of all the variables in the graph g
graphVariables,g,exclude->{var1,var2,...}
will return a list with the specified varialbes excluded

greek2string

greek2string[x] returns a string with all greek characters replaced by their names.

greek2StringRules

greek2StringRules are the rules {α → alpha, β → beta, γ → gamma, δ → delta, ε → curlyEpsilon,
ζ → zeta, η → eta, θ → theta, ι → iota, κ → kappa, λ → lambda, μ → mu, ν → nu, ξ →
xi, ο → omicron, π → pi, ρ → rho, ς → finalSigma, σ → sigma, τ → tau, υ → upsilon,
φ → curlyPhi, χ → chi, ψ → psi, ω → omega, Α → A, Β → B, Γ → CapitalGamma, Δ →
CapitalDelta, Ε → E, Ζ → Z, Η → H, Θ → CapitalTheta, Ι → I, Κ → K, Λ → CapitalLambda,
Μ → M, Ν → N, Ξ → CapitalXi, Ο → O, Π → CapitalPi, Ρ → P, Σ → CapitalSigma, Τ →
T, Υ → CapitalUpsilon, Φ → CapitalPhi, Χ → X, Ψ → CapitalPsi, Ω → CapitalOmega}

grn

grn is an option for run[grpah,options] and writeSBML[graph,options] that specifies whether
or not the grn equations should be included along with the Cellerator equations.

GRN

GRN[options] is an unevaluated function used to specify
  parameters for reactions of the form A↦B to interpret and other functions.
Default options are RGRN→ 1, TGRN→ 1, nGRN→ 1, hGRN→ 0, Sigmoid->GRNexpFunction.
The function generated for A↦B is $\frac{r}{1+e^{-TA^n+h}}$ where RGRN→ r, TGRN→ T, nGRN→ n, hGRN→ h.
If multiple reactions $A_i \mapsto B$ are specified
  for the same product B then function becomes $\frac{r}{1+e^{-\left(\sum_i T_i A_i^{n_i}+h_i\right)}}$

Sigmoid->GRNSigmoidFunction returns $\frac{1}{2}\ r\ \left(1+\frac{h+T*A^n}{\sqrt{1+(h+T*A^n)^2}}\right)$ or $\frac{1}{2}\ r\ \left(1+\frac{h+\sum_i T_i A_i^{ni}}{\sqrt{1+(h+\sum_i T_i A_i^{ni})^2}}\right)$

A user-defined or pure function can also be specified. It should be a function
  of a single argument and enclosed in parenthesis. The function call will be
  F[r*({T1,T2,...}.{X1^n1, X2^n2,...}+h)], where F is the name of the function.
Examples: the following are equivalent:

```
...,Sigmoid→ ((1/2)(#/Sqrt[1+#^2])&),...
or
F[x_]:=((1/2)(x/Sqrt[1+x^2]));
...,Sigmoid->F,....
```

GRNEQ

GRNEQ[equations→ {equation1, equation2,...}, variables→ {variable1, variable2, ...}, parameters→
  {parameter1, parameter2,...}] is a data structure for holding a set of GRN-equations.
  To access individual fields see equations, parameters, parameterNames, variables.

GRNequations

GRNequations[g, options] returns a GRNEQ[]
  data structure that encapsulates the GRN equations for a graph.

Options are:
exclude→{protein1, protein2, ...} list of proteins to exclude from the list of GRN equations.
GRNMatrix→ {TIN,TOUT, TRL, TGR}, names of arrays, TIN[i,j] is the interaction strength for
  the affect of protein j on protein i in the same cell; TOUT[i,j] is the interaction
  strength for the affect of protein j in a neigboring cell on protein i in the current
  cell; TRL[r,l] is the effect of ligand l on recepter r, where r is some protein
  produced in a neighboring cell; TGR[g,r] is the effect of receptor r on gene r.
GRNg→ sigmoid, name of the function g(x)
includeArrows→ False, True to include Cellerator arrow generated terms in the odes also.

GRNRatesQ

GRNRatesQ[k] returns True if k is a valid rate constant
  list for a GRN reaction, i.e., it is either null, has precisely the form
  GRN[options], or is an option list that contains the option type→GRN.

GRNSigmoidFunction

GRNSigmoidFunction[x] returns $\frac{1}{2}\ (1+\frac{x}{\sqrt{1+x^2}})$;

GRN$PROTEIN

GRN$PROTEIN is a Cellerator global generated whenever GRN equations are requested. It contains a list of replacement rules of the form GRN$v[i]→variablename, indicating, in order the actual GRN variables that represent the v[i] in the Bioinformatics paper. For example, if GRNequations[graph,...,include→ {Y,Z,RAFK, K[3,2]}] is called then GRN$PROTEIN={GRN$v[1]→RAFK,GRN$v[2]→Y,GRN$v[3]→Z,GRN$v[4]→K[3,2]}.

growthExponent

growthExponent calculates and returns the value of the exponent used to determine growth rates, 3 for spheres and 2 for disks.It is determined by the value of the global variable node$shape and can be reset by by using the option shape.

help

Cellerator help can be found in the following ways:
Typing ?symbol will give the Mathematica usage string for symbol, if one is defined;
help[x] or Help[x] is equivalent to ?x or x::usage
createCelleratorReference[] will generate a reference manual.

hexTable

hexTable[r] generates a hemispherical array of 3 dimensional points on a hexagonal-close-packed grid. The argument r is the radius of the grid in cell diameters and hence should be an integer.

hill

hill[options] is an unevaluated function used to specify hill function parameters for reactions of the form A↦B to interpret and other functions. Default options are: vmax→ 1, nhill→ 1, khalf→ 1,basalRate→ 0. Thill is ignored unless multiple non-catalytic reactions are specified for the same product.

The hill function is $r_0+\frac{r_1+v\,CA^n}{k^n+A^n}$ where vmax→ v, nhill→ n, khalf→ k, ,basalRate→ {r0,r1}.

If basalRate is an atom, it is used for r_0 and r_1=0; only one basalRate can be specified for the catalytic version $(A \overset{c}{\to} B)$, in which case the generated hill function is $\frac{r_0+v\,CA^n}{k^n+A^n}$.

If multiple non-catalytic reactions $(A_i \mapsto B)$ are specified form the same product variable (B) they are combined as $r_0+\frac{(r_1+\sum_{i=1}^{p} v_i\,A_i)^n}{k^n+(r_1+\sum_{i=1}^{p} v_i\,A_i)^n}$, where Thill→$T_i$.

See hillFunction for an evaluated hill function.

hillFunction

hillFunction[x,options] returns a hill function of the form
$\frac{v\,x^n}{K^n + x^n}$, where v, n, and K are determined according to options as follows:
vmax→v (default =1)
nhill→n (default=1)
khalf→K (default=1).

hillRatesQ

hillRatesQ[k] returns True if k is a valid rate constant
list for a hill reaction, i.e., it is either null, has precisely the form
hill[options], or is an option list that contains the option type→hill.

holderRules

holderRules is a list of string replacement rules that replace common symbols
like dots and brackets with string place holders; to invert, use releaseRules.

htmlCitation

htmlCitation[string,URL->value,opt] returns an html
formatted citation as string URL. Options: indent->""

htmlModelDescription

Returns html for the <notes> section of html, and optionally generates an html file.
Options are:
rates->{}
citation->string
URL->string
title->string
Greek-> False, if True, greek characters will
be left as greek; if False, they will be translated to their names
description->optional string text to be included
htmlfile->optional file name

icDistance

icDistance[g,{n1,n2}] or icDistance[g,n1,n2] gives the distance (a number)
between the initial conditions for the embedding positions in two nodes in a
graph even if they are not linked.See also distance, linkLength, linkDistance.

identifySBMLVariableParameters

identifySBMLVariableParameters[options]
determines which Cellerator variables are really sbml parameters
Options are
exclude->{var1,var2,..} list of variables to be excluded, defaults are splv and tspl
variables,parameters,species,numberOfCells are as produced by run; normal usage is as follows:
m=organism[..];
r=run[m,run->false,..];
identifySBMLparameters[r,exclude->{...}
identifySBMLparameters is called by writeSBML.

include

include is an option to run[graph,options] and writeSBML[graph,options] that gives a list of specific variables to be included in the grn equations. If the option grn->False then this option is ignored.If include is specified, then the option exclude is ignored.

indexDomain

indexDomain[x] represents a list where each element is either an integer or an indexSet. An indexDomain is instantiated by expandDomain. See also indexSetQ.

indexedSpecieDefinitions

indexedSpecieDefinitions[{variable,list-of-compartment-numbers, list-of-indices}, options] returns a list {domain-SBML, specie-defs-SBML} where domain-SBML is a list of domain definitions and specie-defs-SBML the specie definition foer a single list.
Options are:
compartment→name of enclosing compartment (default: newCompartment)
domain→name of enclosing domain
initialConditions→ complete cellerator initial conditions for indexed variables]
EXAMPLE. Suppose that the variable K is defined in cellerator as K[2,0][j],K[2,1][j],K[2,2][j],K[3,0][j],K[3,1][j], for j = 1, 2,3, meaning that there are separate instantiations of K[2,0],K[2,1],K[2,2],K[3,0],K[3,1] in each of cells 1, 2, 3. Then indexedSpecieDefinitions[{K,{1,2,3},{{2,0},{2,1},{2,2},{3,0},{3,1}}},compartment→ cell, domain→ barney, initialConditions→ {K[2,0][1][0]==201,K[2,0][2][0]==202, K[2,0][3][0]==203,K[2,1][1][0]==211,K[2,1][2][0]==212,K[2,1][3][0]==213,K[2,2][1][0]==221,K[2,2][2][0]==222,K[2,2][3][0]==223,K[3,0][1][0]==301,K[3,0][2][0]==302,K[3,0][3][0]==303,K[3,1][1][0]==311,K[3,1][2][0]==312,K[3,1][3][0]==313}]
would return a list containing two strings of SBML:
{"<domain ...> ... (may be multiple domains here) ...</domain>", "<specie name="K[..domains...]"... />"}

indexSetQ

indexSetQ[x] returns True if x is an indexSet and False otherwise. An indexSet is a a list where each element is either an integer or a list containing integers and indexSets (recursively).

initializeSBMLReactions

initializeSBMLReactions reinitializes the counters and globals
used to generate SBML reactions:$SBMLReactionCounter, $reactionSpecies

initialMass

initialMass is an option for standardNode, organism or meristem
the defines the desired initial mass values. The default is initialMass→1.

For standardNode[...] the format isinitialMass→*value*

For organism[...], initialMass may be specified in any of the following forms:
initialMass→*value*: set mass[i][0]==value for all i
initialmass→{*value1*,*value2*,*value3*,...,*valuej*}: set mass[
1][0]==*value3*, mass[2][0]==*value2*,...,mass[j][0]==*valuej*; If there are
more than j cells in the organism, the remaining masses are set *value1*.
initialMass→{mass[k1]→*value1*,mass[k2]→*value2*,..., mass→*defaultValue*}, sets mass[
k1][0]==*value1*, mass[k2][0]==*value2*,etc., and sets all unspecified masses to
defaultValue. If mass→*defaultValue* is not specified the default reverts to a mass of 1.

For mersitem[...], the format is initialMass→ {CZL1→
1.0, PZL1→ 1.2, CZL2→ 1.20, PZL2→ 1.20, Rib→ 1.85, Floor→ 2.0} and the
values are randomized with a uniform distribution over a range of +/- 15%.

initTimeChecker

initTimeChecker[] initializes the CPU counter used by timeChecker.

intDomain

intDomain[m], intDomain[m,n], intDomain[m,n,p] are Domain objects represents the lists
Range[m], Range[m,n], Range[m,n,p]. Use expandDomain to instantiate these domains.

integerform

integerform[i,n] returns a string of length n
that is padded in front by zeroes. Only really works for integers.

integrateGraph

integrateGraph[graph, step, options] formats and executes the call NDSolve and generates any required plots, and returns a list {*tstart, eventTime, p, eventName, eventData*} where *tstart* is the actual start of integration; *eventTime* is the time at which the first event takes place (which is *tstart+step* if no event occurs); *p* is a pointer to the interpolating function solution, *eventName* is one of the following strings: 'normal-exit', 'link-broken', or 'cell-divides'; *eventData* is either an empty list, the list of links that are broken (integer link numbers) at *eventTime*, or the list of cells that divide (integer node numbers) at *eventTime*.

Options are:
rules→ {}: list of valid Mathematica Replacement rules to be applied before integration.
plotVariables→ None(default),All or {*var1,*
var2, ...}, list of variables to plot immediately after call to NDSolve
plotTrajectories→ False,if True, generate a 3 dimensional plot of the trajectories of each node
plotCoordinates→ False, if True, a separate plot of all the x,y,z coordinates of each node will be plotted; note that if plotTrajectories→ All, these plots will be redundant.
run→True, if False the simulation will not be performed, but the differential equations and initial conditions that would have been used will still be generated. Use this option if you just want to generate the grn equations and look at them.
shape→sphere,disk: determines if mass growth is proportional to square or cube of radius. Overrides current global value of node$shape; normally it is unnecessary to set this variable because it is preset by organism, but if some intervening calculations have be performed between creation of the organism and the run command.
useStoppingTest→ False, if True, the undocumented NDSolve option StoppingTest will be used to check for cell division or link breakage requirements; this may be be faster. If False, NDSolve is run for the entire duration step. In either case Cellerator interpolates to find the precise time in which an event occurs; the only difference is the duration of the NDSolve run, which may, or may not, affect the accuracy of the interpolating Function.

evaluate-> True - turn to falst to inhibit evaluation of Graph
Parameters such as the T matrices, and in stead return a set of equations with just the names of the matrices. This option is ignored unless run-> False

In addition, any valid NDSolve option may also be used.
Note:integrateGraph is called by runStep and is not intended for the end user.

intermediateCompound

intermediateCompound[x,y] returns a list containing the name (as a symbol) of the compound formed by joining x and y.See also compoundName, intermediateCompoundName, dash, complexLeft, complexRight

intermediateCompoundName

intermediateCompound[x,y] returns the name (as a symbol) of the compound formed by joining x and y.See also compoundName, intermediateCompound, dash

interpret

```
interpret[{reaction,reaction,...}] will generate a
  system of differential equations corresponding the specified reactions.

interpret[{{reaction,rates},{reaction,rates},...}] will interpret the reactions using
  the specified rate constants, which may either be a sequence of rate constants or
  an unevaluated function (only for certain types of reactions) such as hill, GRN,
  or NHCA; any hybrid of reaction or {reaction, rates} may be combinded together.

interpret[reactionList,frozen] will not generate
  differential equations for each of the variables listed in frozen.

interpret[list,options] can also be specified where
  list is the first argument of any of the above versions and options are
frozen-> {} (same function as variable frozen)
showODEs→False, do not print the odes to the screen
showODEs→True, print the ODEs to the screen one line per equation
showODEs→List, display the ODEs on the screen as a standard mathematica list
showODEs is normally not intended to be used in manual calls to
  interpret but may be useful if interpret is being called by a wrapper function.

examples:
(1)All rates specified:
                x
interpret[{{∅ → PX,k1},
  {PX→∅,k2},{PY↦Z,hill[vmax→  α1, nhill→ n]}}]

(2)no rates specified:
interpret[{x→y, u→v}]

(3)hybrid (note that each reaction must be enclosed in curly brackets:
interpret[{{A→B,k1}, {C→F}}]
```

interpretArrowType

interpretArrowType[{reaction,rates}] returns a string indicating the type of Cellerator Arrow
interpretArrowType[reaction] will also
  return the same integer Except for transcriptional right-t arrows

See also arrowType for an integer instead of a string.

The return values are:
for ∅→x or {∅→x} or {∅→x,k} the return value is "creation"
for x→∅ or {x→∅} or {x→∅, k} the return value is "annihilation"
for x→y or {x→y} or {x→y, k} the return value is "conversion (unidirectional)"
for x⇄y or {x⇄y} or {x⇄y, k,...} the return value is "conversion (bidirectional)"
for $x \overset{z}{\to} y$ or $\{x \overset{z}{\to} y, k\}$ the return value
  is "catalytic mass action without any intermedidate complex"
for $x \underset{}{\overset{fwd}{\rightleftarrows}} y$ or $\{x \overset{fwd}{\rightleftarrows} y, k,...\}$ the return value is "catalytic
  mass action with substrate/Enzyme complex"
for $x \underset{rev}{\overset{fwd}{\rightleftarrows}} y$ or $\{x \underset{rev}{\overset{fwd}{\rightleftarrows}} y, k,...\}$ the return value is "catalytic mass
  acton with substrate/Enzyme complex, bidirectional"
for $x \overset{z}{\mapsto} y$ or $\{x \overset{z}{\mapsto} y, ...\}$ the return value is "catalytic (saturating hill function)"
for $x \underset{b}{\overset{a}{\Longrightarrow}} y$ or $\{x \underset{b}{\overset{a}{\Longrightarrow}} y, k, ...\}$ the return value is "GMWC"
for {x↦y,hill[stuff]} or {x↦y,type→hill, ...} the return value is "Hill"
for {x↦y,GRN[stuff]} or {x↦y,type→GRN, ...} the return value is "GRN"
for {x↦y,NHCA[stuff] or {x↦y,type→NHCA, ...}} the return value is "NHCA"
for {x↦y,SSystem[stuff]} the return value is "SSystem"
for $x \overset{z}{\rightleftharpoons} y$ or $\{x \overset{z}{\rightleftharpoons} y, k,...\}$ the return value
  is "catalytic Mass Action with substrate/Enzyme and product/Enzyme complex"
for tanything else the return value is "unknown"

interpretedSystemQ

interpretedSystemQ[x] returns True if x is syntacticaly equivalent to
  the output of interpret, i.e., it has the form {{ode,ode,...}, {var,var,...}}
  where ode is a Mathematica differential equation and var is a variable name.

interpretReaction

interpretReaction[r] returns a string indicating the type of reaction.

interpretTree

interpretTree[tree] returns the tree as an option List.

intersects

```
intersects[options] determines if a line segment defined by two points
  intersects a circle when the line segment is extended to a line; options are: :
center→{0,0}
radius→1
endPoints→{{x1,y1},{x2,y2}}
plot->False
See also: findIntersection, blocks.
```

intersperse

```
intersperse[list1, list2] returns a list with the elements interspersed.
  Both lists must be of the same length. Example: intersperse[Range[5],{A,B,C,D,
  F}] returns {1,A,2,B,3,C,4,D,5,F}. See also multsperceBefore, multisperceAfter.
```

isAPartialRateList

```
isAPartialRateList[x] returns True if x is a list composed of a combination
  of Cellerator reactions with and without rates, or if it is a rate list.
```

isARateList

```
isARateList[x] returns True if x is a list of
  Celllerator reactions including rate  constants, and False otherwiase.
```

isAReaction

```
isAReaction[x] returns True if x is a Cellerator Reaction and False otherwise.
```

isAReactionList

```
isAReactionList[x] returns True if x is a list of Cellerator
  Reactions and False otherwise. Will return False if any of the reactions
  have rate constants specified. See isARateList, isAPartialRateList
```

isFlat

```
isFlat[g] returns True if graphDomain g is 2-dimensional AND only in the x-y plane.
```

jacobianMatrix

jacobianMatrix[sys] returns the Jacobian of sys,
where sys is either a signal transduction network (interpretable); an
interpreted signal transduction network; or a list of first order ODEs

jog

jog is another name for predictTransferFunction]

LambdaApproximation

LambdaApproximation[Km,Kcat,lambda] gives an approximation
to the three rate constants in the catalytic reaction $A \overset{Enz}{\Longrightarrow} B$ or $A \overset{Enz}{\rightleftarrows} B$.
$\{A \overset{Enz}{\Longrightarrow} B$, MM[LambdaApproximation[Km, Kcat, Λ]]} returns the value $\{A \overset{Enz}{\Longrightarrow} B$, Λ*
Kcat/Km, (Λ-1)Kcat, Kcat}, which simplifies to $\{\{A \overset{EnZ}{\Longrightarrow} B$, MM[Km,Kcat]}.
$\{A \overset{Enz}{\rightleftarrows} B$,LambdaApproximation[Km,Kcat,Λ]} returns the value $\{A \overset{Enz}{\rightleftarrows} B$, Λ*Kcat/Km, (Λ-1)Kcat, Kcat}

lambdaMatrixMode

lambdaMatrixMode is an option to run (graph/organism) that is used to select
the uppercase lambda in the GRN equations. This option is ignored if grn→False.
Values are case insensitive and may be specified as either symbols or strings, and are:
Chord: chord length of overlapping circles; overlap is specified with the option overlapFraction→
0.1 (Default value) as a fraction of total distance between two cells, e.g., a=(1+
overlapFraction)*r*m1$^{1/3}$/(m1$^{1/3}$+m2$^{1/3}$). The formula is $\Lambda^2=(4a^2b^2-(r^2-a^2-b^2)^2)/(4r^2)$
Connection: use the connection matrix, ie., ∧(i,j)=1 if nodes
i and j are connected and 0 if they are not connected.
Default: use the length of a line that is tangent to the two circles and has length
subtended by the convex hull of the two cells. Relative radii are taken as proportional
to the cube root of the mass, e.g., a=r*m1$^{1/3}$/(m1$^{1/3}$+m2$^{1/3}$).The formula is $\Lambda=2\sqrt{ab}$.
Hexagonal: average of the side lengths of circumscribed hexagons of the two
cells. Relative radii are taken as proportional to the cube root of the
mass (same as in default mode). The type of average is specified as average→
geometric (default, e.g., $\sqrt{ab}$) or average→arithmetic (e.g., (a+b)/2)
Shadow: same as default.

leavesIn

leavesIn[tree] returns the number of leaves in a tree

leftBracketHolder

leftBracketHolder is a global symbol used during Cellerator
output translation to hold the desired text symbol used for a left bracket.

leftParenHolder

leftParenHolder is a global symbol used during Cellerator output
translation to hold the desired text symbol used for a left parenthesis.

lhsShortRightArrow

lhsShortRightArrow[a→b] returns a list of the
reactants a, e.g., lhsShortRightArrow[a+b+c→x+y+z] returns {a,b,c}

limitedRandomNumber

limitedRandomNumber[*size*] returns a random number between -*size* and +*size*

lineageTree

A lineage tree is a binary tree containing the integers 1,...,n,in the leaves,representing
the birth order of n cells.Whenver a new cell is born,it is assigned number n+1,and
the leaf representing its parent is split into two branches,with the parent on the
left branch,and leaf n+1 on the right branch.Exception:the lineage tree might not be
binary in the root node,because we can start with>2 cells whose lineages are unknown.
lineageTree[] returns tree[1] (a tree with a single node,the number 1).
lineageTree[t,parent→p] modifies the lineage tree t,with a new node representing the child of
parent p,which means that the leaf representing p in t is split into two branches,with p on the
left,and n+1 on the right,where n is the number of leaves in t.The modified tree is returned.

lineageTreeQ

lineageTreeQ[t] returns True only when t is a properly formatted lineage tree

link

link[g,n] is the same as linkDomain[g,n] where g is a graphDomain and n an integer.

linkData

```
linkData[linkDomain[...]] returns the link data
linkData[graphDomain[...],n] returns the link data for the nth link
```

linkDistance

```
linkDistance[g,n] gives the length of link n in graph g using
  the initial conditions on the embeddings of the two ends of the linke (
  i.e., it returns a number). See also linkLength, distance, icDistance.
```

linkDistances

```
linkDistances[g] returns an ordered list of the lengths of thelinks in
  g using the initial conditions of the embeddings of the two ends of the links.
```

linkDomain

```
linkDomain[options] represent a linkDomain in a graph
linkDomain[linkDomain[...], options] returns a modified linkDomain

Options are:
linkFromTo→{n1,n2} (integer node numbers)
data→{}
type→{}
spring→springDomain[]

linkDomain[graphDomain[...],n] returns the nth linkDomain from the graph;
see also: linkedNodes, linkData, linkType,
  linkSpring, linkDistance, linkLength, springConstant, springLength
```

linkedNodes

```
linkedNodes[linkDomain[...]] returns the pair of
  integer node numbers {n1, n2} representing the nodes that are linked
linkedNodes[graphDomain[...],n] returns list of pairs of integer node numbers for all nodes
  that are linked to node n in either direction, {{n, n1}, {n, n2},..., {n3,n}, {n4,n},...}.
linkedNodes[grpahDomain[...]] retursn a list of all node
  pairs that are linked {{n1,n2},{n1,n2},{n1,n2},...}
 See also linkNodePair
```

linkLength

```
linkLength[g,n] returns the length of link n in graph g as
  a formula based on the variables that represent the enmbedding variables
  of the two ends of the link. See also linkDistance, distance,icDistance.
```

linkNodePair

linkNodePair[graphDomain[...],n] returns the node pair {n1,n2} for the nth link of the graph

linksIn

linksIn[g] returns an integer, the number of links in g.

linkSpring

linkSpring[linkDomain[...]] returns the Spring
linkData[graphDomain[...],n] returns the Spring for the nth link

linkType

linkType[linkDomain[...]] returns the link type
linkType[graphDomain[...],n] returns the link type for the nth link

localDensity

localDensity[g,n] returns the local density of a graph at a given node based on the node's distance from its neighbors, assuming that it is touching all of its neighbors. The local density is the average over all the neighbors of node n of (mass^1/3+neighborsMass^1/3)/((4/3)*Pi*separation). If a node has no neighbors a density of 1 is returned.

lowLevel

lowLevel[reactions] counts the number of low level reactions in reactions.
To display the reactions, see lowLevelReactions or lowLevelReactionTable.

lowLevelReactions

lowLevelReactions[circuit] gives a list of the low-level reactions in circuit.
circuit has the same format as the input to interpret. See also LowLevelReactionTable

lowLevelReactionTable

lowLevelReactionTable[circuit] lists the low-level reactions
as a table rather than a Mathematica list. See also lowLevelReactions.

makeConditional

```
makeConditional[{condition,result,default}] returns
  the SBML level 2 conditional string "condition ? result : (default)".
makeConditional[{condition,result}] returns the SBML level
  2 conditional string "condition ? result : (0)".
 The default value is always enclosed in parenthesis so that
  conditionals can be easily (and correctly) nested.
```

makeConditionals

```
makeConditionals[{ {c1,r1}, {c2,r2},...,{cn,rn,d} }] returns the nested SBML level
  two conditional "c1?r1:(c2?r2:(c3?r3:...(cn?rn:(d))))". Default values are ignored
  except in the last conditional. Thus makeConditionals[{{c1,r1,d1},{c2,r2,d2},{c3,r3,
  d3},{c4,r4,d4}}] and makeConditionals[{{c1,r1},{c2,r2},{c3,r3},{c4,r4,d4}}] both
  return "c1?r1:(c2?r2:(c3?r3:(c4?r4:(d4))))".Warning: No error checking is performed.
```

makeIndexed

```
makeIndexed[x,i] returns x[i]
```

makeList

```
makeList[expression] replaces all function f[x] by lists
  {f, lists-in-x} where lists-in-x is the possibly nested list of arguments
  where each function call is also replaced by a list, to all depths.
```

makeMovie

```
makeMovie[s, {tstart, tend, dt}] creates a move from a list of graphs and
  solutions generated by runFixedInterval. s is the list produced by runFixedInterval.
```

MAPKCascade

```
MAPKCascade[options] generates the biochemical reactions
  for an n-stage MAP-Kinase phosphorylation cascade including scaffold

If no options are specified, a 3-stage cascade in solution without phosphatases will be returned.

Options are:

scaffoldName→S (default: None) name of scaffold
signal→RAFK (default value: RAFK) name of signal that starts the cascade
phosphatase→{p1,p2,...,pn} If specified, the reaction schema will include phosphatases
  in solution; if not specified (default), no phosphatase reactions will be generated.
stages→{a[1],a[2],...,a[n]} (default:{2,2,1}) maximum
  number of phosphorylations for K[1],K[2],...
scaffoldBindingRates→ {kon,koff,kpon,kpoff} optional names for binding/
  release of unphosphorylated/phosphorylated kinases to scaffold; if any
  names are omitted, each binding reaction will have a separate rate constant
scaffoldPhosphrylationRates→{{k11,k12,...},{k21,k22,...},...}, optional names
  of rate constants, where kij is the rate for the jth phosphorylation in the
  ith slot; names for the rate constants will be assigned if not specified.
scaffoldSignalRates→{a,d,k}, optional names of rate constants to use for the signal
  reaction on the scaffold; names for the rate constants will be assigned if not specified.
solutionSignalRates→{a1,d1,k1,a2,d2,k2}, optional names of rate constants to use for
  the signal reaction in solution; the first three rates are for phosphorylation (
  action of kinase), and the second three rates are for de-phosphorylation (action
  of phosphatase); names for the rate constants will be assigned if not specified.
solutionPhosphorylationRates→{list, list, list, ...} are the rates for phosphorylation in
  solution; each list contains up to six rates, the first three for phosphorylation and the
  second three for dephoshporylation; the lists are interpreted in the following order:
    1st phosphorylation of K[1],
    2nd phosphorylation of K[1],
    ...,
    a[1]'st phosphorylation of K[1],
    1st phosphorylation of K[2],
    ...
rateConstants→{{reaction,rates},{reaction,rates},...}, an
  optional list of rate constants to use for specific reactions in the cascade.
```

mapTimeVariable

```
mapTimeVariable[{x1,x2,...},t] returns a list{x1[t],x2[t],...}
```

massGrowth

```
massGrowth is an Option for mass$formula that selects a particular
  massGrowth model. Both run and writeSBML pass this option to mass$formula.
massGrowth→exponential: means dm/dt=massGrowthRate*m
massGrowth→logistic: means dmdt=massGrowthRate*m(1-m/steadyStateMass),
massGrowth→None or False: means dm/dt=0
```

massGrowthRate

massGrowthRate is an option for standardNode
  and organism as well as a subfield of a nodeDomain's nodeData.
In standardNode or the nodeData field it should be an atom that gives the desired mass growth
  rate to be used in exponential or logistic growth models of the corresponding node's mass.
In organism it can be any of the following:
massGrowthRate→*value*, use value for every node
massGrowthRate→{*value1*, *value2*,...}, use *value1* for
  node1, *value2* for node2, etc. Extra values are ignored. If not enough
  values are provided then the first value in the list is used as a default.
massGrowthRate→{mu[i1]→*val1*, mu[i2]→*val2*,...,default→*defaultvalue*, use the specified values
  for the specified growth rates, and set all of the others to *defaultvalue* (optional, default
  value is 0.065). The option mu→*defaultvalue* can be used in place of default→*defaultvalue*

For mersitem, the format is massGrowthRate→ {CZL1→ 0.065, PZL1→0.065, CZL2→0.065, PZL2→ 0.065,
  Rib→ 0.065, Floor→ 0.065}, and the values are randomized with a uniform distribution by +/- 15%.

mass$formula

mass$formula[options] returns a Mathematica pure function of a single
  argument that represent the rate of change of mass as a function of time.
  To execute the function one would time enter mass$formula[options][mass].
Options are:
massGrowth→exponential, logistic, none (see massGrowth).
massGrowthRate→0.065
steadyStateMass→2.
Example:
mass$formula[steadyStateMass→ 99.3,massGrowth→ logistic, massGrowthRate→ 0.01][m]
returns: 0.01*(1 - m/5)*m

mergeIndices

mergeIndices[x[i,...][j,...]...[k,...] returns a symbol with a single index list x[i,...,j,...,
  k,,,], for example, mergeIndices[a[1,2,3,4,5][3][5,12]] returns a[1,2,3,4,5,3,5,12]

mergeModelDomains

mergeModelDomains[m1, m2] or mergeModelDomain[{m1,m2}]
  creates a single modelDomain by merging the two modelDomains m1 and m2.

meristem

```
meristem[n,options] generates a simulated meristem of hemispherical radius n cells.
Options are:
grid->"spherical" (default, hemisphere of radius n);  "fcc" (face-centered cubic
  array circumscribed by hemisphere of radius n); "hcp" (hexagonal close-packed array
  circumscribe by hemisphere of radius n); "circular" (2 Dimensional semi-circular array
  of radius n);  "flat" (2 Dimensional triangular array inscribed in semi-circle of
  radius n).Values are case-insensitive and may (but need not) be enclosed in quotes.
dmax->2.0, maximum allowed link length (only used for circular or spherical)
CZAngle->45, angle of central zone, degrees
mersitemGrowthRates→ {CZ→ 1, Rib→ 5, PZ→ 10,Floor→ 0}, how fast the mitotic oscillator works.
initialMass→ {CZL1→ 1.0, PZL1→ 1.2, CZL2→ 1.20,
  PZL2→ 1.20, Rib→ 1.85, Floor→ 2.0}, initial conditions on cell masses
steadyStateMass→ {CZL1→ 2.0, PZL1→2.4, CZL2→2.4, PZL2→ 2.4, Rib→ 3.7, Floor→ 4.0}
The following options for organism[] can also be used: such as
  mitoticOscillator, force, shape,species,speciesIC,speciesodes,speciesReactions
```

mitosis

```
mitosis is an option for run[graph,...] that indicates whether or not cell division is allowed.
  If mitosis→True (default), Cellerator will check for cell division. If False, checking will
  be inhibited. If there is no mitoticOsillator (e.g., the organism was created with option
  division->False) then even if mitosis->True cell division will not occur. This option is
  only intended to be used  to inhibit cell division when the oscillator is otherwise present
  and working; it will not create an oscillator in each cell if one is not already present.
```

mitoticOscillator

mitoticOscillator is an option for organism and standardNode
that indicates which mitotitic oscillator should be used. The option
should be used as mitoticOscillator→{option→value,option→value,...}.

Valid options are:
name→Goldbeter,Tyson2, Tyson6, Norel,User- select which model to use
rate->1, rate to multiply right hand side of all
differential equations, thereby changing the frequency of the oscillations.
threshold→see model defaults - threshold to use to indicate cell division
thresholdVariable→variablename - name of
variable in the model to test against threshold (under construction)
initialConditions→see model defaults - initial conditions to use for the model variables
variables→see model defaults - names of model variables to use

Specific model defaults and options:
Gardner: variables→{C,M,X,Y,Z}, thresholdVariable→M, threshold→
.7, α→ 0.1, a1→ 0.5, a2→ 0.5, d1→ 0.05, vs→ 0, also any of the parameters in
the Goldbeter model. Reference: T. S. Gardner et. al., PNAS 95:14190 (1998).
Goldbeter: variables→{C,M,X}, thresholdVariable→M, threshold→0.7, initialConditions→
{Random[0,0.3],0,0},rate→1 (period is approximately 20/rate), K1→ 0.005, K2→0.005,
K3→ 0.005,K4→0.005, vi→0.025, Kc→ 0.3, kd→ 0.01, VM1→ 3, V2→ 1.5, VM3→1,
V4→ 0.5, vd→ 0.25, Kd→ 0.02. Reference: A. Goldbeter, PNAS, 88:9107 (1991).
Norel: e→3.5, f→1, g→10, i→1.2. Reference: R. Norel and Z. Agur, Science, 251:1076 (1991).
Tyson, Tyson2, or Tyson2Variable: variables→{pMPF,aMPF}, thresholdVariable→aMPF, threshold→
0.1, initialConditions{Random[0,0.3}, 0}, rate→1,κ→0.015,k6→1,k4→ 180,α→(0.018/180).
Reference: J. J. Tyson, PNAS, 88:7328 (1991). aMPF is the same as u in the paper, and
pMPF is the same as the difference $z=v-u$, where v and u are as defined in the paper.
Tyson6 or Tyson6Variable: variables→{C2,CP,M,pM,Y,YP}, initialConditions-> {0, Random[0,0.3],
0,0,0,0}, k1aa→ .015,k2→ 0, k3→ 200, k4→ 180, k4prime→ 0.018, k5notP→ 0, k6→ 1, k7→ 0.6,
k8notP→10^6, k9→ 1000}Reference: J. J. Tyson, PNAS, 88:7328 (1991) (same paper as 2-variable)
User: to create a user-defined model. Any option allowed to createUserModel may be specified.

MM

MM is an unistantiated function that holds data for a Michaelis-Menten reaction.
{a⟹b,MM[KD,V]} produces $\frac{db}{dt}=-\frac{da}{dt}=\frac{V*a}{KD+a}$
{a $\overset{c}{\Longrightarrow}$ b, MM[KD, v]} produces $\frac{db}{dt}=-\frac{da}{dt}=\frac{v*C*a}{KD+a}$ (KD+a)
{a $\overset{c}{\Longrightarrow}$ b, MM[k1,k2,k3]} is the same as {a $\overset{c}{\Longrightarrow}$ b, MM[(k2+k3)/k1, k3]}
{a⟺b,MM[KD,V]} is equivalent to the pair of reactions {a⟹b,MM[KD,V]}, {b⟹a,MM[KD,V]}}
{a⟺b,MM[KDa,Va], MM[KDb, Vb]} is equivalent
to the pair of reactions {a⟹b,MM[KD,Vb]}, {b⟹a,MM[KDb,V]}}
{ $\underset{EReverse}{\overset{EForward}{a\Longleftrightarrow b}}$, MM[list1],MM[list2]}, { $\overset{EForward}{a\Longrightarrow b}$, MM[list1 arguments], { $\overset{EReverse}{b\Longrightarrow a}$, MM[list2 arguments]}}

model

model is an option for writeSBML that specifies the name be used for
the SBML model.If unspecified,a unique model name is generated automatically.

modelDomain

modelDomain[*options*] represents a model within a cell;
modelDomain[modelDomain[...], *options*]
  modifies the specified model domain as per the specified options.

options are:
molecules→ {}, a list of variable names representing molecular concentrations;
odes→ {}, a list of differential equations that molecules obeys
ic→ {} or {*value1, value2, ...*} or {*molecule*[*t0*]=*value0, molecule2*[*t0*]=
  value2,...}, alist of IC's for the molecules, unspecified ics default to zero;
time→ 0, value at which ic's are applied; default is zero

Operations on modelDomains include:modelODEs[m],modelMOLECULES[m],modelIC[m]

modelDomains

modelDomains[nodeDomain[...] gives a list of all the modelDomains in the specified nodeDomain.

modelIC

modelIC[m] gives a list of the initial conditions in m.

modelMOLECULES

modelMOLECULESs[m] gives a list of the molecules in m.

modelODEs

modelODEs[m] gives a list of the differential equations in m.

modeltime

modeltime[m] gives the value of the *time* field of m.

modifyNode

modifyNode[n, options] returns a modified node based on
  options, where options are any valid options for either a nodeDomain,
  an embeddingDomain, or a modelDomain, and are applied in that sequence.

movieKey

movieKey[nodeColors→*color-list*] returns a list of the form
{{- Graphics -, *nodetype*}, {- Graphics -, *nodetype*},...} that is compatible
with ShowLegend. *color-list* is the output of generateColorTable or a list
of the form {{*nodetype*, *colorfunction*}, {*nodetype*, *colorfunction*},...}.
Note:movieKey is not an end-user function. It is called
by showGraph, showFlatGraph, and makeMovie.

multiplot

multiplot[x] is plots a variable from a cellerator run sequence that may be composed of
several consecutive,overlapping numerical sequences. Users should not invoke multiplot
directly but should use runPlot. The plot option in run produces a call to multiplot.

multisperseAfter

multisperseAfter[list1, {list2a, list2b, list2c, ...}] intersperses each of list 2a, 2b, ...
into list 1 See also multisperseBefore,stringMultisperseAfter. Example.multisperseAfter[
Range[5],{{A,B,C,D,F},{P,Q,R,S,T}, {alpha, beta, gamma, delta, epsilon}}] returns {{A,
1,B,2,C,3,D,4,F,5},{P,1,Q,2,R,3,S,4,T,5},{alpha,1,beta,2,gamma,3,delta,4,epsilon,5}}.

multisperseBefore

multisperseBefore[list1, {list2a, list2b, list2c, ...}] intersperses list 1
into each of list 2a, 2b, ... See also multisperseAfter,stringMulitsperseAfter,
stringMultisperseBefore. Example.multisperseBefore[Range[5],{{A,B,C,D,F},
{P,Q,R,S,T}, {alpha, beta, gamma, delta, epsilon}}] returns {{1,A,2,B,3,C,
4,D,5,F},{1,P,2,Q,3,R,4,S,5,T},{1,alpha,2,beta,3,gamma,4,delta,5,epsilon}}.

myPadRight

myPadRight is a work-around a bug in PadRight in Mathematica version 4.1.5.
myPadRight[list,n] implements PadRight[list,n]
myPadRight[list,n,x] implements PadRight[list,n,x]
myPadRight[list,n,{x}] implements PadRight[list,n,{x}]
other versions of PadRight are not implemented and may lead to error.

neighborFunctionSBML

neighborFunctionSBML[cm, options] generates the level-2 SBML for a function that is true when the
square matrix cm[[i,j]] is nonzero, and is false when the i,j element of cm is zero.

neighbors

neighbors[g,n] returns a list of the (node numbers of the) neighbors of node n in graph g.
neighbors[g] returns a list {{1, neighbors of 1}, {2, neighbors of 2},...} for the whole graph

newCompartment

newCompartment will return a string "Compartment*nnn*" where *nnn* is the current
value of $CompartmentNumber, which is then incremented. See also resetSBML.

newDomain

newDomain will return a string "Domain*nnn*" where *nnn* is the current value of
$DomainNumber, which is then incremented. The string "Domain" can be replaced with
any other string by setting the value of $DomainName. The number of digits (default
is 3) is set by $DomainDigits. For example, if $DomainName="foo" and $DomainDigits=5,
the names generated will be "foo00001","foo00002","foo00003",... See also resetSBML.

newModel

newModel will return a string "Model*nnn*" where *nnn* is the
current value of $ModelNumber, which is then incremented. See also resetSBML

newRateConstant

newRateConstant[x] returns a rate constant xnnn where nnn is the value of
$rateConstantNumber, and is incremented during interpret when rate constants are defined.

news

The news command is no longer supported by
Cellerator. To learn about a particular command, enter ?command

NHCA

NHCA[options] is an unevaluated function used to specify
parameters for reactions of the form A↦B to interpret and other functions.
Default options are TPLUS→ 1, TMINUS→ 1, nNHCA→ 1, kNHCA→ 1,mNHCA→ 1,TNHCA→ {},
vmax-> 1, NHCAmode→0; if TNHCA is specifed then both TPLUS and TMINUS are ignored.

The function generated for the reaction A↦B is $\frac{v\,(1+T^{+}\,A^{n})^{m}}{k\,(1+T^{-}\,A^{n})^{m}+(1+T^{+}\,A^{n})^{m}}$

where TPLUS→ T^{+}, TMINUS→ T^{-}, nNHCA→ n, kNHCA→ k,mNHCA→ m, vmax-> v,

and TNHCA is not specified; and is $\frac{v\,(1+T\Theta\,(T)\,A^{n})^{m}}{k\,(1-T\Theta\,(-T)\,A^{n})^{m}+(1+T\Theta\,(T)\,A^{n})^{m}}$

if TNHCA→T,nNHCA→ n, kNHCA→ k,mNHCA→ m, and Θ(x)=UnitStep[x].

If multiple reactions of the form $A_i \mapsto B$ are specified for the same

product B then function becomes $\frac{v\prod_i (1+T_i^{+}\,A_i^{n_i})^{m}}{k\prod_i (1+T_i^{-}\,A_i^{n_i})^{m}+\prod_i (1+T_i^{+}\,A_i^{n_i})^{m}}$

or $\frac{v\prod_i (1+T_i\,\Theta\,(T_i)\,A_i^{n_i})^{m}}{k\prod_i (1-T_i\,\Theta\,(-T_i)\,A_i^{n_i})^{m}+\prod_i (1+T_i\,\Theta\,(T_i)\,A_i^{n_i})^{m}}$, respectively.

If NHCAmode→1, the equation becomes $\frac{v\,(T^{-}\,A^{n}\,(1+T^{-}\,A^{n})^{m-1}+T^{+}\,A^{n}\,(1+T^{+}\,A^{n})^{m-1})}{k\,(1+T^{-}\,A^{n})^{m}+(1+T^{+}\,A^{n})^{m}}$ and so forth.

NHCARatesQ

NHCARatesQ[k] returns True if k is a valid rate constant
list for a NHCA reaction, i.e., it is either null, has precisely the form
NHCA[options], or is an option list that contains the option type→NHCA.

node

node[g,n] is the same as nodeDomain[g,n] where g is a graphDomain and n an integer.

nodeDomain

nodeDomain[options] represents a node (cell) in a Cellerator Graph.
nodeDomain[*nodeDomain*, *options*] returns the original nodeDomain modified as per *options*.
nodeDomain[g,n] returns the nth node domain in graph g.

Options are:
models→ {} or {*modelDomain1*, *modelDomain2*,...} (a list of models in the node)
embedding→ *embeddingDomain* (embeddingDomain that represents the node geometry)
nodeType→ Cell (or any other string or identifier that is desired)
nodeData→ {birth→ 0,steadyStateMass->2,massGrowthRate-> 0.065}.

To extract data from a nodeDomain see: nodeODEs,
nodeVariables, nodeIC, getNodeType, getNodeTypes,getNodeData, getNodeModels

nodeIC

```
nodeIC[n] returns a list of all the initial conditions
  for all variables in node n (a nodeDomain), including all variables
  in all modelDomains as well as the embedding position variables.
nodeIC[g,n] returns the the initial conditions for all
  variables in node n (an integer) of graph g.
```

nodeModels

```
nodeModels[n] returns a list of the models (modelDomains) in node n.
nodeModels[g,n] returns a list of the models in the nth nodeDomain of graph g.
```

nodeODEs

```
nodeODEs[n] returns a list of all the differential
  equations for all variables in node n (a nodeDomain), including all
  variables in all modelDomains as well as the embedding position variables.
nodeODEs[g,n] returns the differentnial equations for all
  variables in node n (an integer) of graph g.
```

nodesIn

```
nodesIn[g] retursn an integer, the number of nodes in g.
```

nodeVariables

```
nodeVariables[n] returns a list of all the variables in node n (a nodeDomain), including
  all variables in all modelDomains as well as the embedding position variables.
nodeVariables[g,n] returns the variables in node n (an integer) of graph g.
```

node$Shape

```
node$Shape is a global variable that stores the most recently selected value of
  shape->"SPHERE","DISK"; this value is used unless it is overridden by the shape option.
```

nonOptionQ

```
nonOptionQ[x] returns True if x is not an Option and false if x is an Option
```

note

note is an option for writeSBML that specifies the contents of the SBML <notes> field. SBML notes may include any valid xhtml and are encapsulated in the tags <body xmlns="http://www.w3.org/1999/xhtml">...</body> (These tags are added automatically and should not be specified). The following additional information is always automatically included in the SBML note: Cellerator version number, mathematica version number, operating system, date/time file written, and a reference to the cellerator web site.

notListQ

notListQ[x] returns True if x is not a list and False if x is a list.

Now

Now[] prints the curent date and time in the format "February 15, 2002 10:55:22"; compare with dhmsDate.

nullQ

nullQ[x] returns True if x is {}

numberQ

numberQ is the same as NumberQ but also allows +/- Infinity to be a number.

odeCounter

odeCounter counts the number Cellerator ODE pointers that have been allocated

odeQ

odeQ[x] returns True if x is a single differential equation or list of differential equations.

ODEsin

ODEsin[g] returns an integer, the number of differential equations in the graph g.

offsetEmbedding

embeddingDomain[emb, {*value1,value2,value3*}] adds the specified vector of valuess to the initial conditions of the embeddingDomain emb and returns the modified embeddingDomain

offsetNode

offsetNode[*nodeDomain*,{*x,y,z*}] returns a node with the initial conditions moved by the vector {*x, y, z*}. See also putNode.

optionPairs

optionPairs[opt1→val1, opt2→val2, opt3→val3,...] returns a list {{opt1,val1}, {opt2,val2},...}; opt may be either a list or a sequence of rules and/or options.

organism

organism[*options*] creates a Cellerator graph object based on the input provided.

Options are:
embeddings→{{*x1,y1,z1*},{*x2,y2,z2*},...}, or see:
  regularPolygon, rectangularGrid,triangularGrid,semiHexagonalGrid
links→{{n1,n2},{n1,n2},...} or links→All or links-> None or links->
  Cyclic, list of node integer node pairs to be linked or links->r,
  where r is any number, will connect all nodes closer than a distance r
directed→ False, if True, the links are assumed to be directed; if
  False, for every link {n1,n2}, there is also a link {n2, n1}
types→{*nodetype*, *nodetype*, ...} or types→*nodetype* will label every node as *nodetype*
division→{*flag*, *flag*, ...} or division→*flag* (*for all nodes*) where *flag*=True (default) or False
growing→{*flag*, *flag*, ...} or growing→*flag* (*for all nodes*) where *flag*=True (default) or False
force→True, if False, the spring force will not be calculated.
fastForce→True(default), use fastSpringForce to calculate springForce (more efficient
  algorithm) or False, use springForce (more elegant algorithm, using sumOverNeighbors)
initialMass→ initial cell mass (see initialMass)
steadyStateMass→ steady state cell masses (see steadyStateMass)
massGrowthRate→cell mass growth rates (see massGrowthRate)
mitotocOscillator→ {*option1, option2,...*} options
  to be use to generate the mitotic oscillator; see mitoticOscillator
cellDivisionMassThreshold is passed through to standard Node (should be an atomic value)

periods→*value* (default=10) or {*period1, period2,...*} list of cell division periods for the
  mitotic oscillator. The actual period may be different if any mitotic oscillator besides
  Goldbeter is used, or if the Goldbeter parameter values are modified. This option overrides
  anything specified by *rates* option. If only one period value is specified, all nodes have
  the same period. If fewer periods are specified than the number of nodes the value of
  period1 is used for the unspecified nodes. This option is ignored unless rates->"Periods"
rates→1 (default) or *value* or {*rate1,rate2,...*} list of cell division rates for the mitotic
  oscillator. This option is overridden by the *periods* option if rates->"Periods". If only
  one *value* is specified every cell has the same ratewith default parameters the period is
  20/*rate*. If too few rates are specified the unspecified nodes have rates given by *rate1*.
shape->Sphere,Disk - determines whether mass is proportional to the square
  or the cume of the radius. Overrides current global value of node$Shape.
species→{} (default) for no additional species to be added to each cell; or species→
  speciesname to have a single species called *speciesname* in every cell; or species→{*sp1,*
  sp2,...} to have a set of species *sp1,sp2, ...* in every cell; or species→{{*sp1,sp2,...*},
  {*sp21,sp22,...*},...} to have *sp1,sp2,..* in cell 1, *sp21,sp22,...* in cell 2, etc.
speciesIC→{}, *value*, {*value1*, *value2*, ...}, or {{*value11,value12,...*}, {*value21,value22,...*},...}
  gives initial conditions for the variables given by option species.speciesIC→{} sets all
  initial conditions to zero. speciesIC→{*value1*, *value2*, ...} sets the initial condition of
  first variable in each cell to *value1*, the second variable in each cell to *value2*, etc.
  speciesIC→{{*value11,value12,...*}, {*value21,value22,...*},...} uses {*value11*, *value12,...*}
  for the first cell, {*value21*, *value22,...*} for the second cell, etc. If there are fewer
  ics than variables in any given cell the remaining variables are initially set to zero.
speciesodes-> {} (default); right hand side of species odes; form is {*rhs1,rhs2,...*} to
  use *rhs1* for the first species in every cell, *rhs2* for the second species in every
  cell, etc., in every cell; or {{*rhs11,rhs12,...*},{*rhs21,rhs22,...*}} to use *rhs11*
  for the first species in cell1, *rhs12* for the second species in cell1, *rhs21* for
  the first species in cell 2, *rhs22* for the second species in cell 2, etc. If fewer
  sets of odes than nodes are specified the remaining cells clone the equations from
  cell 1. The odes for each cell must be specified in the same order as the species (
  in option *species*) for that cell. A null list for any cell means the right hand
  sides of all odes for that particular cell are zero. If any cell has fewer odes
  than species the unspecified species in that cell have rhs odes that are zero. The
  right hand sides of the odes specified by this option are added to any right hand
  sides of odes produced by *speciesReactions* or grn equations produced at run time.
speciesReactions→list of reactions to be used in addition to and odes specified by
  speciesodes or generated by grn equations.speciesReactions→{} means there are no additional

reactions. speciesReactions→{{*reaction1*,*k1*,*k2*,..},{*reaction2*,*k3*,*k4*,..},...} gives a list of reactions to be used in every cell. speciesReactions→{{{*reaction1*,*k1*,*k2*,..},{*reaction2*, *k3*,*k4*,..},...}, {{*reaction3*,*k5*,*k6*,...},{*reaction4*, *k7*,*k8*,...}},...} means *reaction1* and *reaction2* are used in cell 1, *reaction3* and reaction4 are used in cell 2, and so forth. Any cells with {} have no additional reactions. If fewer reaction lists than cells are given the remaining cells have no additional reactions. The k1,k2, etc., are the rate constants or variables representing the rate constants to be passed to interpret.
stats→False, if True, prints statistics on CPU use and graph size.

parameterNames

parameterNames[GRNEQ[x]] retrieves a list of the names of parameters in the GRN equations.

parameters

parameters[GRNEQ[options]] retrieves the list of parameters in the GRN equations.

parametricRunPlot

parametricRunPlot[r,{x,y}] plots y[t] as a function of x[t] for the entire cellerator run r.
parametricRunPlot[r,{x,y},{tstart,tend}] plots
y[t] as a function of x[t] for the period from t=tstart to t=tend.
Usage notes:
(1) r, is the output of run (sprint, predicTimeCourse)
(2) not implemented for graph runs.
(3) Any valid option for Plot may be specified.

performEvent

performEvent[graph, solution] returns a modified graph with new initial conditions and starting time based on any events that occur (cell division, split, or end of integration run without any event). The first event that occurs determines the time of new initial conditons. If the first event is link-breakage, the modified graph has those links removed. If the first event is cell division, the modified graph has all of the required new cells and corresponding links added to it. performEvent is not intended for end-user use.

phasePlot

phasePlot[x,...] is another name for parametricRunPlot[x,...]

phasePortrait

```
phasePortrait[stn,{xvar,xmin,xmax,xdelta},{yvar,ymin,ymax,ydelta}] returns a
  phase portrait of the xvar vs yvar based on the Cellerator signal transduction
  network stn, using starting conditions xvar = xmin,xmin+xdelta,...,xmax, yvar=ymin,
  ymin+ydelta,...,ymax. The duration of each run can be set by option timeSpan.
Any valid options for Plot or predictTimeCourse can be used.
If there are additional variables in the system besides xvar and yvar
  they will be set automatically to zero unless they are specified with the option
  initialConditions. Any initialConditions specified for xvar or yvar will be ignored.
For systems with more than 2 variables, it is likely that the trajectories
  will cross because phasePortrait plots only a two-dimensional
  projection of the full phase spase onto the plane {xvar,yvar}.
```

plotColumns

```
plotColumns->n is an option for run[...] that indicates the number of
  columns to used in generating plots (default = 3). Ignored if plotVariables->None.
```

plotCoordinates

```
plotCoordinates->True (Default: False) is an option for run[graph,...] that will produce
  plots of the x,y, and z coordinates for each node at the end of each call to NDSolve.
```

plotGraphRun

```
plotGraphRun[variable, solution, {tstart, tend}, options] plots a variable or
  list of variables against a solution produced by run for an organism. The variable
  solution is the output of run or runFixedInterval. If more than one variable name
  is specified the variables are plotted on a grid. Any options valid for Plot can
  be used. Unindexed variables are fully expanded to include all possible indices.

plotGraphRun[solution,options] plots variables on a grid
Additional Options for this version:
plotVariables-> {var1,var2,...} or All
plotColumns->3 (number of columns in the grid)
plotType->"Linear"(Default) ,"Log" ,"LogLinear" ,"LogLog"
```

plotLinks

```
plotLinks[r], where r is the output of r[graph,...], will plot the lengths of the
  links as a function of time. The lengths of all links will be plotted. Options are
legend→ True, display legend (link number vs. color)
Valid plot options can be specified, e.g., p=plotLinks[r,PlotRange→{{9,15},Automatic}]
```

plotSTNrun

```
plotSTNrun is equivalent to runPlot for signal
  transduction (non graph) run.  See runPlot for allowed syntax and options.
```

plotTrajectories

```
plotTrajectories-> True (default=False) is an option for run[
  graph,...] that will produce plots of the trajectories of the embedding of
  each node to be plotted in a three-dimensional grid after each call to NDSolve.
```

plotVariables

```
plotVariables->{variable, variable,...} is an option for run[...]
  that specifies a list of variables to be plotted after the run is complted.
plotVariables->None(default) means plot nothing.
plotVariables->All means plot everything. Note: using
  plotVariables->all with run[graph,..] may cause various warning messages if
  cell division occurs because it attempts to plot all variables for all times.
```

pointer

```
Cellerator pointers are uninstantiated functions pointer[s], where s
  is a string that gives the name of an actual globally defined data structure.
```

powerToRepeat

```
powerToRepeat[x] replaces expressions of the form x^n with lists {x,x,...}
  of length n. If multiple arguments are supplied they are combined into a list
powerToRepeat[A^3] returns {A,A,A}
powerToRepeat[A,B^2,C^4] returns {A,{B,B},{C,C,C,C}}
```

predictTimeCourse

```
predictTimeCourse[s,options] is a smart wrapper for run. The first
  argument must be a Cellerator™ reaction list, which may contain rate constants.
Any unspecified initial conditions will default to zero.
Any unspecified rate constants or parameters will default to 1.
All options to run are valid. The following additional options are allowed:
initialConditions→{ic1, ic2,...} where each ic has the form variable→
  value or variable==value or variable[t0]==value. If the third form is used, the
  value of t0 will ignored and will be superceded by option tstart or timeSpan.
concise→False, if True, all warning messages that tell you when
  default initial conditions are assumed are suppressed.
rates→{p1→value,p2→value,...} specify value to be used for parameters, where
  p1, p2, ... are parameters such as rate constants and hill-function exponents.
table→None (default),Automatic, or {tableTstart, tableTend, tableTdelta}: generate
  a table of data at the desired output parameter(s) with the requested spacing.
tstart→0, starting time for run. This option is ignored if timespan→{start,duration}
  format is used; however, if timespan→duration is used, this will be the starting time.
verbose→False, if True, detailed listings of initial conditions and rate
  constants will be produced. If concise→True, this option will be supressed.
```

predictTransferFunction

predictTransferFunction[c,options] is an extension of predictTimeCourse that performs multiple runs to calculate an input/output relationship between a single signal (initial condition) and any number of output (predicted concentrations at the end of a run).
Any valid option for predictTimeCourse may be used.If either the signal or the output is unspecified a single run will be performed and the interpolating functions returned. Otherwise the interpolating functions will not be returned and the ouputs will be evaluated at the end of each run and a table of values will be returned.

Additional Options include:

output→ None (default), variable, or {var1, var2, var3, ...}, a variable or list of variables that will be considered as the output signal.
signal→ None (default) or variable, the name of the input signal.
loop→ {vmin, vmax, dv} - looping parameters. The simulation will be run for signal = vmin, vmin+dv, vmin+2dv,..., vmax
format→ "spreadsheet" (default) or "list". This option is only relevant if there are multiple output variables. If format→"spreadsheet", a spreadsheet-compatible list will be returned; if format→"list" a Graphics`MultipleListPlot-compatible list will be returned, as in the following examples:
plot->False, if True, will plot the transfer function(s) using ListPlot or MultipleListPlot; any valid option for ListPlot that is in the options will be passed to this plot.
function→function defintion: a pure function to apply to the output variable. Example: Suppose you want to get 3x, where x is the name of the output variable. Method 1: used function→(3#)& as an option to predictTransferFunction; Method 2: define a global function f[x_]:= 3x; then pass function→f to predictTransferFunction.
integrate→False, if True, the output variable will be integrated over the entire time course. If a function is defined then the integral of f[x[t]] will be returned.

s=jopredictTransferFunctiong[c,output→{P,Q,R,S},loop→{1,5,1}, timeSpan→{0,500},format→"spreadsheet",signal→ ...] will set s= {{1,P[500],Q[500],R[500],S[500]},{2,P[500],Q[500],R[500],S[500]},{3,P[500],Q[500],R[500],S[500]},{4,P[500],Q[500],R[500],S[500]},{5,P[500],Q[500],R[500],S[500]}}
Export[filename,s,"Table"] will save the spreadsheet as a text file.

s=predictTransferFunction[c,output→{P,Q,R,S},loop→{1,5,1},timeSpan→{0,500},format→"list", signal→ ...] will return a list of the form{{{1,P[500]},{2,P[500]},{3,P[500]},{4,P[500]},{5,P[500]}},{{1,Q[500]},{2,Q[500]},{3,Q[500]},{4,Q[500]},{5,Q[500]}},{{1,R[500]},{2,R[500]},{3,R[500]},{4,R[500]},{5,R[500]}},{{1,S[500]},{2,S[500]},{3,S[500]},{4,S[500]},{5,S[500]}}}
Graphics`MultipleListPlot[s] will plot the results.

prepReactionsForSBML

prepReactionsForSBML[reactions] prepares a reaction list for generation of SBML code. It is not intended to be called by the end user.
The input is a list of reactions compatible with input to interpret. The output is a list of reactions, each of which is compatible with SBMLLevel1ArrowReaction. The main reason for preparation is to combinine multiple transcriptional reactions with the same product into reactions of the form A⊕B⊕C⊕...↦product, as required by SBMLLevel1ArrowReaction.

processParameters

processParameters[parameters,options] will generate an sbml level
  2 string for a set of Cellerator constants, i.e., a set of SBML parameters.
parameters has the form of a list-enclosed option list, e.g., {a[1]-> 1, a[2]->
  3, a[3]-> 4, tau-> 43, T[1,1]-> 10, T[1,2]-> 20, T[2,1]-> 30, T[2,2]-> 40}
options areindent->extra indentation string (default is two tab stops)
The string returned will include a list of
  domain definitions followed by a list of parameter definitions.

processSumsOverDomains

processSumsOverDomains[expression,options] returns a list {newexpr,
  domains} domains is a string of SBML defininig all SBML domains needed by all
  calls to sumOverDomain, and newexpr is a modified version of expression with
  sumOverDomain expressed as sumOverDomain[expression, domainSymbol, domainSymbol,...]

protectAll

protectAll will protect every defined symbol in
  the context Global that does not have an attribute of Temporary.

proteins

proteins[node,options] returns a list of proteins in the the specified nodeDomain.
proteins[graph,n,options] returns a list
  of proteins in node n (integer) of the specified graphDomain
proteins[graph,options] returns a list of all the proteins in the graph.
Options are:
exclude->{variable, variable,...} list of variables to exclude from the list]
include->{variable,variable,...} list of variables to include
  in the list. If the include option is used, the exclude option is ignored.

pulse

pulse[t] returns a unitPulse
pulse[t,magnitude] returns magnitude if -1/2≤t≤1/2, and zero otherwise
pulse[t,magnitude, width] returns
  magnitude if t is within width/2 of the origin and zero otherwise
pulse[t,magnitude, width, tcenter] returns magnitude if tis
  within width/2 of tcenter, and zero otherwise
See also: unitPulse, unitPulseTrain, pulseTrain.

pulseTrain

pulseTrain[t,magnitude,n] delivers a train of n pulses of the given
  magnitude and unit time duration where the first pulse is centered at the origin.
pulseTrain[t,tstart, magnitude, n] delivers a train of n pulses of the given
  magnitude and unit time duration, where the first pulse is centered at tstart
pulseTrain[t, tstart, magnitude, pulseWidth,n] delivers a train of n pulses of
  the given magnitude and pulseWidth, where the first pulse is centered at tstart.
pulseTrain[t, tstart, magnitude, pulseWidth, interval,n] delivers a train of
  n pulses of the given magnitude, pulseWidth, and center-to-center spacing.

putNode

putNode[*nodeDomain*,{*xvalue, yvalue, zvalue*}] returns a node with the embedding
  initial condition modified to {*xvalue, yvalue, zvalue*}. See also offsetNode.

rand

rand returns a random number between 0 and 1

randomEmbedding

randomEmbedding[options] returns an embeddingDomain
  with random initial conditions on the position variables in the (0,1)×
  (0,1)×(0,1) cube. Options are otherwise the same as for embeddingDomain.

randomGraph

randomGraph[n,options] returns a graph with n nodes at randomly placed locations
  in the unit cube in the first quadrant. The nodes are all 'standardNodes.'

Options are:
connected→All,None,Some, or an integer n number of links.
nodeTypes→ {Cell} or a list of desired nodeTypes;
  if the list is not long enough it is padded with its last element.
symmetric→ True, if True, the links are two-way; otherwise they are directed.

randomIC

randomIC[] returns a held expression for a random number between 0 and 1
randomIC[x] returns a held expression for a random number x and 1 (or between 1 and x if x>1]
randomIC[x1,x2] returns a held expression for a random
  number between the smaller of {x1, x2} and the larger of {x1, x2}
Example: randomIC[5,12] returns Hold[Random[Real,{5,12}]]

randomize

randomize[list, fraction] returns list with every element perturbed
by a random number in the range +/- fraction of its original value. If list
is a number, a number is returned; if list is a list, a list is returned.

randomSign

randomSigns returns either 1 or -1, randomly.

rateConstant

rateConstant generates a new rate constant (as a symbol, not a string) of the form kxxx where xxx
is an integer that is incremented by 1 every time rateConstant is called; (xxx continues
to be incremented until a rate constant of the form kxxx has not been previously defined)

rateDB

rateDB is a database that contains definitions of the rate constants needed
by Cellerator. It is generated automatically from the rate constants supplied to
interpret. It is not user-modifiable. To reinitialize the database use setDB[];

reactionType

```
reactionType[reaction] returns an integer indicating the  type of reaction.
  Note that only the reactions are examined, so no distinction is made between
  GRN, hill, and NHCA. Assignments are made according to the following table:
0 unknown
1 simple (one-way)
2 simple (two-way)
3 enzymatic (one-way)
4 enzymatic (two-way)
5 on (phosphorylated)
6 on (non-phosphorylated)
7 off (phosphorylated)
8 off (non-phosphorylated)
9 on/off (phosphorylated)
10 on/off (non-phosphorylated)
11 phosphorylation in scaffold
12 signal reaction (solution)
13 signal reaction (scaffold)
14 hill function (simple)
15 hill function (complex)
16 enzymatic (simplified, one-way, non-saturable)
17 enzymatic (simplified, one-way, saturable)
18 non-enzymatic Cascade
19 unidirectional enzymatic cascade
20 bidirectional enzymatic cascade
21 unidirectional simple cascade
22 Generalized MWC
23 Michaelis-Menten (Enzyme Implicit Form)
24 Michaelis-Menten (Enzyme Explicit Form)
25 enzymatic (mass action, SE and PE complexes)
```

realRandom

```
realRandom[a,b] returns a random real number between a and b
```

recoverSolution

```
recoverSolution retrives the global solutions
  and graphs generated by the most recent call to runFixedInterval.
```

rectangularGrid

```
rectangularGrid[nx,ny] generates an nx by ny rectangular grid of points in the xy plane
  between centered at the origin and with horizontal and vertical spacing of 1-unit.
rectangularGrid[nx,ny,{x1,y1},{x2,y2}] uses (x1,y1) for the upper
  left hand corner and (x2,y2) for the lower right hand corner
```

regularPolygon

```
regularPolygon[n] generates the coordinates of
  a regular polygon centered at the origin with one corner at {1, 0}
regularPolygon[n,radius] generates an n-sided regular polygon
  that can be circumscribed by a circle of radius
regularPolygon[n, radius, offset] rotates the polygon by an angle offset in radians
```

relax

```
relax->True (default) is an option for run[graph,...] that indicates that graph
  relaxation is supposed to be performed when new nodes are added to the graph.
```

relaxGraph

```
relaxGraph[graph,options] relaxes an unrelaxed graph by allowing all springs to relax
  towards their local energy minimum for the requested timespan (duration).  The
  relaxed graph is returned; the only difference between the initial and relaxed graph
  are the values of the initial embeddings. The time remains the same, e.g., zero.
relaxGraph[graph,duration, options] is the same as
  relaxGraph[graph,timeStep→ duration, options].
Options are:
maxdldt→0.1, desired maximum value of dl/dt, where l is the link length
plot→False, if True, make a plot of the intercellular distances as a function of time
timeStep→1, duration of each call to NDSolve
```

releaseRules

```
releaseRules inverts holderRules.
```

removeLink

```
removeLink[g,n1,n2] or removeLink[g,{n1,n2}] returns graph g with
  the link from n1 to n2 removed. See also removeLinkBothways. Options are:
exception->Floor (only allow exception cells to move horizontally).
```

removeLinkBothWays

```
removeLinkBothWays[g,n1,n2] returns graph g
  with links {n1,n2} aned {n2,n1} both removed. See also removeLink.
```

removeLinks

```
removeLinks[g, {{from,to},{from,to},...}] returns a new graph with the specified nodes removed.
```

replaceArrows

replaceArrows[x] returns a string version of x with Cellerator arrows convertged to their ASCII representations, as →,→ become ->, ↦ becomes|->, ⇄ becomes <=>

resetGraphIC

resetGraphIC[g, initialConditions→ {name1→val1, name2→val2,name3→val3,...}] will return a modified graph with the specified initial conditions reset. If namei is unindexed then every occurance of namei[j] will be reset. resetGraphIC invokes resetVariables (for unindexed varialbes) and resetSpecificVariables (for the indexed variables).

resetNodeType

resetNodeType[g,n,newType] returns a modification of the graph g, with the nodeType of node n changed to newType.

resetSBML

resetSBML[] resets the values of $CompartmentNumber, $DomainNumber, and $ModelNumber to 1. resetSBML[n] resets the values of these globals to n. writeSBML invokes resetSBML[1] prior to generating an SBML model.

resetSpecificVariables

resetSpecificVariables[graph, options] resets the values (initial conditions) of specific variables to specific values. Specific indices must be specified. Thus if C is an indexed variable, resetSpecificVariables[variables-> {C[1], C[13]}, values->{12, 14}] will set C[1][0]==12 and C2[0]==14. To set every occurance of C[i], use resetVariables instead. See also resetGraphIC.
Options are:
variables→{var1, var2, ...} list of variavble names
values→{var1value, var2value,...} values for each variable

resetVariables

resetVariables[graph, options] resets the values (initial conditions) of variables to specific values. Every occurance of var1[i], i=1,2,3,... will be set equal to var1value, etc. To set a single variablem e.g., C[17], use resetSpecificVariables. See also resetGraphIC.
Options are:
variables→{var1, var2, ...} list of variavble names
values→{var1value, var2value,...} values for each variable.

rhsShortRightArrow

rhsShortRightArrow[a→b] returns a list of the
products b, e.g., lhsShortRightArrow[a+b+c→x+y+z] returns {x,y,z}

rightBracketHolder

rightBracketHolder is a global symbol used during Cellerator
output translation to hold the desired text symbol used for a right bracket.

rightParenHolder

rightParenHolder is a global symbol used during Cellerator output
translation to hold the desired text symbol used for a right parenthesis.

rmvItem

rmvItem[{item1, item2, ...,itemn}, form] returns a list with with the specified
item or items in form removed; form may be either a single item or a list of times.

ruleListQ

ruleListQ[x] returns True if x has is a list of Rules; the difference between ruleListQ and
OptionQ is that OptionQ does not allow indexed options, such as k[5]-> 12; see also RuleQ,rulesQ

rulePairs

rulePairs[x] is identical to optionPairs[x]
optionPairs[opt1→val1, opt2→val2, opt3→val3,...] returns a list {{opt1,val1},
{opt2,val2},...}; opt may be either a list or a sequence of rules and/or options.

RuleQ

RuleQ[x] returns True if x is a rule
and False otherwise. Compare with OptionQ. See also ruleListQ.

rulesQ

rulesQ[x,...] returns True if x is a sequence or list of Rules and False otherwise. Thus it
will return True for any option list as well as rules of the form y[13]-> 12, which is
not an option because of the function (index) argument. Compare with ruleQ, ruleListQ

run

```
run[celleratorSystem,options] will solve the system of differential equations
  produced by interpret, where celleratorSystem is either the output of or input
  to interpret. If celleratorSystem does not resemble the output of interpret, run
  will automatically call interpret, unless it has a Head of graphDomain or pointer.
run[graph,options], will pass control to runFixedInterval; the option
  timespan→duration, timespan→{duration,step}, or any valid option
  to runFixedInterval may be used in this case (more detail below)

Options for run[s,options] are:
timespan→duration, (default=100) integrate and plot from t=0 to t=duration, or
timespan→{start,end},integrate and plot from t=start to t=end (default is 0 to 100)
initialConditions→{v1[t0]=value,...}; unspecified initial conditions
  are set to zero,or if this option is omitted all ics are set to zero;
plotVariables→None (default), All, or {v1,v2,...} (Important:
  do not specify the '[t]' after the variable names.)
plotFunctions→{{title,expression},{title,expression},...} (default is plotFunctions→
  None); additional functions to plot across results, e.g., {{flintstones,
  fred[t]*wilma[t]+0.5*Barney[t]}, {simpsons,bart[t]-homer[t]},...}; Important:
  the '[t]' MUST be specified after each variable name in the expression.
samePlotVariables→None (default) or {v1,v2,...} or {{v1,v2,...}, {u1,
  u2,...},...} a list of variables to plot on the SAME plot or plots.
runMessages→True, If set to False then a message will not be written to
  the screen indicating when 'run' is creating each plot; may be desirable
  to suppress extraneous display generated by the 'Export' function
rules→{}, an optional list of rules to apply to rate constants.
plotColumns→3, number of columns to use when displaying plots.
resetConditions→{If[condition,rules],If[condition,rules],...} is a list of
  conditions, and an associated list of rules of the form variable→value, to apply
  if condition is met. This option can also be specified as a list {{condition,
  rules},{condition,rules},...} (i.e., the 'If[...]' can be replaced by '{...}'
maxRunSteps→200, maximum total number of iterations to allow before assuming an infinite loop.
frozen→{}, frozen variables to pass to
  interpret (only applies if the input is not an interpreted system)
showODEs→False, this option is ignored unless interpret needs
  to be called, in which case it is passed on to interpret
run→True, if False, run will not integrate the system but will still
  write any requested ode output, and will return an interpreted system.

Also, any valid option for NDSolve or Plot may be specified.

Example:
s=interpret[reactionList];
run[s, timeSpan→ {0,50}, MaxSteps→ 15000,plotVariables→ All, resetConditions→
  {If[x>.5,{x->3,y->2},If[z<.2, z->0]}, rules→{k1→(y[t]+3*x[t]), k2→Sin[x[t]]}]

Options for run[graph,options] are
breaklinks-> True (default): check to see
  if links should be broken as a result of cell grows. See dmax.
exclude→{protein1, protein2, ...} list of proteins to exclude
  from the list of GRN equations, only relevant if grn→True.
grn→False: set to True to run with GRN terms included.
mu→ 0.065: growth rate, m'=mu*m
dmax→ 2: distance at which links are broken. Links will only
  be broken if they grow from less than dmax to greater than dmax during an
  integration step, i.e., if they are already > dmax initially, the link will
  remain intact. To inhibit checking for link breakage, use breaklinks-> False
initialConditions->{name→value,name→value,...} name can be either indexed (like x[3])
  or unindexed. If it is unindexed, then every occurance of name[i] will be reset to the
  specified value.  The unindexed variables are assigned first, so that {mass→3, mass[2]→
  1.5} will reset every mass[i][0]==3 EXCEPT for mass[2][0]==1.5. Note that the format
```

for this option is different from the run[celleratorSystem...] option of the same name.
lambdaMatrixMode→Default - when grn→True, tells how to calculate
the geometrical connectivity matrix (described separately).
massGrowth->exponential,logistic,none
mitosis→True (default): check for cell division. False means to inhibit the check for cell division. If there is no mitoticOsillator (e.g., the organism was created with option division->False) then even if mitosis->True cell division will not occur. This option is only intended to be used to inhibit cell division when the oscillator is otherwise present and working; it will not create an oscillator in each cell if one is not already present.
rules→ {}: list of valid Mathematica Replacement rules to be applied before integration.
plotVariables→ None (default), All, or {*var1, var2, ...*},
list of variables to plot for complete joined solution at end of run
plotVariablesAlways→ None(default),All or {*var1, var2, ...*}, list
of variables to plot immediately after *EACH* call to NDSolve
plotTrajectories→ False,if True, generate a 3 dimensional plot
of the trajectories of each node after *EACH* call to NDSolve
plotCoordinates→ False, if True, a separate plot of all the x,y,z
coordinates of each node will be plotted after *EACH* call to NDSolve;
note that if plotTrajectories→ All, these plots will be redundant.
run→True, if False the simulation will not be performed, but the differential
equations and initial conditions that would have been used will still be generated.
Use this option if you just want to generate the grn equations and look at them.
useStoppingTest→ False, if True, the undocumented NDSolve option StoppingTest will be used to check for cell division or link breakage requirements; this may be be faster. If False, NDSolve is run for the entire duration step. In either case Cellerator interpolates to find the precise time in which an event occurs; the only difference is the duration of the NDSolve run, which may, or may not, affect the accuracy of the interpolating Function.
In addition, any valid option for NDSolve may be used.

runFixedInterval

runFixedInterval[*duration,step,options*] returns a list of the form {{solution1, solution2, ...}, {graph1, graph2, ...} }, where each solution is {*actual-start-time, actual-end-time, interpolating-function*}; *graphs1* is the starting graph and the remaining *graphs* are the graphs at the end of each solution; thus there will be one more graph than solution. The arguments are: *duration* = total duration of run; and *step* = time step of each call to NDSolve; if an event occurs during a call to NDSolve, the subsequent call will start before the end of the prior call. The current list being built is stored in the global variables {globalSolutions, globalGraphs} which can still be accessed even if the run terminates prematurely.

runFixedInterval may also be invoked as run[*duration, step, options*]

Options are:
graph→*name-of-graph*; if omitted, a simple one-node graph will be created.
breaklinks-> True (default): check to see
if links should be broken as a result of cell grows. See dmax.
exclude→{protein1, protein2, ...} list of proteins to exclude
from the list of GRN equations, only relevant if grn→True.
grn→False: set to True to run with GRN terms included.
mu→ 0.065: growth rate, m'=mu*m
dmax→ 2: distance at which links are broken. Links will only
be broken if they grow from less than dmax to greater than dmax during an
integration step, i.e., if they are already > dmax initially, the link will
remain intact. To inhibit checking for link breakage, use breaklinks-> False
initialConditions->{name→value,name→value,...} name can be either indexed (like
x[3]) or unindexed. If it is unindexed, then every occurance of name[i] will be
reset to the specified value. The unindexed variables are assigned first, so that
{mass→3, mass[2]→1.5} will reset every mass[i][0]==3 EXCEPT for mass[2][0]==1.5.
lambdaMatrixMode→Default - when grn→True, tells how to calculate
the geometrical connectivity matrix (described separately).
log →True; if False, will suppress message printing
mitosis→True (default): check for cell division. False means to inhibit the check for cell
division. If there is no mitoticOsillator (e.g., the organism was created with option
division->False) then even if mitosis->True cell division will not occur. This option is
only intended to be used to inhibit cell division when the oscillator is otherwise present
and working; it will not create an oscillator in each cell if one is not already present.
rules→ {}: list of valid Mathematica Replacement rules to be applied before integration.
plotVariables→ None (default), All, or {*var1, var2, ...*},
list of variables to plot for complete joined solution at end of run
plotVariablesAlways→ None(default),All or {*var1, var2, ...*}, list
of variables to plot immediately after *EACH* call to NDSolve
plotTrajectories→ False,if True, generate a 3 dimensional plot
of the trajectories of each node after *EACH* call to NDSolve
plotCoordinates→ False, if True, a separate plot of all the x,y,z
coordinates of each node will be plotted after *EACH* call to NDSolve;
note that if plotTrajectories→ All, these plots will be redundant.
run→True, if False the simulation will not be performed, but the differential
equations and initial conditions that would have been used will still be generated.
Use this option if you just want to generate the grn equations and look at them.
useStoppingTest→ False, if True, the undocumented NDSolve option StoppingTest will be used
to check for cell division or link breakage requirements; this may be be faster. If False,
NDSolve is run for the entire duration step. In either case Cellerator interpolates to
find the precise time in which an event occurs; the only difference is the duration of
the NDSolve run, which may, or may not, affect the accuracy of the interpolating Function.
In addition, any valid option for NDSolve may be used.

runFixedIntervalWrapper

```
runFixedIntervalWrapper[g,options] is used by run[] to format a call to
  runFixedInterval. g must be either a graphDomain or a pointer to a graphDomain
Options are:
timeSpan→duration or timeSpan→{duration,stepsize} (default {10, 5}).
Any other valid option to runFixedInterval can also be specified.
NOTE:Not intended for end user execution.
```

runGRN

```
runGRN[G, T, options] executes NDSolve and returns
  the list of interpolating functions for all proteins in a graph.
```

runPlot

```
runPlot[variable, solution, {tstart, tend}, options] plots a variable or list of
  variables against a solution of run.  If more than one variable name is specified the
  variables are plotted on a grid. Any options valid for Plot can be used. Unindexed
  variables are fully expanded to include all existing indices in the solution.

runPlot[solution,options] plots variables on a grid
Additional Options for this version:
plotVariables-> {var1,var2,...} or All
plotColumns->3 (number of columns in the grid)
plotType->"Linear"(Default) ,"Log" ,"LogLinear" ,"LogLog"
```

runStep

runstep[graph, step, options] formats a call of duration step to NDSolve for a growing organism described by *graph*, checks to see if any events have occured, and creates a modified graph as described by cell growth, cell division, and/or link breakage. It returns a list {{*actual-begin-time, actual-end-time, interpolating-function*}, *final-graph*}. The duration of the *interpolating-function* may be longer than the time span, if either cell division or link-breakage occurs The actual-end-time is the smallest of *actual-begin-time+step*, or the time of the first event. Any valid option to NDSolve may be used. The *final-graph* is the modified graph after at *actual-end-time* after all events have occured, including link breakage and cell-division.

Options are:
rules→ {}: list of valid Mathematica Replacement rules to be applied before integration.
plotVariables→ None(default),All or {*var1, var2, ...*}, list of variables to plot immediately after call to NDSolve
plotTrajectories→ False,if True, generate a 3 dimensional plot of the trajectories of each node
plotCoordinates→ False, if True, a separate plot of all the x,y,z coordinates of each node will be plotted; note that if plotTrajectories→ All, these plots will be redundant.
useStoppingTest→ False, if True, the undocumented NDSolve option StoppingTest will be used to check for cell division or link breakage requirements; this may be be faster. If False, NDSolve is run for the entire duration step. In either case Cellerator interpolates to find the precise time in which an event occurs; the only difference is the duration of the NDSolve run, which may, or may not, affect the accuracy of the interpolating Function.
In addition, any valid option for NDSolve may be used.

Note:runStep is called by runFixedInterval and is not designed to be called by the end user.

samePlot

samePlot[variables, sys, options] will plot a set of variables from a Cellerator run on the same plot. sys is the output of run, e.g., a list of the form {{range, list of int. funcs}, {range, list of int. funcs},...}. Options is any valid Plot option. Here sys is either a String that gives the name of variable containing the Cellerator System or the actual variable name itself.

samePlots

samePlots[{{var, var,...}, {var, var, ...},...}, sys, opt] plots each set of variables together on a single plot. Each sublist of varibables is plotted using samePlot.

sbml2ODEs

sbml2ODEs[options] generates level 2 SBML. It is called by writeSBML via writeODESinSBML and is not intended to be user invoked.

SBMLLevel1ArrowReaction

SBMLLevel1ArrowReaction[reaction,options] returns the level-1 SBML for a reaction. The format of reaction is the same as the format of a single reaction being sent to interpret, including an arbitrary number of rate constants. New symbols are generated for unspecified rate constants. Values of rate constants can be specified, e.g., as SBMLLevel1ArrowReaction[{a→b,k1}, k1->42]. Unspecified values are set to zero in the SBML. The following reactions have been implemented: S→P, S⇄P, $S \overset{X}{\rightleftarrows} P$, $S \underset{Y}{\overset{X}{\rightleftarrows}} P$, $S \overset{X}{\mapsto} P$, S↦P (hill, GRN, and NHCA)

The following have not been implemented: Comp[..] reactions (except for Comp[S]→P,$\text{Comp}\overset{X}{[S]} \mapsto P$ (S, P must be single species), Comp[S]↦P (hill)); transcriptional reactions (↦) with multiple reactions producing the same product, although they can be specified with CirlcePlus (A⊕B↦C)
Implmentation Note: use of ∅ results in a new species EmptySet being created, which should be set as a boundary condition, so that non ODE will be created for it.

SBMLLevel1Reaction

SBMLLevel1Reaction[options] generates the level 1 SBML for a single reaction that is described by the following options
indent→"", additional string to be added to left-side of every line.
kineticLaw→formula, where formula is a string (C-like) that describes the reaction (see SBML level 1 spec); a formula of "0" is the default.
name→ reactionName, string naming the reaction, or reactionnnn if name→ None is specified
parameters→{} or a list of rate constants, whose values are given, in order, in parameterValuues
parameterValues→{} values of the parameters; defaults are zero.
products→{x1,x2,...} list of species on the right hand side of the arrow.
stoichiometry→{{m1,m2,...},{n1, n2,...}}, stoichiometries to use for each source (m's) and product (n's). If insufficient stoichiometries are specified they are set equal to one. (Symbolic stoichiometries may be used in the n's but not in them m's. Note that SBML does not allow symbolic stoichiometry at all)
reactants→{r1,r2,...} list of species on the left hand side of the reaction

sbmlODEs

smblODEs[{odes, variables}, file, options] writes a system of differential equations in sbml to the file name "file". Options are:
initialConditions->{var1[x0]==val1, var2[x0]==val2,...}
compartment->*value* (default is "celln" where n is a counter that increments by 1 every time that sbmlODEs is called.
model→*value* - name of SBML model; default produces a name "modeln" where n is a counter that increments by 1 every time that sbmlODEs is called.
note->text message to include

sbmlVariable

sbmlVariable[x[i],comp] returns a string representing the SBML representation of Cellerator indexed variable x. In cellerator, x[i] refers to protein x in graph node i; in SBML this becomes protein x in compartment i and is references as comp[i].x.

semiHexagonalGrid

```
semiHexagonalGrid[n] returns a grid of points arranged
  on a hexagonal grid filling up the top half of a hexagon (e.g., a
  meristem), where n is the number of points on each edge of the hexagon.
```

separateIndices

```
separateIndices[x[i,j,k,...] returns x[i][j][k][...],
  i.e., the indices are separated with brackets from one another.
```

setDomain

```
setDomain[x] is a Domain object that represents any list. It can be instantiated with expandDomain
```

setIndex

```
setIndex[n,i] replaces the index on all the indexed variables in node n with i. It assumes
  that all indices have the same value. Thus all if n is node 5, with variables C[5][t],
  M[5][t], etc., then setIndex[n,7] returns a node with variables C[7][t], etc.. All
  variables, embeddings, differential equations, and ICs are modified to be self-consistent.
```

shape

```
shape is an option to organism that selects the shape of a cell; values are:sphere,disk.
```

showFlatGraph

```
showFlatGraph is called by showGraph if the
  graph is strictly two dimensional; it should not be used directly.
```

showGraph

showGraph[graphDomain[...], options] draws a picture of the graphDomain
labels→ True (default) if nodes are numbered
showlinks→ True (default) to draw the links
nodeColors→ {{nodetype, color}, {nodetype,
  color}, ...} or Automatc to use the colors in colorChart automatically
colorChart-> AllColors: a table of color names or color assignment functions to be
  used cyclically for displaying node colors; ignored unless nodeColors-> Automatic
linkColor→ Hue[0.7] color of links if showlinks is True
labelColor→ Hue[0.9] color of labels if showlabels is True
key→ False, set to True to display a legend of cell-types
nodeSize→ 0.05, size of dots for nodes as fraction of plot width
showDisks→True: for 2-D graphs, draw Circles proportional to radii of each cell
fillDisks→True: for 2-D graphs, when showDisks->True, if fillDisks->
  True, the interiors of the circles will be filled with a color whose Hue
  is proportional to the value of colorVariable of the disk (Hue[0]...Hue[1])
colorVariable→mass (Default), name of variable to used to
  color cells, only meaningfull when showDisks is True
massRange→"Unknown" automatically figure out range of masses for radii of disks setting
  smallest value to a radius of 0 and largest to magnifier; or massRange→{mlittle,
  mbig}, use radious of 0 for mlittle and radius of 1for mbig, and interpolate
  linearly in between. To generate movies using the same scale one would want to pass
  in a value for this parameter to assure that the same scale is used on each frame.
variableRange→"Unknown" automatically figure out the ranve of colorVariable for colors
  of disks, using a Hue[0] to display the smallest value and Hue[1] to display the
  largest value, and interpolate linearly in between; or variableRang→{vlittle, vbig},
  use Hue[0] for colorVariable=vlittle, Hue[1] for colorVariable=vbig, and interpolated
  linearly in between. To generate movies using the same scale one would want to pass
  in a value for this parameter to assure that the same scale is used on each frame.
magnifier->1 (radius of largest disk)
Any valid option for Graphics3D (or Graphics,
  if the graph is strictly 2-dimensional) can also be specified

showGraphODEs

showGraphODEs[graph] displays a table listing the names of the variables, the initial
  condition, and the corresponding differntial equation.It does not include grn terms.

showGraphs

showGraphs[graph, solution, {tstart, tend, dt},options] generates a movie of the specified
  graph by interpolating into the interpolatingFunction solution at the specified time
  intervals. Options are any valid option for showGraph or Graphics (or Graphics3D).

showGrid

showGrid[x,options] will display a grid of points. x must have the form {{x1,y1,0},{x2,y2,0},...}

showMeristem

showMeristem[g,options] applies showGraph to the meristem, with Rib
meristem in Blue, the Peripheral zone in Green, adn the Central zone in red.

showPoints

showPoints[{{x1,y1,z1},{x2,y2,z2},...}, options] displays and returns a 3-dimensional
plot of the points whose coordinates are given. If all of the z coordinates are zero
then 2-dimensional plot in the x-y plane will be generated. A Graphics3D object
is returned. To suppress display use the DisplayFunction option. If the points are
entirely in the x-y plane (all z's are zero) then a Graphics object will be returned.
Options are:
color->Any graphics color function such as RGBColor[R,G,B], CMYKColor, etc. (default is Red).
size->size of points as fraction of the plot width (default is 0.02).
Any valid option for Graphics3D can also be used

showProtected

showProtected[] returns a list of all protected symbols in the context Global.

sigmoid

sigmoid is the default name for for the Cellerator sigmoid
function used by run during simulations. The default value is (1 + x/Sqrt[
1 + x^2])/2; this value can be overwritten by either redefining the function
GRNSigmoidFunction[x] or using the option sigmoid in run[]. See also Sigmoid.

singleParameterSBML2

singleParameterSBML[{options}] returns the level 2 SBML for a single
  parameter. Options must be enclosed in a bracketed list or errors will occur.
options are
parameterName->name
dimension->0 (number of indices)
indices->{index1, index2,...} (list of all the allowed values that an index can take on
values->{value1, value1,...}]
domainName->{domain1,domain2,..} name of domain for each dimension, in order
indent->"" - additional indentation to
  add (e.g., tab stops) to add to left hand side of each line
example (1):
singleParameterSBML2[{parameterName→tau,dimension→0,values→42}] will return
<parameter name="tau" value="42" />
example (2)
singleParameterSBML2[{parameterName→lambdaVector,dimension→1,domains→{max→
  2,min→1},indices→{1,2},values→{0.5,0.7},domainName→{"Domain275"}}] will return
<parameter name="lambdaVector[Domain275]" value="switch(Domain275,1,0.5,2,0.7,0)" />
example (3) (not quite valid SBML)
singleParameterSBML2[{parameterName→T1,dimension→2,domains→
  {{max→2,min→1},{max→2,min→1}},indices→{{1,1},{1,2},{2,1},{2,2}},values→
  {0.616,0.842,0.103,0.794},domainName→{"Domain262","Domain263"}}] will return
<parameter name="T1[Domain262][Domain263]" value="data(0.616,0.842,0.103,0.794);" />

smallRandomNumber

smallRandomNumber returns a random number between 0.001 and 0.01 or between -0.01 and -0.1

smallRandomVector

smallRandomVector[n] returns a list of random numbers of length n,
smallRandomNumber[n,*size*] returns a list of random
  numbers of length n, where each number is between -*size* and +*size*.

solutionCounter

solutionCounter counts the number Cellerator solution
  pointers that have been allocated as pointer["celleratorSolution[n]"]

species

species is an option for organism or standardNode that
  specifies a list of chemical species (e.g., proteins, RNA, etc.) in a
  node or organism. See organism or standardNode for the correct syntax.

speciesIC

speciesIC is an option for organism or standardNode that specifies a list of initial conditions
for the species in a graph node or graph.See organism or standardNode for the correct syntax.

speciesodes

speciesodes is an option for organism or standardNode that specifies a
list of differential equations for the species in a graph node or graph. These
differential equations are used in additon to any equations generated as a
result of speciesReactions.See organism or standardNode for the correct syntax.

speciesReactions

speciesReactions is an option for organism or standardNode
that specifies a list of Cellerator reactions for the species in a
graph node or graph.See organism or standardNode for the correct syntax.

springConstant

springConstant is an option to springDomain that specifies
the name of the variable that represent that spring's spring constant
springConstant[springDomain[...]] retrieves the spring constant from a springDomain
springConstant[graphDomain[...],n] retrieves the
spring constant from the springDomain of the nth link of the graph
springConstant[linkDomain[...]] retrieves the springConstant
from the springDomain of the specified link

springDomain

springDomain[options] represents a springDomain
springDomain[springDomain[...],options] returns a modified springDomain
springDomain[graphDomain[...],n] retrieves the springDomain from the nth link of the graph

Options are:
springConstant→k
springLength→d

see also springConstant, springLength

springForce

springForce[g,n1,n2] gives the spring force (a formula) between two linked nodes (integers
n1 and n2) in a graph. If the nodes are not linked, an error message is displayed.
springForce[g,n] gives the total spring force (a formula) on node n (an integer)
in graph g due to all links to that node.This function is grossly inefficient
because it uses SumOverNeighbors. For a faster computation see fastSpringForce

springLength

springLength is an option to springDomain that specifies the name
  of the variable that represent that spring's resting length, typically d
springLength[springDomain[...]] retrieves the springLength from a springDomain
springLength[graphDomain[...],n] retrieves the
  springLength from the springDomain of the nth link of the graph
springLength[linkDomain[...]] retrieves the springLength
  from the springDomain of the specified link

sprint

sprint[s,options] is the same as
  predictTimeCourse but does not have the options table, file, or save.

sprintSystem

sprintSystem[variables, odes, ic returns a tabular listing of
  variables, initial conditions and differential equaitons (using TableForm).

ssMultipleListPlot

ssMultipleListPlot[spreadsheet] formats a spreadsheet for MultipleListPlot
Options:
header->False, if True, assume the first row of the spreadsheet contains headers

SSystem

SSystem[arguments] is an uninstantiated function that indicates to interpret that a
  reaction is part of an SSystem. Typical usage is {{$v_i \mapsto v_a$, SSystem[τ, k^+,k^-,c^+,c^-]}.
If there is only a single reaction of the form $v_a \mapsto v_i$ then the ultimate
  interpretation is $\tau \frac{dv_a}{dt} = k^+ (v_i)^{c^+} - k^- (v_i)^{c^-}$.\If there are several
  reactions of the form $v_1 \mapsto v_a$, $v_2 \mapsto v_a$, ..., typically written as
{{$v_1 \mapsto v_a$, SSystem[τ, k^+,k^-,c_1^+,c_1^-]},{$v_2 \mapsto v_a$, SSystem[τ, k^+,k^-,c_2^+,c_2^-], ... },
  then they are combined as $\tau \frac{dv_a}{dt} = k^+ \prod_i v_i^{c_i^+} - k^- \prod_i v_i^{c_i^-}$. Note that all S-System
  reactions with the same product must have matching τ, k^+, and k^- values. If
  different values are supplied the value supplied with the first occurance of an S-
  System reaction with a given product is used, and the addtinal values are ignored.
 Missing parameters are supplied with default values as τ=1, k^+=k^-=c^+=c^-=0

SSystemFunction

SSystemFunction[{x_1, x_2,...}, τ, k^+, k^-,
  {c_1^+,c_2^+,...}, {c_1^-, c_2^-,...}] returns $\frac{1}{\tau}(k^+\prod_i x_i^{c_i^+} - k^- \prod_i x_i^{c_i^-})$.

SSystemFunction[x, τ, k^+,k^-,c^+,c^-] returns $\frac{1}{\tau}(k^+ x^{c^+} - k^- x^{c^-})$.

Default values are τ=1, k^+=k^-=c^+=c^-=0

standardNode

standardNode[options]; options are:
coordinates→ random,initial coordinates for node
index→ 1, integer that labels the node
initialMass-> 1, value of mass at t=0
massGrowthRate->0.065, the constant "mu" in mass growth formulas
mitotic → True: is cell division allowed
cellDivisionMassThreshold→1, minimum mass of cell to allow for cell division
mitoticOscillator→{} Options to be sent to
  createMitoticOscillator (see mitoticOscillator for more information)
growing→ True, if cell growth allowed
reinitialize→False, if True reset $FirstMitoticOscillator to
  True first.species→ {}: list of additional species to include in this node
speciesodes→ {}: odes that species obey
speciesIC→ {}, initial concentrations of species
speciesReactions→ {}: Cellerator arrow reactions that species obey (both reactions
  and odes can be specified; the right hand sides of the resulting odes will be added)
steadyStateMass->value: reset the steadyStateMass field
  of the nodeData field to the specified values

Usage note: If the global variable $FirstMitoticOscillator=False then the mitotic
  oscillator call will look for predefined variable names in the global variable
  $CelleratorMitoticVariables. These globals are used by the function organism to define
  a consistent set of variable names. However, if standardNode is invoked by the user
  directly, either (a) the global variable $FirstMitoticOscillator should be set to
  True, or (b) the option reinitialize->True should be set to avoid assignment errors

steadyStateMass

steadyStateMass is an option for standardNode, organism,
and meristem, as well as a subfield of a nodeDomain's nodeData.

In standardNode or the nodeData field it should be an atom
that gives the desired steady state mass to be used for logistic growth
models of the corresponding node's mass, e.g., steadyStateMass→*value*.

In organism it can be any of the following:
steadyStateMass→*value*, use value for every node
steadyStateMass→{*value1*, *value2*,...}, use value1 for
node1, value2 for node2, etc. Extra values are ignored. If not enough
values are provided then the first value in the list is used as a default.
steadyStateMass→{mass[i1]→val1, mass[i2]→val2,...,default→defaultvalue}

For Meristem, the only allowed format is steadyStateMass→
{CZL1→ 2.0, PZL1→2.4, CZL2→2.4, PZL2→ 2.4, Rib→ 3.7, Floor→ 4.0} and the
values are randomized with a uniform distribution and a +/- 15% variation.

stnQ

stnQ[x] returns True if x looks like a list of reactions,
possibly including rate constants, that can be sent to interpret.

stringMultisperseAfter

multisperseAfter[list1, {list2a, list2b, list2c, ...}] intersperses each of list 2a, 2b, ... into
list 1 and then pairs the results into strings. Example.multisperseAfter[Range[5],{{A,B,C,D,
F},{P,Q,R,S,T}, {alpha, beta, gamma, delta, epsilon}}] returns {{"A1","B2","C3","D4","F5"},
{"P1","Q2","R3","S4","T5"},{"alpha1","beta2","gamma3","delta4","epsilon5"}}.

stringMultisperseBefore

stringMultisperseBefore[list1, {list2a, list2b, list2c, ...}] intersperses list 1 into each of
list 2a, 2b, ... and then pairs the results into strings. Example.stringMultisperseBefore[
Range[5],{{A,B,C,D,F},{P,Q,R,S,T}, {alpha, beta, gamma, delta, epsilon}}] returns {{"1
A","2B","3C","4D","5F"},{"1P","2Q","3R","4S","5T"},{1alpha,2beta,3gamma,4delta,5epsilon}}.

stringPair

stringPair[list] pairs off successive elements of a list into
strings, e.g., stringPair[{A,B,C,D,E,F,G,H}] returns{"AB", "CD","EF","GH"}.
If list constains an odd number of elements, the odd element is dropped from
the list. Thust stringPair[{A,B,C,D,E}] is the same as stringPair[{A,B,C,D}].

subconvertGK

subconvertGK[specie,digestedReaction,index] converts digested reactions into terms in differential equations. digestedReaction is a single reaction generated by digest, such as {A,B}→{C,D}. index is used to automatically generate rate constants which have not been previously defined. subconvertGK should not be invoked directly. It is call automatically, when required, by interpret. An error message will be generated if the rateDB is not initialized prior to being called.

subTree

subTree[t, options] returns a subtree of tree t
options are:
address-> hash code, e.g., {2,1,2} means right,left,right sub-branch; if address-> {}, the entire tree is returned; if the address is a non-existent address, addrecheck is used.
addrcheck->True (default): an error occurs if address is non-existent
addrcheck-> False: an empty tree (a tree with a single node that contains the empty set) is returned.

sumOver

sumOver[f, x] returns sumOverList[f,x] if x is a list and sumOveraDomain[f,x] if x is a domain. It gives an error otherwise. sumOver returns a number (or at least an expression) and not a domain. It can alaso be expressed as $\sum_{x} f$, e.g., $\sum_{\{10,20,30\}} \text{Cos[g[\#]] \&}$ returns Cos[g[10]]+Cos[g[20]]+Cos[g[30]].

sumOveraDomain

sumOveraDomain[f,d] applies the function f to each element of the expanded domain d and returns the sum. For example, sumOveraDomain[f,intDomain[50,100,25]] returns f[50]+f[75]+f[100]. See also sumOverList.

sumOverDomain

sumOverDomain[expression, {domain1, domain2,...}] is an uninstantiated function that represents the sum of expression over all of the domains listed. The format for domaini is {index, list-of-values}. Examples:
sumOverDomain[x[j],{{j,Range[5]}}] represents x[1]+x[2]+x[3]+x[4]+x[5]
sumOverDomain[x[j][k],{{j,Range[5]},{k,{14,19,23}}}] represents the sum x[1][14]+x[1][19]+x[1][23]+x[2][14]+x[2][19]+x[2][23]+x[3][14]+ x[3][19]+x[3][23]+x[4][14]+x[4][19]+x[4][23]+x[5][14]+x[5][19]+x[5][23]
To expand the sum, see expandSum or expandSums.

sumOverDomainSBML

sumOverDomainSBML[expression, {domainDescription1,domainDescrption2,...},options] returns a list {domains,functionCall}, where functionCall an a Mathematica function call to sumOverDomain that has the arguments in the same order as the SBML2 function of the same name, and domains is a list of the new SBML domains that are being summed over.

sumOverList

sumOverList[f,s] applies the function f to every element of the list s and returns the sum. For example, sumOverList[ff, {a,b, c,{q}}] returns ff[a]+ff[b]+ff[c]+ff[{q}]. Compare with sumOveraDomain

symbolQ

symbolQ[x] returns True if Head[x] is Symbol and False otherwise. Thus indexed variables (like x[7]) will not be considered symbols and will return False, whereas unidexed variables (like x) will return True.

table2HTML

table2HTML[table,options] returns an html formatted table version of the input table.
indent->"" extra text to add to the beginning of each line (typically whitespace)
newline->True: include a newline character at the beginning of each table row
tableheader->True, the first line of the table is assumed to contain header information that is tagged with <thead>

template

template is an option for run[graph,...]; it has the format template->{h→*vectorspec*, lambda→*vectorspec*,T1→*matrixspec*,T2→*matrixspec*,TAU→*vectorspec*,TI→*matrixspec*,TO→*matrixspec*}, where *matrixspec* has the form {{*value*(11), *value*(12),...},{*value*(21),*value*(22),...}}} and *vectorspec* has the form {{*value1*, *value2*,...}} and the dimensions of the matrix and vector are given by number of interacting GRN proteins.

textRates

textRates[options] returns an ASCII version of the rate constants in an interpreted system. The same options as displayRates may be used, with the following additions:
htmlTable->True, return as an html formatted table, using table2HTML (any option for table2HTML may be used as well)
indent->"" extra text to add to beginning of each line in html table
newline->True, if True, a newline character is included in each line of the html table (will make it more readable)

threadfield

threadfield[f,d1,d2,...] threads the function f over the expanded domains d1, d2, d3,...For example, expandDomain[threadfield[f,setDomain[{3, 42.7, Pi}],intDomain[3]]] returns {f[3,1],f[42.7`,2],f[π,3]}. See also field.

timeChecker

timeChecker[message] prints the text message and the amount of CPU time that has elapsed since the last call to timeChecker. The CPU timer can be initialized with initTimeCheker[]

timeSpan

timeSpan->{duration, step} is an option for run[graph, options] that indicates the maximum duration of the run (duration) and the duraton of each invocation of NDSolve (step)
timeSpan->duration or timeSpan->{start,stop} is an option for run[system, options], where system is either the input to or the output from interpret.

touchingDistance

touchingDistance[mass1,mass2,density] gives the distance two cells will be from one another if they are just touching.

tree

tree[x,y,z,...] represents a tree with x on the first branch, y on the second, z on the third, etc. Trees may be nested. See the following functions to manipulate trees: expandTree, leavesIn, treeQ, binaryTreeQ,showTree, lookup, subTree,addLeaf, interpretTree,lineageTree

treeQ

treeQ[x] returns True if x is a tree and False otherwise

triangularGrid

triangularGrid[r] generates a grid of points {{x,y, 0},{x,y,0},...} that form an equilateral triangle of side r+1 points.

underscore

underscore="$underscore$"

unionDomain

unionDomain[d1,d2,..] represents the disjointUnion
of all the specified domains. It is useful for encoding trees.

uniqueRateConstant

uniqueRateConstant generates a new rate constant (as a symbol, not a string) of the form kxxx
where xxx is an integer that is incremented by 1 every time rateConstant is called; unlike
rateConstnat, uniqueRateConstant increments xxx until a rate constant of the form kxxx has
not been previously defined); the counter is not reinitialized by initializeSBMLReactions

unitGraph

unitGraph[options] returns a graph with a single standardNode
at the origin. Options can be any option for GraphDomain or standardNode.

unitPulse

unitPulse[t] returns 1, -1/2≤x≤1, and zero otherwise
unitPulse[t,width] returns 1, if -width/2≤x≤width/2, and zero otherwise
unitPulse[t, width, tcenter] returns 1 if t is within ±width/2 of tcenter, and zero otherwise
See also: pulse, unitPulseTrain, pulseTrain

unitPulseTrain

unitPulseTrain[t,n] delivers a train of n unit pulses.The first pulse is centered at t=0.
unitPulseTrain[t,width,n] delivers a train of n unit pulses of
the given width; the time between pulses is the same as the pulse duration.
unitPulseTrain[t,width,interval,n] delivers a train of n unit
pulses at the specified center-to-center interval.]
See also pulseTrain, pulse, unitPulse

unprotectAll

unprotectAll removes protection from all protected symbols in the context Global.

Vacuum

Vacuum is a Cellerator global symbol that is equivalent
to ∅, the Mathematica \[EmptySet] character. See ∅ for more details.

validateRateConstants

validateRateConstants[{reaction,k1,k2,...,km},options] returns {reaction, K1,K2,...,KN} where
  K1,...,KN are rate constants for reaction. If sufficient rate constants k1,...,km are already
  supplied, they are used; if too few are supplied, additional ones are added to produce
  the right amount; if too many are added, only the precisely correct amount are returned.
validateRateConstants[{{reaction,rates},{reaction,rates},...,{reaction,rates}},
  options] will return a validated circuit (compatible with interpret).
Options are:
verbose→ False, if True, print the reaction and rate constant to the screen.
validate->True, if False, don't automatically generate rate constants.

validateReaction

validateReaction[reaction] returns a correctly
  formatted reaction from a partially correct reaction filling in assumed
  defaults. It is currently only implemented for the following reactions:

GMWC reactions: $\{S1, S2, \ldots\} \overset{\text{Enzyme}}{\underset{\{\{A1,A2,\ldots\},\{I1,I2,\ldots\}\}}{\Longrightarrow}} \{P1, P2, \ldots\}$

If any other reaction is passed the reaction will be returned unchecked.
validateReaction is intended to be called by xlateGK (or before xlateGK is called).

variables

variables[GRNEQ[options]] retrieves the list of variables in the GRN equations.

variablesIn

variablesIn[g] returns an integer, the number of variables in the graph g.

warningTest

warningTest[test, options], prints warning message if test is True.
options are:
    id→name of calling module (used for message)
    message→message to print in case test passes

welcomeMessage

```
welcomeMessage is a global that contains the text of the message
  that is printed when Cellerator is loaded. Its current Value is Cellerator™
  Version 1.5.0 (03-Jan-2005), loaded at Jan. 13, 2005, 14:39:11.825742
©2001-2005 California Institute of Technology.  U.S. Government
  Sponsorship Acknowledged. All rights reserved.
Patent Pending (USPTO App 09993291).
The contents of this file may not be
  copied, distributed or transferred without written permission.
```

wrapNicely

```
wrapNicely[s,options] returns a modified string that includes newline characters for the SBML
Options are:
wrap->True: if false, no wrapping is done
indent->"" additional string to add to the beginning of each new line
linewdidth->80 desired maximum string line length (excluding indent characters)
If there are no convenient places to break the
  line, the line lengths could be longer than the requested linewidth.
Any valid option for breakPoints can be used.
```

writeODESinSBML

```
writeODESinSBML[options] writes a cellerator model in SBML. It formats
  the call to either level 1 or level 2 SBML and is intended to be called
  via the wrapper function writeSBML and is not intened to be user-invoked.
  The model must be encapsulated in an organism. For options see writeSBML.
```

writeSBML

writeSBML[graph, options] will generate the SBML for a cellerator model encapsulated by a graph.
writeSBML[options] will do the same thing, but the options equations, initialConditions,
  variables, parameters,species,lambda,numberOfCells are all required. Note that these
  required parameters can be generated as a single option list with run[graph,run->False].
  The first form of writeSBML generates this option list by invoking run in this manner.
writeSBML[reactions,options] will write a list of cellerator reactions to SBML. The
  reactions must be in the same format as they are input to interpret. Values of rate
  constants may be specified in the option list, e.g., as a1->.1, a2-> .2, etc. Initial
  Conditions can be specified as initialConditions->{var1->val1, var2->val2,...}

Options are:
level→2 - SBML Level
version→1 - SBML Version
file->*filename* (defaults to "runYYYYMMDDThhmmss.sbml")
directory->name of directory that file is to be created in (defaults to Directory[])
model->*modelname* - name of SBML model; if not specifies,
  a name *modelxxx* where *xxx* is a counter, is generated automatically.
note->*hypertext* - text of a note to be written to the SBML <notes> field; will be
  encapsulated in ... ; any valid text or html may be included as per SBML speces
compartment->*compartmentName*, name to be used for the SBML compartments that
  represent cells; can be an unindexed symbol or a string; if not specified a name
  compartmentxxx, where *xxx* is an integer counter, will be generated automatically.

Options for graphs only:
functions->{*functionName1*->{*functionDef1*,{arg1,arg2,...}},*functionName2*->
  {*functionDef2*,{arg1,arg2,...}},...} wherere functionName1,functionName2,... are
  names of functions to be include; functionDef1, functionDef2,... are the function
  definitions; the functions are invoked as functionDef1[arg1,arg2,...]. The function
  Sigmoid, distance, and LAMBDA are always written and do not need to be specified.

The following parameters can be generated automatically by run with the "run->False" option used:
equations->{ode1, ode2,...}, list of mathematica differential equations
initialConditions->{ic1,ic2,...} list of mathematica initial conditions
variables->{var1,var2,...} list of all the variables for which there are differential equations
parameters->{*par1*->value1, *par2*->value2, ...} list of all the mathematica variables in the
  equations that are to be treated as constants; the names *par1*,*par2*,... may be indexed
species->{sp1, sp2,...} list of those mathematica variables
  listed in variables that represent protein concentrations
lambda-> {LAMBDA[1,1]->formula, LAMBDA[1,2]->formula, ...}
  list of rules that describe the LAMBDA function
numberOfCells->Value - number of cells in the model (nodes in the graph)

Typical usage example: (abbreviated)

```
m=organism[...options...]
writeSBML[r,grn→True,      compartment→ "myCompartment",
  model→ "myModel",      level→2, note→ "<h1>This is my Cellerator Model</h1>
         <div>Generated by User F. Cellerator, May 1, 2002</div>"
```

xlateGK

xlateGK[reaction] translates reactions in Cellerator arrow notation into an internal
  Mathematica list format for further processing. For example, xlateGK[A+B→C+D] returns
  the list {{A,B}→{C,D}}. There is normally no reason for a user to invoke xlateGK
  directly. xlateGK is invoked automatically, when required, by the interpret command.

yymmddhhmmss

```
yymmddhhmmss[] returns a string representation of the current date and time
  in the JPL/TOPEXTIME format of  "20020215T105718" for Feb 15 2002 at 10:57:18 AM
```

$CelleratorMitoticIC

```
$CelleratorMitoticIC is a global variable that defines the initial conditions
  that are currently being used for a mitotic oscillator.  It is typically
  defined when an organism is created, e.g, by organism[] or meristem[], is used
  by functions such as standardNode to create an oscillator, and by run[] to set
  initial conditions after cell division. It can be an expression with Hold[].
The value can be initialized at the call to organismm[] with the option mitoticOscillator→ {...,
  initialconditions→value,...}; default values are based on oscillator (see also randomIC):
Gardner:{randomIC[0,.3],0,0,0,0}
Goldbeter: {randomIC[0,.3],0,0}
Norel: {randomIC[0,1],randomIC[0,.4]}
Tyson2: {randomIC[0,.3],0}
Tyson6: {1, randomIC[0,3],0,0,0,0}
```

$CelleratorMitoticVariables

```
$CelleratorMitoticVariables contains the names of the variables being used
  in the current mitotic oscillator, for example, {C,M,X} for a Goldbeter model.
```

$CompartmentNumber

```
$CompartmentNumber is a Cellerator™ global variable used to count the number of
  automatically generated compartments in an SBML model. See newCompartment,resetSBML.
```

$DebugCelleratorLoad

```
$DebugCelleratorLoad is used to generate a report to
  the developers when Cellerator generates an error message during load.
```

$DomainDigits

```
$DomainDigits is a global integer (default value = 3) that is used to determine the
  minimum number of digits in a automatically generated domain name. See newDomain.
```

$DomainName

```
$DomainName is a global string (default value "Domain")
  that is used to label automatically generated SBML domains via newDomain.
```

$DomainNumber

$DomainNumber is a Cellerator™ global variable used to count the number of
SBML domains that have been automatically generated. See newDomain,resetSBML.

$firstIntegrationStep

$firstIntegrationStep is a global flag used by runFixedInterval to communicate with
integrateGraph to prevent resetting of random paramters during each integration step.

$FirstMitoticOscillator

$FirstMitoticOscillator is a flag set by functions that call standardNode to indicate
that the variable names and initial conditions being passed should be written to the
global variables $CelleratorMitoticIC and $CelleratorMitoticVariables for later user.

$ModelNumber

$ModelNumber is a Cellerator™ global variable used to count the number of SBML model names that
have been automatically generated during the current session. See newModel,resetSBML.

$newReactionName

newReactionName generates a string "Reaction*nnn*" where nnn is the
value of the global $SBMLReactionCounter, which is also incremented by 1.

$RateConstantCounter

$RateConstantCounter is a global that counts the
number of rate constants that have created during SBML translation.

$RateConstantMinDigits

$RateConstantMinDigits is a global that specifies the minimum number of digits in a
rate constant symbol, i.e, 4 implies knnnn, 5 implies knnnnn, etc. The default is 4.

$ReactionName

$ReactionName is a global string (default value
is "Reaction") that is used to generate new reaction names via newReationName

$reactionSpecies

reactionSpecies is a global variable that holds the
names of the species that are referenced in SBML level 1 reactions.

$savedMatrixParameters

$savedMatrixParameters is used by integrateGraph to save randomly set Matrix Parameters.

$savedParameterRules

$savedParameterRules is a global flag used by integrateGraph to save
randomly set parameters so that they are not reset on each integration step.

$SBMLReactionCounter

$SBMLReactionCounter is a global that counts the number of automatically generated SBML reactions.

$UniqueRateConstantCounter

$UniqueRateConstantCounter is a global that counts the number of unique rate constants that
have created using uniqueRateConstant. It is not reset by initializeSBMLReactions.

∇

{X→∇,d} is a diffusion reaction where d is a diffusion constant or tensor (matrix)
.The diffusion equation is

$$\partial_t X = R + \nabla \cdot (D \nabla X)$$

where R is the right hand side of the ODE due to other cellerator reactions.
If any species undergoes diffusion, only pdes are returned by interpret.
Species that do not undergo diffusion have their diffusion tensor set to zero.
Diffusion options may be specified as interpret[reactions, diffusionOptions->
{diffusion->True, dimensions->3}] (default values shown)

ϕ

∅ is used to represent the empty set in Cellerator. The cellerator reactions X→
∅ and ∅→X mean that X is removed from system (e.g., annihilated) or created out
of nothing, respectively. The symbol ∅ in mathematica can be inserted by entering
\[EmptySet] and is not the same as the symbol ϕ (the Greek Letter phi which
can be entered into Mathematica via the sequence of keystrokes <esc>phi<esc>.
Cellerator also defines the global symbol and Vacuum which can be used interchangeably with ∅.