

Getting Started:

- Download and install bGUI from <http://littleb.org/download.html>
- Update your initialization file (Program Files/littleb/init.lisp):
Add this line to point at your Matlab work folder:
`{#/b/matlab/settings:*model-output-default-pathname* := "C:\Program Files\Matlab7\work\}`
where "C:\Program Files\Matlab7\work\" is the location of your Matlab work directory.
- Run the little b Environment (Start->Program Files->little b->little b Environment)
- Open a new file (File->New)
- Add these lines to the top of the file:
`(in-package :b-user)`
`(include b-user/non-dimensional-ode-biochemistry)`

Syntax:

symbols and numbers: used as variables and function names:

1, +1, -1, 1.2, 1e-3, 1/4 are all numbers

abc, 1a, a1, !a, !, *, 1+ are all symbols

abc and ABC are the same symbol

keywords: special constant symbols starting with a colon:

`:abc` ; the value of `:abc` is `:abc`

() function calls:

`(* abc 123)` ; multiplies the value of `abc` by the number 123, returning the result

[] object construction: causes an object to be found or created

`[species-type 'mapk]` ; creates an object of type `species-type`

{ } math syntax:

`{1 + 4}` ; adds 1 and 4, returning the number 5

`{:a + :b + :a}` ; creates the math-expression `{2 :a + :b}`

`{3 pascals * 3 liters}` ; must (include `b/units/standard`) to use these units

. field access: accesses fields of an object

`[compartment 'c].id` ; gets the value of the `id` field

`cytosol.(contains mapk)` ; calls the `contains` function in the `cytosol` object

' quote - prevents evaluation (code execution)

`'(* abc 123)` ; prevents the multiplication from occurring, returns a list instead

`'[x 1]` ; does not construct the object; returns the object form instead

`'{3 + 4}` ; does not add the numbers; returns the math form instead

Library modules: (`include (modules)`) to load library files; (`reset`) or (`init`) to clear definitions

b-user - standard system configurations (you will often include these)

`b-user/non-dimensional-ode-biochemistry` - Ordinary Differential Equations representing species in well-stirred compartments; kinetic constants and initial values are raw numbers

`b-user/3d-ode-biochemistry` - as above, but kinetic constants and initial conditions are specified in SI units - eg. `{3 katal / seconds}`, `{1 molar}`

b - main system library (you probably won't include these directly)

`b/biochem` - definitions for species, species-types, reaction, reaction-type, and other such things

`b/biochem/ode` - modifies the `biochem` objects so that ODE systems can be described

`b/matlab/ode-translation` - converts a system containing `ode-vars` to a set of `matlab` files

`b/math` - the mathematics subsystem (loaded automatically)

`b/units/standard` - defines standard SI units

b - main system library (you probably won't include these directly)

examples - several simple models demonstrating how to use little b forms

segment-polarity - multicompartment model of segment-polarity gene network,
contains a little b version of the `ingeneue` program

scaffold - generic module for describing scaffolding molecules, and models of `ksr`

Basic Commands:

(include *modules*) - loads one or more library modules:

(include b-user/non-dimensional-ode-biochemistry) ; include only a single module

(include (b-user/3d-ode-biochemistry
b/matlab/ode-translation)) ; include 2 modules

(reset) - resets the b-user workspace while retaining library definitions.

(init) - initializes little b to a “pristine” state. all library definitions are cleared from memory.

(in-package *pkg-name*) - used at the top of a file to set the namespace. You will almost always use:

(in-package :b-user)

(def-species-types *location-class &rest species-type-definitions*) - defines multiple species types

(def-species-types compartment (mapk :t0 {10 micromolar} (mek :t0 {1 micromolar}))

long-hand notation:

[species-type *&optional id*] - a location-independent molecular entity, which sits in a compartment, by default

(define mapk [[species-type] :t0 1]) ; mapk a species-type, which sits in compartments

(define receptor [[species-type] :location-class membrane]) ; this species sits in a membrane

{lhs -> rhs} - reaction-type in which left hand side (lhs) species are converted to right hand side species (rhs)

{enz + sub -> enz + prod} ; syntax generates object described above

{R + L @ :outer -> RL-cplx} ; receptor binds to ligand in the outer compartment to form a complex

{pump + ion @ :inner} @ membrane -> pump + ion @ :outer}

long-hand notation:

[reaction-type *lhs rhs &optional location-class*]

fields: set-rate-function, k, in

(define myrtype [reaction-type {enz + sub} {enz + prod}])

myrtype.(set-rate-function 'mass-action .3) ; myrtype will have mass-action kinetics, with mass-action constant .3

myrtype.k.mass-action ; access the mass-action rate constant

myrtype.(set-rate-function 'custom-rate {e * s ^ 2 / {km ^ 2 + s ^ 2}} :km .02) ; custom rate

(defmonomer *symbol &rest slot-definitions*) - defines a component which can join with others to form complexes

(defmonomer ksr a b c) ; ksr has 3 bond sites

(defmonomer mapk a (p :value (member :u :p))) ; mapk has one bond site (A) and one state site (P) for phosphorylation

[[ksr _ 1 _][mapk 1 :u]] ; ksr site B connected to mapk site A, mapk P site in state :U (“unphosphorylated”)

{lhs ->> rhs} - complex-reaction-type definition (uses patterns which describe complexes)

[[[ksr * 1 *][mapk 1 *]] ->> [ksr] + [mapk]] ; ksr bound to mapk (with anything bound to A and C sites) is unbound from mapk

[[[ksr 1 _ *][mapk 1 *]] ->> [ksr _ _ *]] ; mapk bound to ksr is destroyed

{[R _] + [L _] @ :outer ->> [[R 1][L 1]]} ; receptor binds to ligand in the outer compartment

(define-custom-rate *name (parameter-lambda-list)*

(*&key rate-dimension dictionary entities stoichiometries dimensions*)

&body body) - defines a custom rate function

(define-custom-rate hill-function (species *&key* (kd 1) (hill 1)) (:dictionary d)

{d.kd := (ensure-reference-var kd)} ; store constants in the dictionary

{d.hill := (ensure-reference-var hill)}

{species ^ :hill / {kd ^ :hill + species ^ :hill}}}

(define *symbol object*) - creates and names an object: assigns *object* to *symbol*, and advises the printing system that the object may be abbreviated as *symbol*. (examples below)

{*place := value*} - stores a value in a place. same as (setf *place value*)

{a := 1} ; now, the variable A holds the value 1

{[x 1].p := 2} ; now, property p of [x 1] holds 2

[compartment *&optional id*] - defines a simple location in which species can exist & reactions can occur

(define c1 [compartment]) ; names a compartment (symbol c1 is automatically assigned to ID slot)

fields: size

{c1.size.value := 100 microliters} ;

[membrane *&optional id*] - a location defining a membrane

(define m1 [[membrane] :outer cytoplasm :inner nucleus]) ; membrane with outer & inner compartments

[membrane-apposition *m1 m2*] - a location defining the apposition of two membranes

[enzymatic-reaction {*e*} {*s*} {*p*} *steps &optional location-class*] - set of reactions in which e converts s to p:

[enzymatic-reaction {e} {s1 + s2} {p} '(reversible :irreversible)] ; creates 3 rxns: e+s1+s2<->es->e+p

fields: fwd,rev,es,set-rate-function

myrxn.(set-rate-function 'mass-action :fwd '(1 2) :rev '(1.1)) ; sets the fwd and reverse mass-action rate constants

myrxn.fwd.2 ; name of the 2nd forward reaction-type,

myrxn.es.1 ; name of the first enzyme-substrate complex

[species *species-type location*] - a population of localized molecules. Usually created as follows:

`c1.(contains mapk) ; creates [species mapk c1]`

(create-ode-model *name*) - writes an ODE model to disk

(query *pattern*) - retrieves objects stored in the database

`(query species) ; returns a list of all species objects`

`(query [species ? c1]) ; returns a list of all species objects in compartment c1`

`(query [species mapk]) ; returns a a list of all mapk species in all compartments`

Advanced:

(defun *name (arg-list) &rest body*) - defines a lisp function:

`(defun average (x y) {{x + y} / 2}) ; now we can compute: (average 10 20) => 15`

(defmacro *name (arg-list) &rest body*) - defines a lisp macro

(defcon *name (options) fields &rest body*) - defines a concept class.

`(defcon x () (a)) ; now, [x 1] creates an object of type x, with field A = 1`

(defprop *class.field (options) &rest body*) - defines a property

`(defprop x.p ()) ; now we can set property :P, e.g, {[x 1].p := 1}`

(defield *class.field (arg-list) &rest body*) - defines a field method

`(defield x.times-a (val) {val * .a}) ; [x 3].(times-a 4) => 12`

(defrule *name lhs => rhs*) - defines a rule

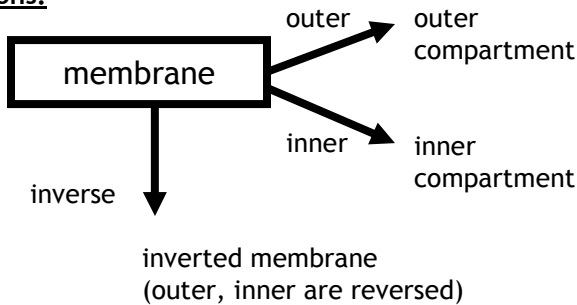
`(defrule detect-species [species ?type ?loc] ; prints a message whenever a new species is created
=> (format t "Species of type ~S created in ~S." ?type ?loc))`

Keyboard shortcuts:

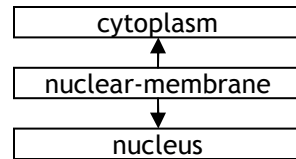
F6 evaluate all code in current editor tab
 Shift-F6 undefines all code in current editor tab
 F7 evaluate selected text
 Shift-F7 undefine selected text
 Ctrl-w close editor tab
 Ctrl-y find source code for selected symbol

Ctrl-. code help: complete symbol
 space code help: show function or object argument list
 . code help: show fields available for an object
 Ctrl-b find mismatched braces
 Ctrl-i auto-indent

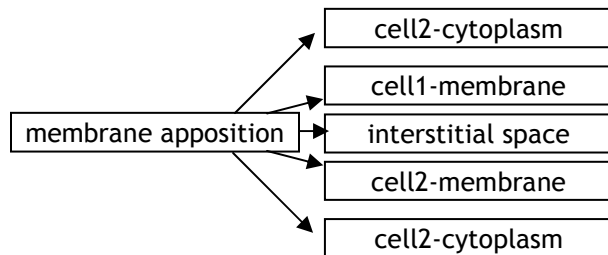
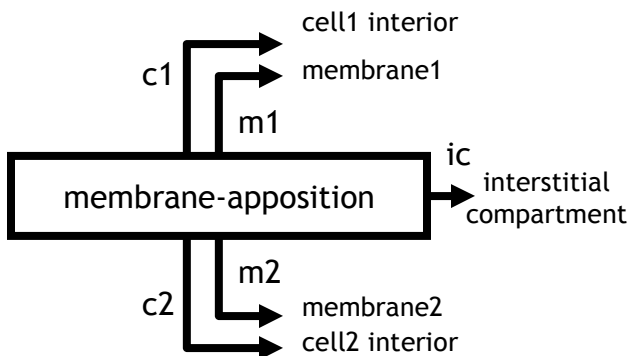
Locations:



example:



```
(define nuclear-membrane [membrane])
(define cytoplasm [compartment])
(define nucleus [compartment])
{nuclear-membrane.outer := cytoplasm}
{nuclear-membrane.inner := nucleus}
```



```
(define cell1 [spherical-cell])
(define cell2 [spherical-cell])
(define membrane-apposition
 [[membrane-apposition cell1.membrane cell2.membrane]
 :c1 cell1.inner :c2 cell2.inner])
```