The Beatbox Cardiac Simulation Software: a Primer

Plan

- Outline: how BeatBox works and how to use it
- BeatBox scripting language
- BeatBox devices
- Running BeatBox and interpreting the outputs

OUTLINE: HOW BEATBOX WORKS AND HOW TO USE IT

Example 1: the problem

Mathematical problem:

$$\frac{\partial u}{\partial t} = f(u,v) + D\nabla^2 u;$$
$$\frac{\partial v}{\partial t} = g(u,v);$$
$$(x,y) \in [0,L] \times [0,L]$$
$$t \in [0,T]$$

$$u(x,y,0) = u_0(x,y);$$
$$v(x,y,0) = v_0(x,y);$$
$$\left.\frac{\partial u}{\partial x}\right|_{x=0,L} = \left.\frac{\partial u}{\partial y}\right|_{y=0,L} = 0$$

Implementation:
Diffusion term,

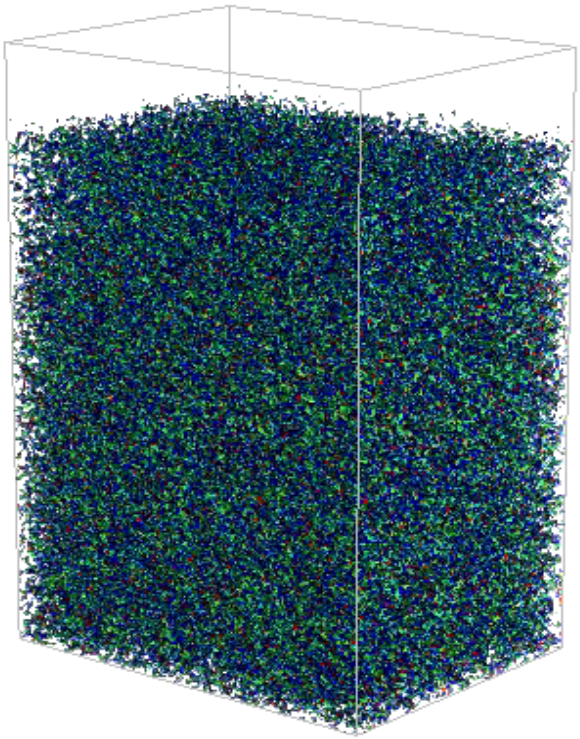
$$Iu(t) = D\nabla^2 u(t) \qquad \text{(central differences)}$$

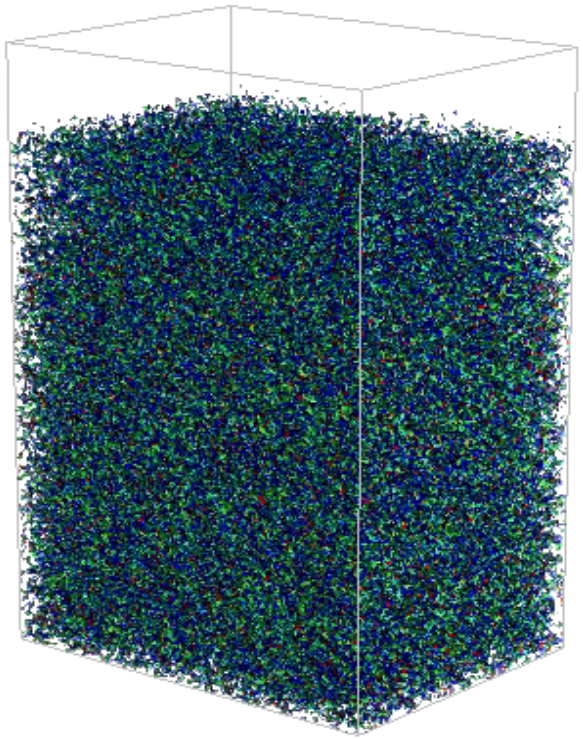
Time step: forward Euler,

$$u(t+\Delta_t) = u(t) + \Delta_t \left(f(u(t),v(t)) + I_u(t)\right)$$
$$v(t+\Delta_t) = v(t) + \Delta_t g(u(t),v(t))$$

$$f(u,v) = \frac{1}{\epsilon}\left(u - \frac{u^3}{3}\right) - v,$$
$$g(u,v) = \epsilon(u + \beta - \gamma v).$$

A more complicated example to have in mind





Low excitability,
Negative filament tension

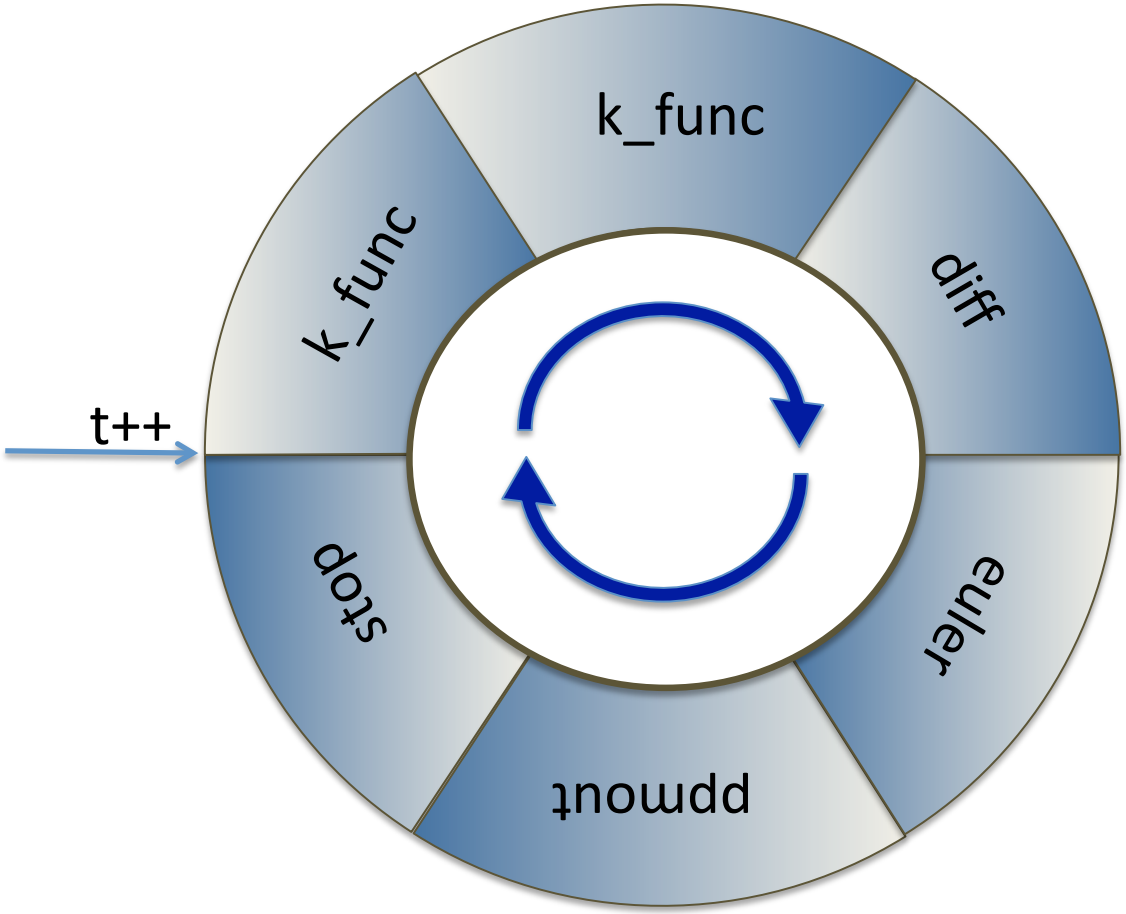
High excitability,
Positive filament tension

V.N. Biktashev, I.V. Biktasheva and N.A. Sarvazyan, ``Evolution of spiral and scroll waves of excitation in a mathematical model of ischaemic border zone'' *PLoS ONE*, **6**(9):e24388, 2011

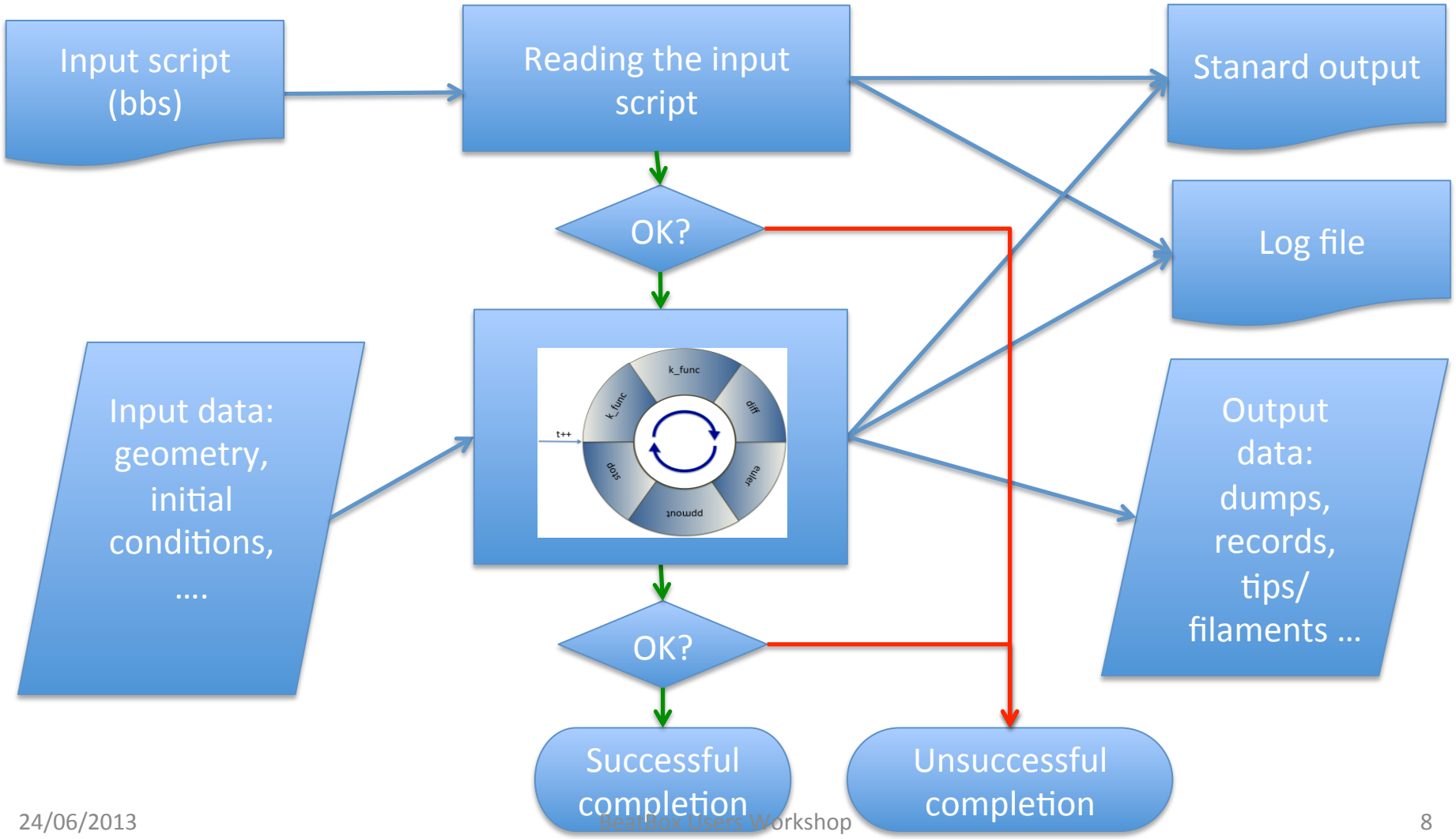
Motivation: Why BeatBox?

- Even in this simplest problem there are quite a few parameters that one might want to change.
- It is most convenient (and reliable!) to specify parameters by name=value method.
- Sometimes it is convenient to specify a parameter not by a value, but by a mathematical expression.
- There are so many parameters, it is convenient to group them by tasks. Say reaction separately, diffusion separately, stimulation protocol separately.
- Those parts will depend on the exact protocol of the numerical experiment.
- Some parameters are not numerical: method of approximation of the Laplacian, time stepper, kinetic model. Different kinetic models, or different solvers, use different sets of parameters.
- To rewrite the code for each new task is a recipe for a mess. So much nicer it would be to be able to do it all by changing the parameter file.

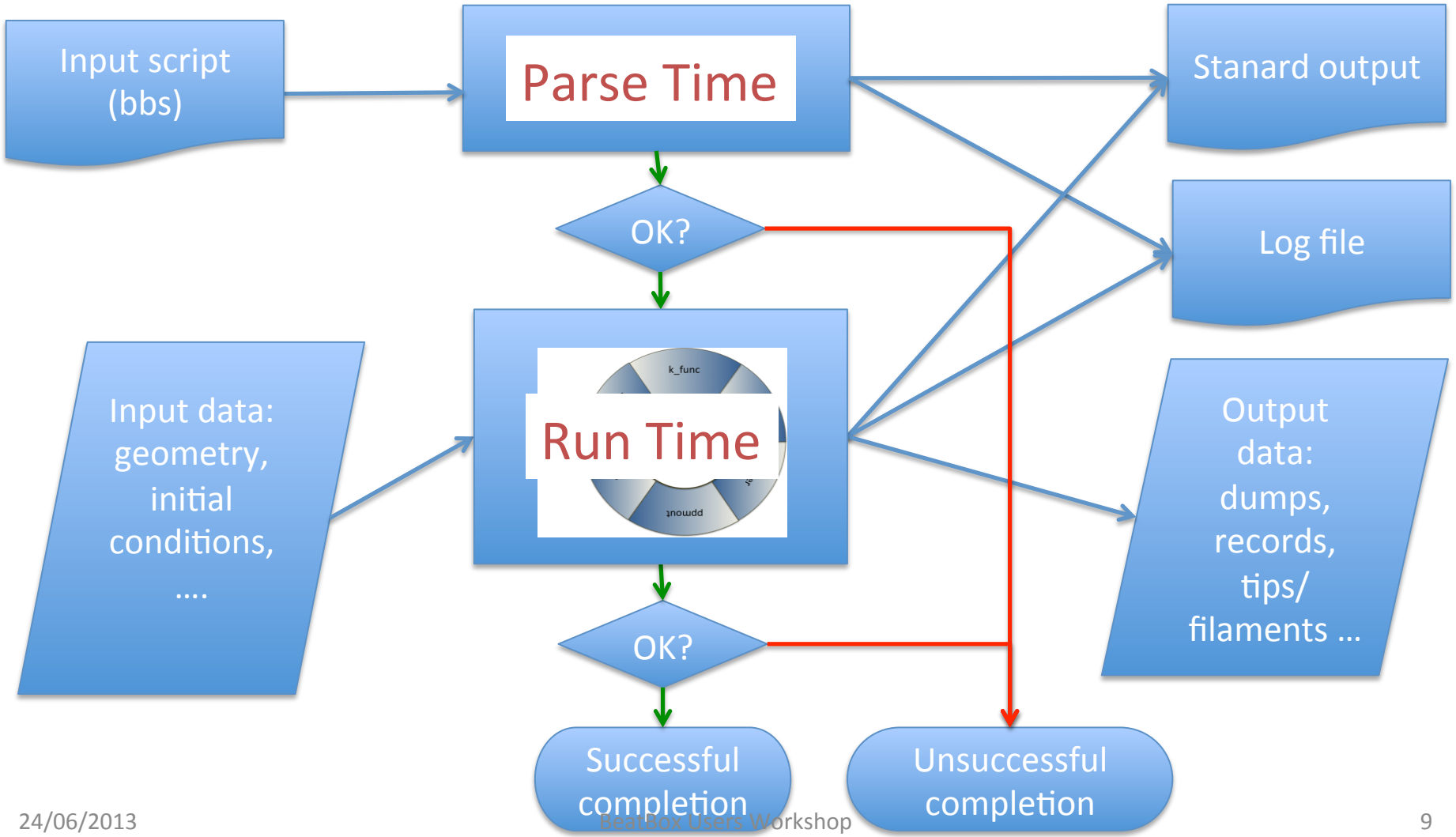
The paradigm: a ring of devices



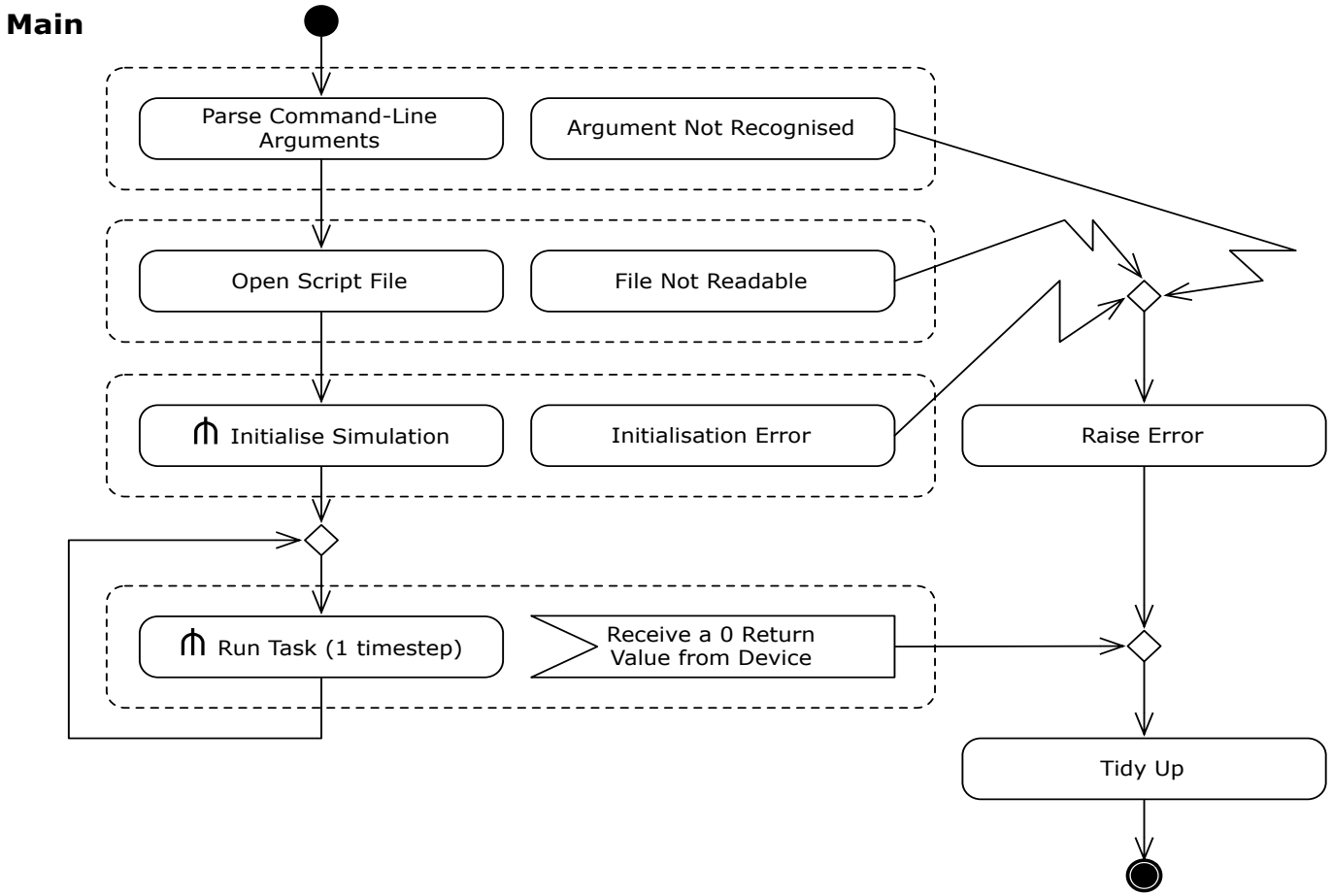
Life cycle of a Beatbox run



Life cycle of a Beatbox run

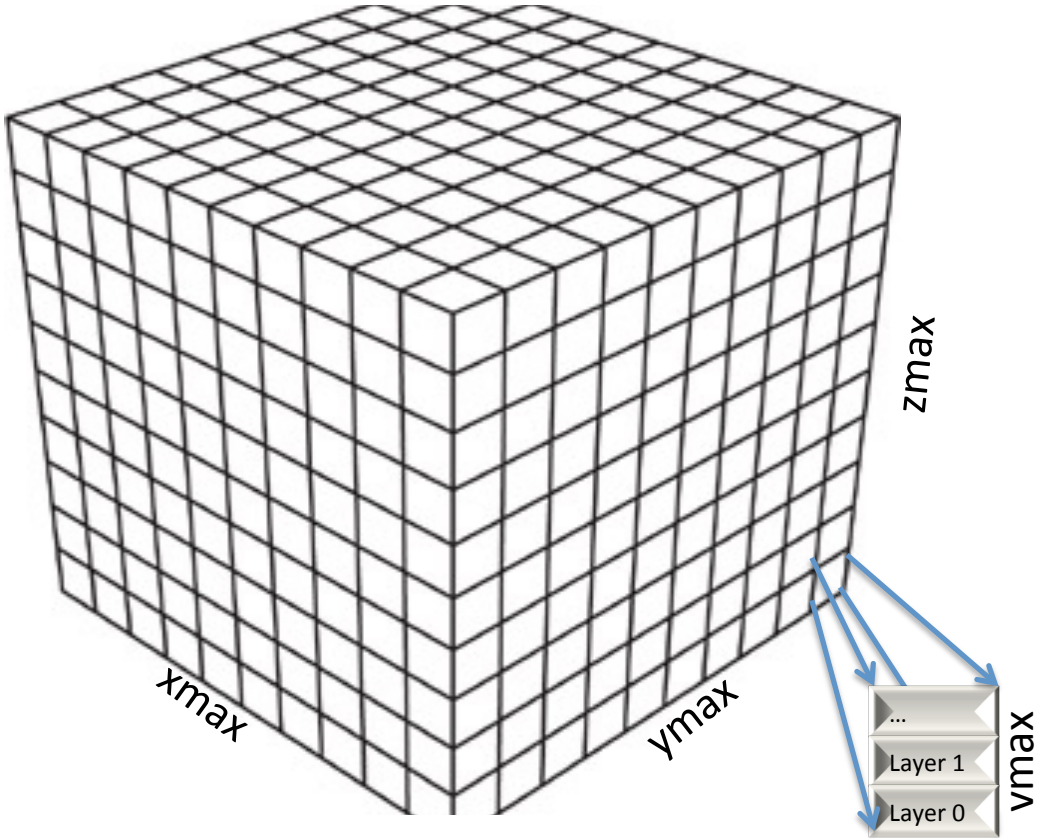


Top-level UML activity diagram
(thanks to R.McFarlane)



Beatbox data

The 4D grid



Individual variables

	Predefined	User-defined
String macros	[0], [1], [2]… (script's command-line parameters)	
Integer k-variables	t,xmax,ymax,.., Graph, BLACK, BLUE…	
Real k-variables	pi, e, inf, always, never, …	

Also: read-write or read-only; global or local.

Three sorts of devices

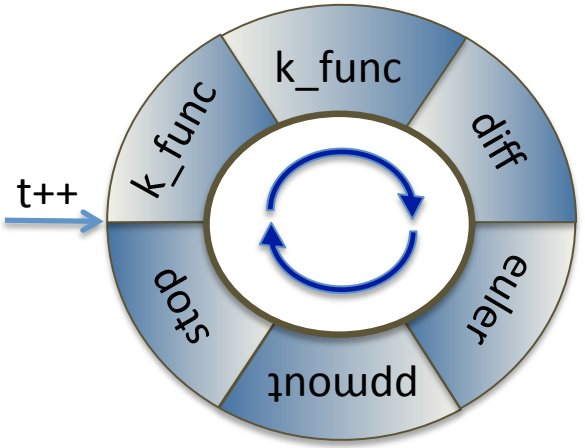
- **Computational**: ODE solvers implementing excitable kinetics; mono- and bidomain "diffusion" solvers; user-defined specialized solvers for pecular problems, …
- **Input/output**: initial conditions, control points, computation results
- **Control**: what device works when, …

Example 1 detail

- **k_func** (1st instance): as a control device, defines when other devices will be active: the second k_func at the start, ppmout every so often, and stop in the end.
- **k_func** (2nd instance): as a computational device defining the initial conditions
- **diff**: computational device, calculates the

diffusion term

- **euler**: computational device, performs the time step, using the diffusion term
- **ppmout**: output device, writes rounded-up simulation results to the disk for subsequent visualization
- **stop**: control device, terminates the run.

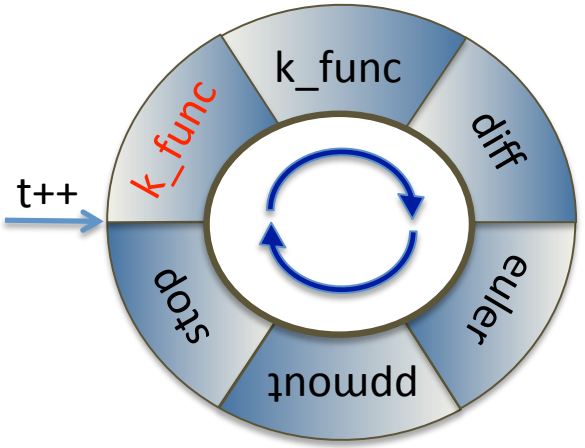


Example 1 detail

- **k_func** (1st instance): as a control device, defines when other devices will be active: the second k_func at the start, ppmout every so often, and stop in the end.
- **k_func** (2nd instance): as a computational device defining the initial conditions
- **diff**: computational device, calculates the

diffusion term

- **euler**: computational device, performs the time step, using the diffusion term
- **ppmout**: output device, writes rounded-up simulation results to the disk for subsequent visualization
- **stop**: control device, terminates the run.

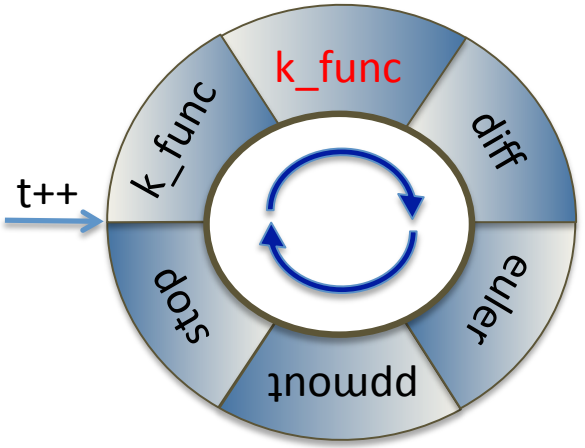


Example 1 detail

- **k_func** (1st instance): as a control device, defines when other devices will be active: the second k_func at the start, ppmout every so often, and stop in the end.
- **k_func** (2nd instance): as a computational device defining the initial conditions
- **diff**: computational device, calculates the

diffusion term

- **euler**: computational device, performs the time step, using the diffusion term
- **ppmout**: output device, writes rounded-up simulation results to the disk for subsequent visualization
- **stop**: control device, terminates the run.

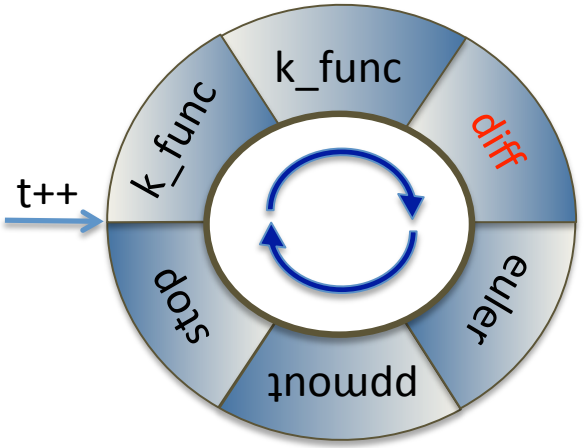


Example 1 detail

- **k_func** (1st instance): as a control device, defines when other devices will be active: the second k_func at the start, ppmout every so often, and stop in the end.
- **k_func** (2nd instance): as a computational device defining the initial conditions
- **diff**: computational device, calculates the

diffusion term

- **euler**: computational device, performs the time step, using the diffusion term
- **ppmout**: output device, writes rounded-up simulation results to the disk for subsequent visualization
- **stop**: control device, terminates the run.

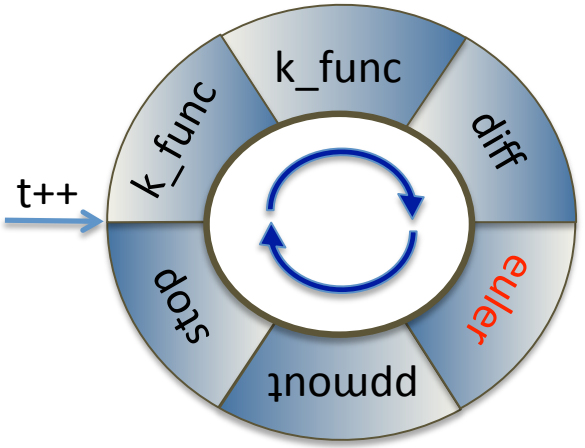


Example 1 detail

- **k_func** (1st instance): as a control device, defines when other devices will be active: the second k_func at the start, ppmout every so often, and stop in the end.
- **k_func** (2nd instance): as a computational device defining the initial conditions
- **diff**: computational device, calculates the

diffusion term

- **euler**: computational device, performs the time step, using the diffusion term
- **ppmout**: output device, writes rounded-up simulation results to the disk for subsequent visualization
- **stop**: control device, terminates the run.

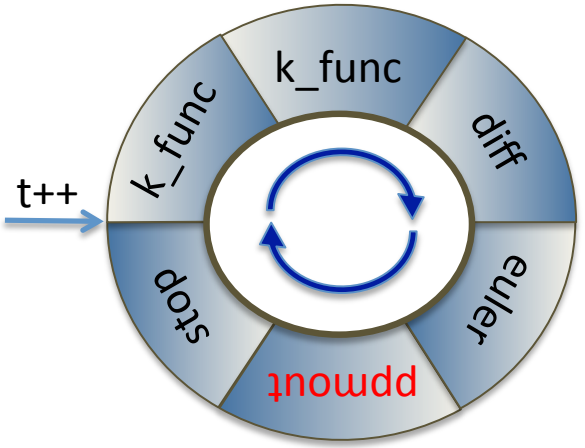


Example 1 detail

- **k_func** (1st instance): as a control device, defines when other devices will be active: the second k_func at the start, ppmout every so often, and stop in the end.
- **k_func** (2nd instance): as a computational device defining the initial conditions
- **diff**: computational device, calculates the

diffusion term

- **euler**: computational device, performs the time step, using the diffusion term
- **ppmout**: output device, writes rounded-up simulation results to the disk for subsequent visualization
- **stop**: control device, terminates the run.

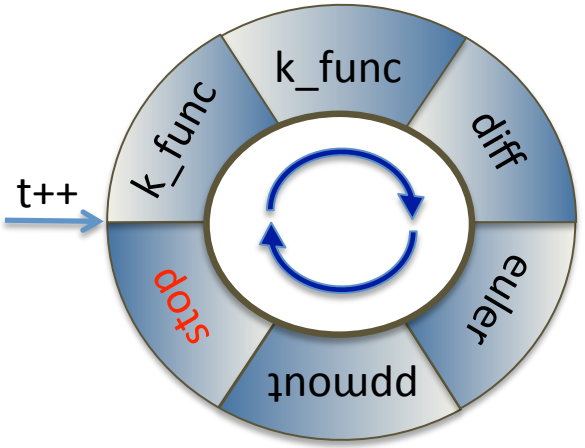


Example 1 detail

- **k_func** (1st instance): as a control device, defines when other devices will be active: the second k_func at the start, ppmout every so often, and stop in the end.
- **k_func** (2nd instance): as a computational device defining the initial conditions
- **diff**: computational device, calculates the

diffusion term

- **euler**: computational device, performs the time step, using the diffusion term
- **ppmout**: output device, writes rounded-up simulation results to the disk for subsequent visualization
- **stop**: control device, terminates the run.



BEATBOX SCRIPTING LANGUAGE

Our first beatbox script: “minimal.bbs”

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

Launching BeatBox: sequential mode

> Beatbox_SEQ minimal.bbs

This is the simplest form, fully adequate for our first example.

More general form is

> Beatbox_SEQ minimal.bbs par1 par2 … -opt1 –opt2 …

(command-line parameters and options can in fact go in any order, and some options require a word after them – example will follow).

Beatbox scripting language: 1st approx

// comment to the end of line as in C++
/* comment between slashes and stars as in C */
command1 … /*comment inside command */ ;
command2 …
…
; //comment outside command
rem command is another form of comment;
end; // this must be present!
Whatever goes after the end is ignored.

NB: linebreaks are not important; semicolons are!

bbs commands

- **def**: defines and initiates a string macro or k-variable. Not necessary syntactically but you do need k-variables even if only to decide when to stop.
- **state**: defines the computational grid. Must be present exactly once before any device commands.
- **device defining commands**: there may be many instances of the same device, can be distinguished by "names".
- **end**: must be the last command in the script. Anything after it is ignored.

def command

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

def command

- Defining a k-variable:

def <type> <name> [[=] <value>];

where <type>=int,long,real,float,double (int=long,real=float=double=precision chosen at compile time). Equal sign = is optional; if value is missing, 0 or 0.0 is assumed. The <value> could be a constant, or an arithmetical expression, possibly depending on k-variables already defined.

- Defining a string macro:

def str <macro name> <value>;

(details of this later)

Some predefined functions for expressions

- atan2(a, b) - Returns the arctangent of a/b, using the signs of the arguments to compute the quadrant of the return value.
- hypot(a, b) - Returns the square root of the sum of the squares of a and b.
- if(test, then, else) - Returns then if test evaluates to nonzero, else otherwise.
- ifne0(test, then, else) - Returns then if test is not equal to 0, else otherwise. Similarly: ifeq0, ifgt0, ifge0, iflt0, ifle0.
- gt(a, b) - Returns 1.0 if a is strictly greater than b. Similarly: ge, lt, le, eq, ne.
- mod(a, b) - Returns the remainder of the integer division of a by b.
- max(a, b)/min(a, b) - Returns the greater/smaller of a or b.
- crop(test, min, max) - Returns min if test<min, returns max if test>max, returns test otherwise.
- rnd(a, b) - Returns a random real number in [a,b) uniformly distributed in between (sequential version only).
- gauss(a, b) - Returns a random real number normally distributed with mean a and standard deviation b (sequential version only).
- u(x,y,z,v) - Returns the value at layer v at point (x,y,z) in the grid. If non-integer values are given for x,y,z, the value is (bi-,tri-)linearly interpolated.

def command

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

Device defining commands

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

Device defining commands

<device> [<name>=<value> [...]] ;

- The <name>=<value> pairs define the device parameters and are separated from each other by spaces.
- Each device has its own set of parameters, whose names will be checked for in the body of the command.
- An <name>=<value> pair with unknown name, or anyting that is not a <name>=<value> pair, is ignored.
- If more than one <name>=<value> pair with the same name, only the first one matters.
- If no <name>=<value> pair for a device parameter is found, the default value is assigned, or it is an error if there is no default for this parameter.
- Device parameters are assigned *at parse time.*

Device defining commands

<device> [<name>=<value> [...]] ;

- Parameters could be arithmetic (integer or real), string, or blocks.
- For arithmetic parameters, <value> is from the = sign until the first blank, intepreted as an arithmetic expression, evaluated to the appropriate type (integer or real), and assigned to the parameter.
- A parameter may have maximal and minimal allowed values. If the given value is beyond, it is an error.
- For string parameters, the value is taken as the literal value, subject to preprocessing. Double quotes "..." can be used for values with special characters, with use of \ character as in C.
- Block parameters have values as text enclosed by curly brackets, {...}. Such block text may span several lines and will usually contain its own set of <name>=<value> pairs.

Device defining commands

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

state command

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

state command

state xmax=<value>…vmax=<value>;

state geometry=<file.bbg> vmax=<value> …;

- Syntax similar to device defining command, but does not add device to the ring
- If geometry is given, the enclosing box size is calculated automatically.
- If no geometry is given, box geometry is presumed and its sizes can be defined explicitly.
- xmax,ymax,zmax default to 1.
- vmax defaults to 2 (as in FitzHugh-Nagumo).
- zmax=ymax=xmax=1 means 0D problem, otherwise zmax=ymax=1 means 1D problem, otherwise zmax=1 means 2D problem, otherwise 3D problem.
- xmax=2 etc are allowed but hardly ever used.

state command

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
…
```

- In our example, xmax=102, ymax=102, which means that the integer coordinates can have values x=0..101 and y=0..101 inclusive.
- Parameter zmax is not specified so defaults to 1. Hence only z=0 is allowed. So our grid is 2-dimensional.
- Parameter vmax=3 means that there are three layers. Layers v=0 and v=1 will be used to store the values of the fields *u* and *v* of the FitzHugh-Nagumo model, and layer v=3 is required to store the value of the diffusion term.
- Note that the designation of layers is not syntactically fixed, and it is up to the user to remember which layer is for what.

end command

```
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

- Does “what it says on the tin”: designates the end of the script.
- Its purpose is to put a check on integrity of the script: if an incomplete script is accidentally launched, it may lead to time waste or other unwanted consequences.

BEATBOX DEVICES

Generic device parameters

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

Generic device parameters

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
```

(str) name

- Given name for the device.
- If the script has more than one instance of a device, the names help to disambiguate the devices in output messages.
- Some devices refer to another device in the simulation by name.
- The default for name is the device name.

Generic device parameters

```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
```

(real) when

- The device's condition, a real k-variable (not a literal real number!).
- NB: exceptionally, only the *run-time* value of this parameter is relevant!
- The device will be run only on timesteps when this variable is nonzero.
- Defaults to always, read-only k-variable which always equals 1.0.

Generic device parameters

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
```

(int) x0, x1, y0, y1, z0, z1

- Limits of a subgrid associated with the device. Typically, the device will operate on points with x0<=x<=x1 etc.
- Defaults to all inner points of the grid, excluding “halo points”, discussed later (precise definition depends on dimensionality).

Generic device parameters

```
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
```

(int) v0

- Lowest layer associated with the device (exact semantics of that depends on the device type). Defaults to 0.

(int) v1

- Highest layer associated with the device (ditto). Defaults to v0.

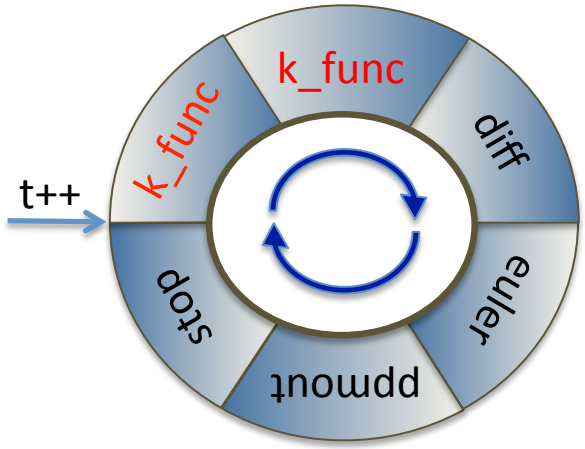
Generic device parameters

```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
```

(int) nowhere

- If 0 (the default), then x0, … z1 are accepted and used, and this instance of the device is associated with the corresponding subgrid.
- If 1, then it is an error to specify x0, … z1, and this instance of the device will not be associated with any particular points of the grid.

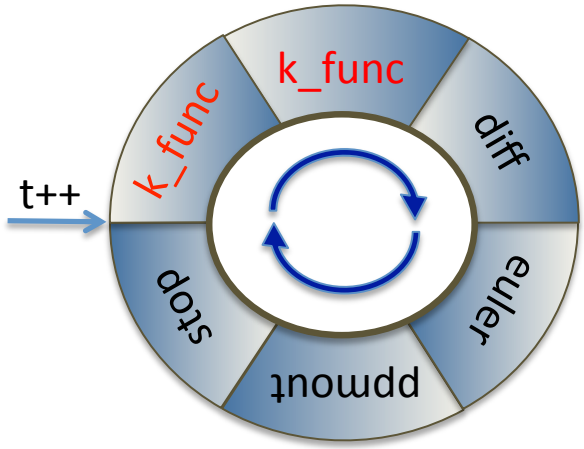
k_func device




```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
```

- This is a generic computational device. Its function is to make computations on *k*-variables and/or grid points, according to given "k-program". "k-" or "k_" stands for the inbuilt compiler of arithmetic expressions, written by Alexander Karpov.

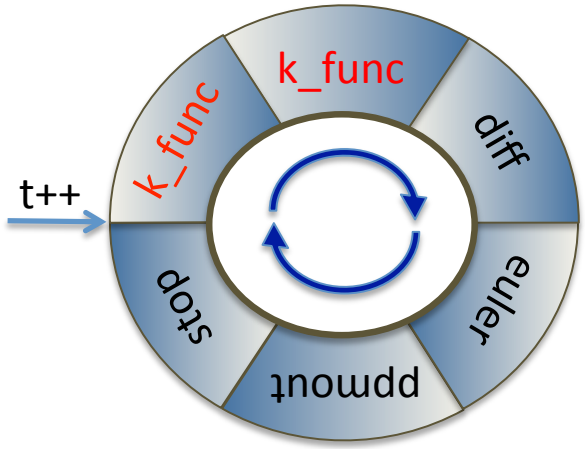
k_func device




```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
```

- The key parameter: block pgm={…}, defines computations to be done at every time step. These computations are defined by a different syntax than other device parameters, which is why their description if is enclosed in a {…} block.

k_func device



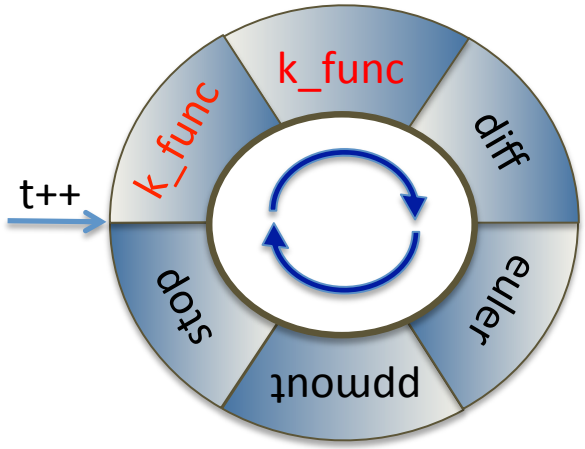

```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
```

- The computations are defined by

<name>=<expression>

pairs, separated by semicolons ';'. The names are any k-variables that are defined up to now (as opposed to a fixed list of parameters which a particular device type understands)

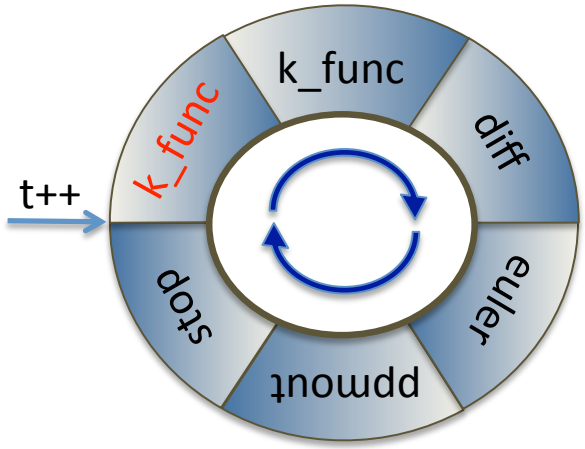
k_func device




```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
```

- NB: the assignments given in a k_func program are executed at *run time*, as opposed to the assignments done for (most of the) device parameters which are done at parse time. This is a yet another reason to enclose them in a {…} block: they are very different!

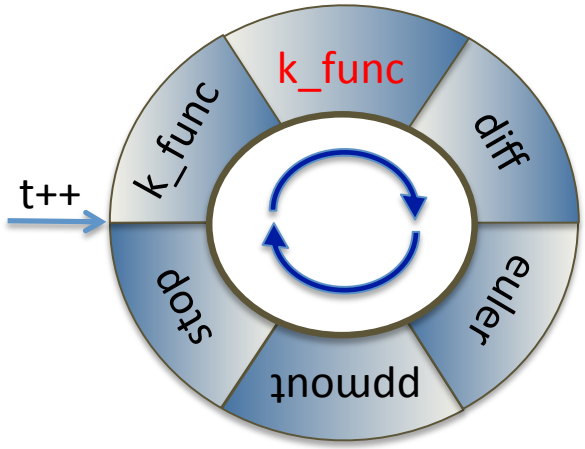
k_func device




```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
```

If nowhere=1, then the computations are done only once per time step.

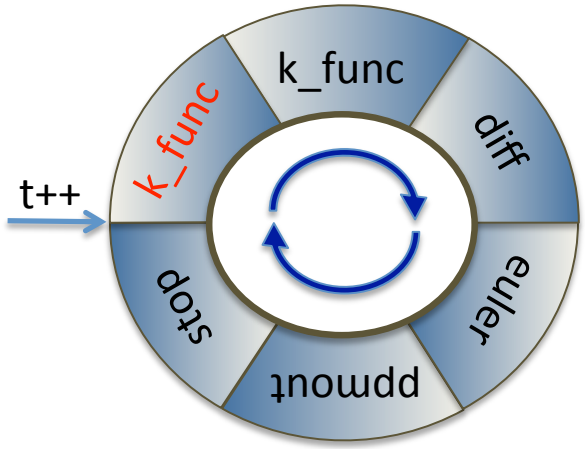
k_func device




```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
```

Otherwise, the computations are done for every point of the devices ‘space’, i.e. the subgrid. Then *local variables* are defined: x,y,z for integer coordinates of the grid point (read-only), and u0, u1, … for the real values stored in the layers of the grid point (read/write).

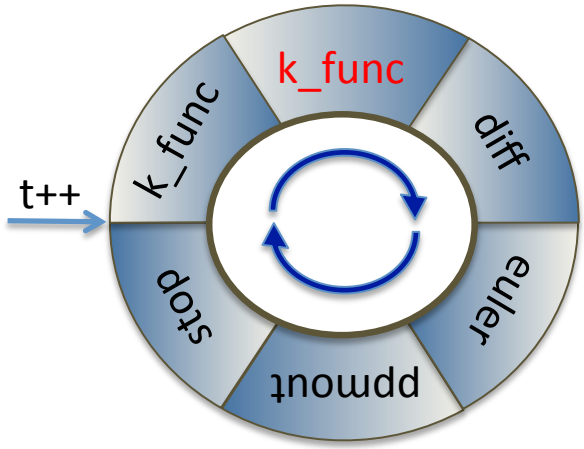
k_func device




```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
```

In our simple script, the first instance of the k_func device has a program that executes only once per timestep. It assigns value 1 to k-variable begin if the timestep counter t=0 (otherwise assigns 0), then assigns 1 to k-variable output if t is divisible by 100 (otherwise 0), and then assigns 1 to k-variable end if t equals or exceeds 2000. NB these k-variables are defined by def commands before used.

k_func device

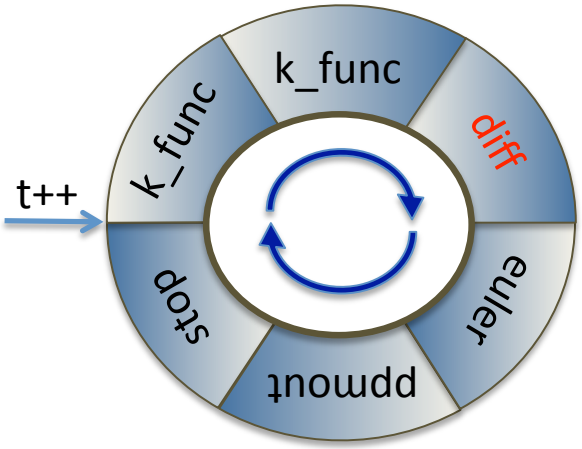



```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
…
```

In the second instance, nowhere is not given, so defaults to 0. Hence the program repeats for every point of the subgrid [1:100]x[1:100] (later it is explained why *this* subgrid).

It puts into layer 0 values -1.7 for points in the left half, +1.7 in the right half, and also puts into layer 1 values -0.7 in the lower half and +0.7 in the upper half.

diff device



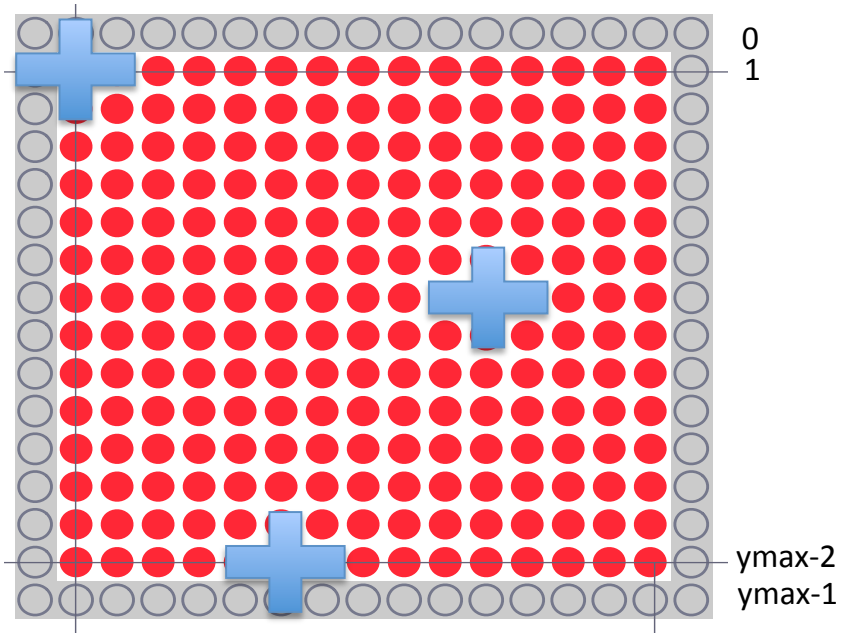

```
...
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
...
```

- This device computes the Laplacian of the field in layer v0=0, assuming space step of hx=0.5, multiplies it by the diffusivity D=1.0, and puts the result into layer v1=2.
- Different approximation of Laplacian are required for 2D or 3D grids, for boxes or realistic geometry, for isotropic and anisotropic media. This device detects it all automatically.

diff device and halo points

```
state xmax=102 ymax=102 vmax=3;
…
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
…
```

$$\left[\nabla^2 u\right]_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{\Delta_x^2}$$

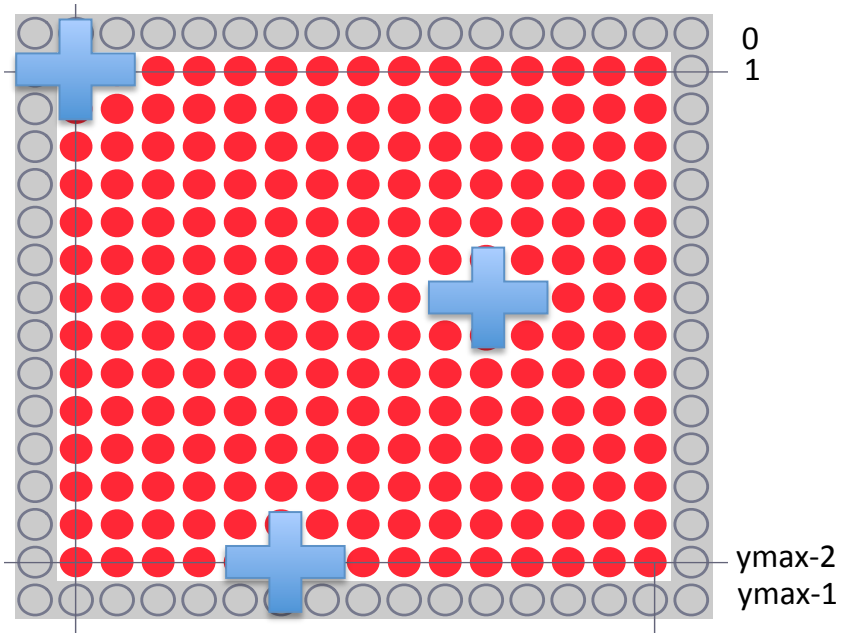


- diff device uses the same template for all points including marginal ones. The templates use only the nearest neighbours, hence one-point thick outermost layers of the grid are normally excluded from operation. They are *halo* points.

diff device and halo points

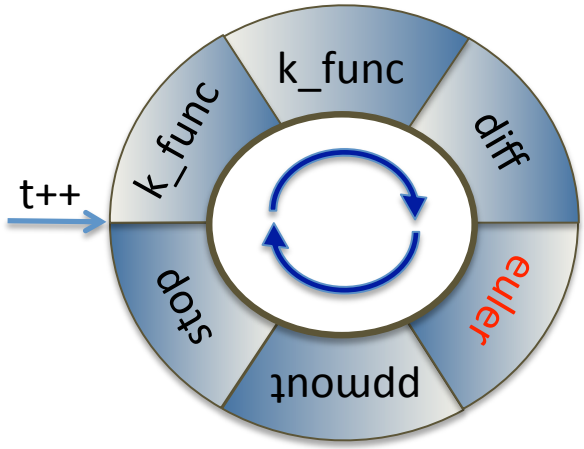
```
state xmax=102 ymax=102 vmax=3;
…
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
…
```

$$\left[\nabla^2 u\right]_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{\Delta_x^2}$$



- In this example, x=0, x=101 and y=0, y=101 are halo strips, which is why euler device is given space of inner points x=1..100, y=1..100.
- In fact, this could have been omitted, since inner points are the default for the generic space parameters.

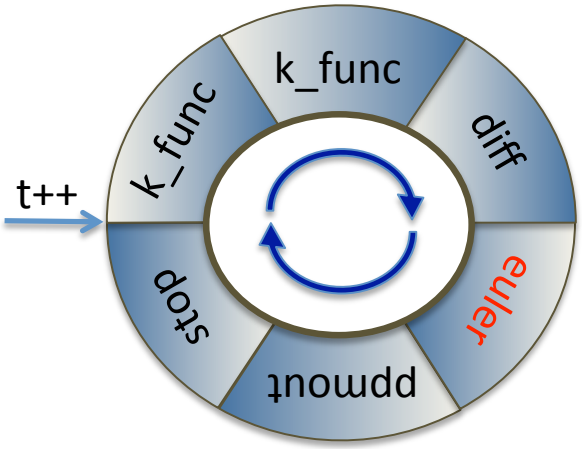
euler device




```
...
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
    par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
...
```

- This device updates values in all points within its space, according to forward Euler scheme.

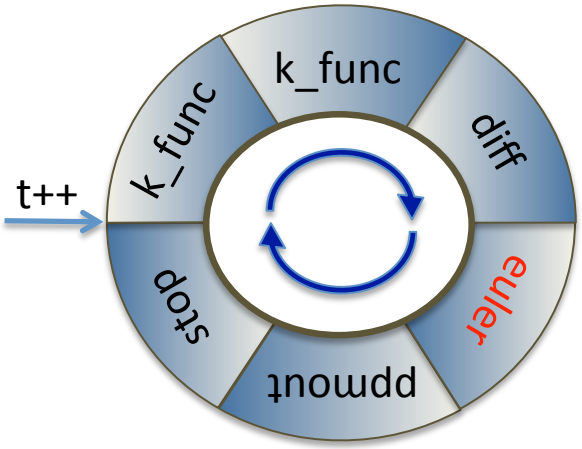
euler device




```
…
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
    par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
…
```

- It assumes the layers to be updated are located in layers from v0=0 through to layer v1=1.

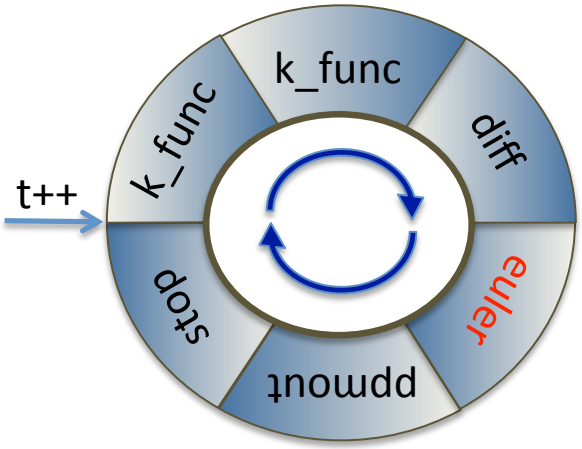
euler device




```
...
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
...
```

The time step in the forward Euler step is defined by device parameter ht=0.03.

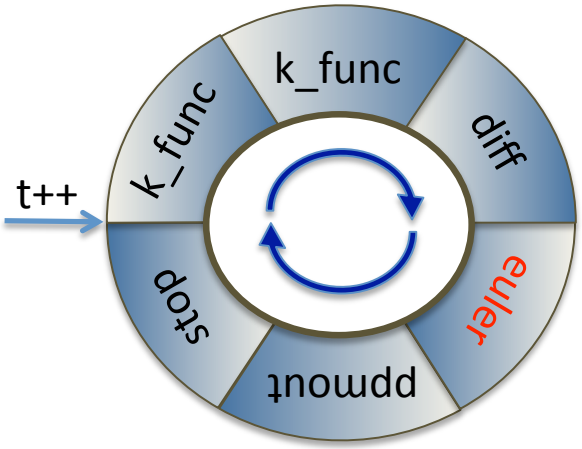
euler device




```
...
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
...
```

- The cell model, i.e. the right-hand sides of the systems of ordinary differential equations (ODE), is defined by name. The name ode=fhncub stands for FitzHugh-Nagumo cubic (there is also a piecewise-linear version, and Barkley version).
- The list of available cell models and their meaning can be found in the Beatbox manual (and in the source codes, of course).

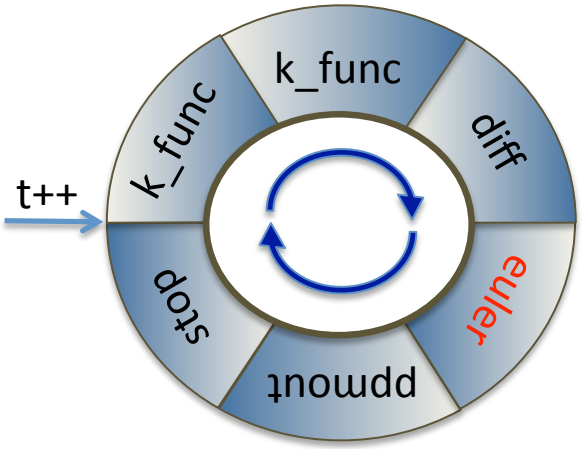
euler device




```
...
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
    par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
...
```

- The cell models can use default values of parameters, or some of them can be modified. Since the list of parameters is different for every model, these are not among euler device parameters. Besides, the same cell model definitions can be used in other, more sophisticated solvers (under development). This is why the list of model parameters has to be enclosed in a {...} block.

euler device




```
...
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
    par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
...
```

- In our case, the first three parameters specified eps, bet, gam which correspond to the epsilon, beta and gamma parameters in Winfree 1991 definition in an obvious way.
- The <name>=<value> pairs here are separated by spaces (just as with device parameters).

“Layer substitution”: a special feature of cell model definition syntax

```
…
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
    par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
…
```

- The right-hand side of the definition of parameter Iu has a special form. The meaning of @<n> is “take the value stored at the same point of the grid, from layer <n> (in this case, from layer 2).
- NB: this operation “take value from layer 2” will be done at run time, not at parse time!
- This is a generic way through which computations are done, where parameters vary in space and/or in time.

“Layer substitution”: a special feature of cell model definition syntax

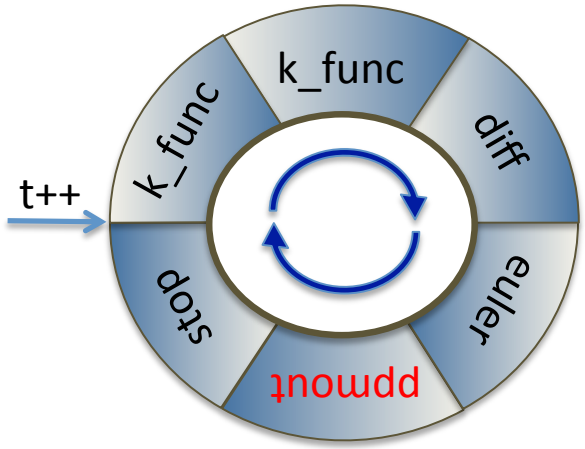
```
…
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
    par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
…
```

- In this example, parameter Iu is simply an extra additive term in the equation for the u-variable (the activator) in Winfree 1991 definition of cubic FitzHugh-Nagumo:

$$du/dt = (u-u^3/3)/epsilon – v + Iu$$

- Notice this is the place in which the diffusion term appears in the reaction-diffusion version of the equation.
- Notice that layer 2 contains the value of the diffusion term!

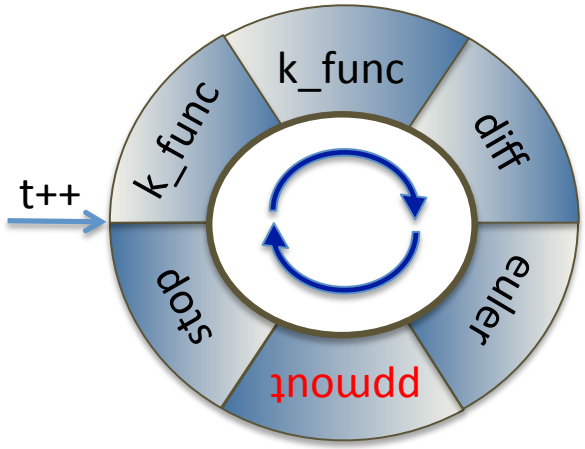
ppmout device




```
...
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
    par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

- This is the only output device in our example. It provides outputs in ppm format, which is an easy to write and interpret format, part of netpbm package available for all platforms under GNU license.
- We use a non-standard 3D extension of this format, but it obviously does not apply to our current simple example which is 2

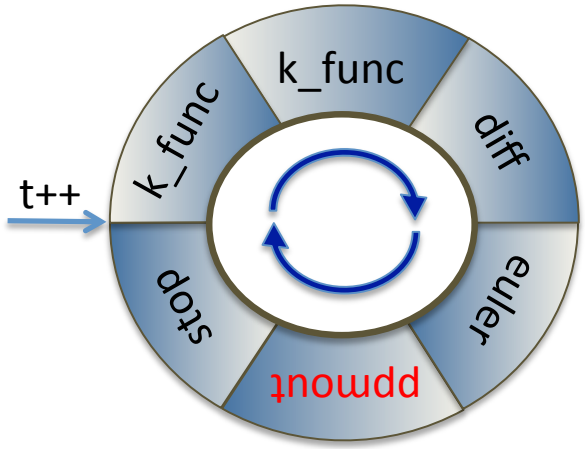
ppmout device




```
...
state xmax=102 ymax=102 vmax=3;
...
par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

- In this example, space parameters x0...z1 are not given so default will be taken, which is all *inner* points of the medium. So by defaule we will have x0=1, x1=100, y0=1, y1=100, z0=0, z1=0.
- The ppmout device is very straightforward in forming output images: each point from its space will generate a pixel in the output image file. Hence this instance of ppmout will produce 100x100 images.

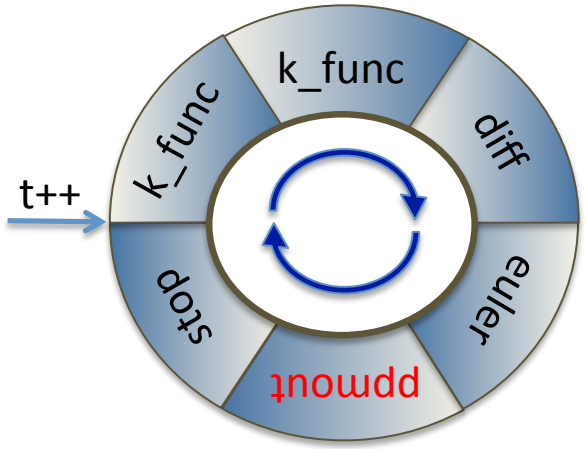
ppmout device




```
...
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
...
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
...
```

- Parameter when is specified, so this instance of ppmout will only run on timesteps at which k-variable output will be nonzero. According to the program of the first instance of the k_func device, this will happen once in 100 steps.
- NB the check of the value of the output variable will be done at *run-time*: in this way the treatment of when parameter is different from all other device parameters.

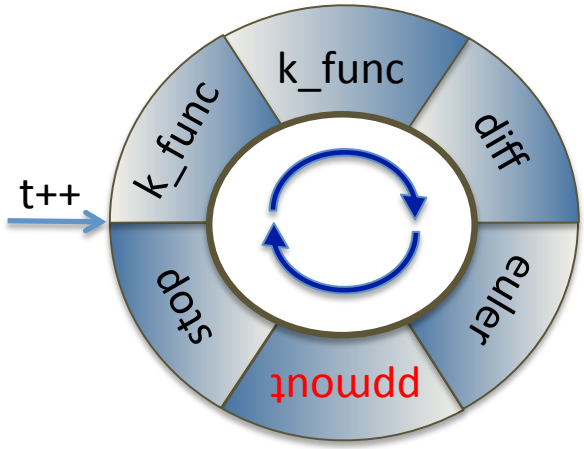
ppmout device




```
...
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
    par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

- Device parameter file defines the name(s) of the output file(s).
- In out example, the value of this parameter contains %. This means this value is not the name itself, but a “template”, used as a C format string, for producing enumerated names. In our example, these will be 0000.ppm, 0001.ppm, 0002.ppm etc .
- If file value does not contain %, the output file will have the fixed name and will be overwritten every time ppmout runs.

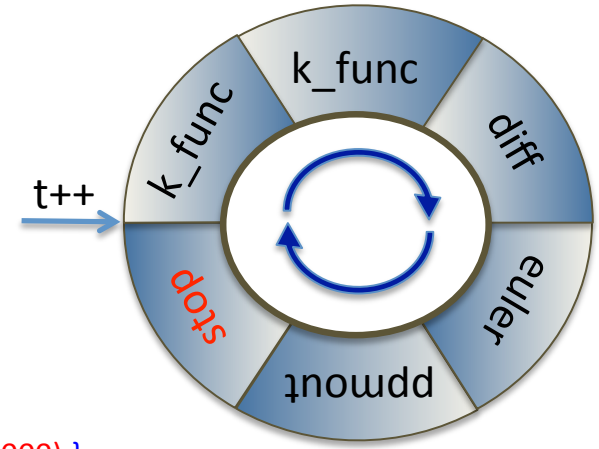
ppmout device




```
...
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
    par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

- The ppm is a colour format, and every pixel has red, green and blue components. The red component is formed from the value in the layer 0 since r=0, grid values r0=-1.7 or below will produce pixel red component 0, grid values r1=1.7 and above will produce pixel value 255, and anything in between will be linearly scaled.
- Likewise, green values are formed from layer g=1, and blue values from layer b=2, with similar meanings of g0, g1, b0, b1.
- How to see the resulting images: see the end of this presentation.

stop device




```
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
…
stop when=end;
end;
```

- "Does what it says on the tin": when this device engages, it means means termination of the beatbox run.
- Just one parameter: the condition. In our example, this is the k-variable end. According to the program of the first k_func device, this will be nonzero as soon as the main loop counter t reaches the value of 2000.

Review: “minimal.bbs”

```
state xmax=102 ymax=102 vmax=3;
/* device control variables */
def real begin; def real output; def real end;
/* Schedule */
k_func name=schedule nowhere=1 pgm={ begin =eq(t,0); output=eq(mod(t,100),0);end=ge(t,2000);};
/* Initial conditions */
k_func name=ic when=begin  x0=1 x1=100 y0=1 y1=100
      pgm={u0=-1.7+3.4*gt(x,50);u1=-0.7+1.4*gt(y,50);};
/* Diffusion substep */
diff v0=0 v1=2 D=1.0 hx=0.5;
/* Reaction substep */
euler v0=0 v1=1 ht=0.03 ode=fhncub
     par={eps=0.2 bet=0.8 gam=0.5 Iu=@2};
/* Output image files */
ppmout when=output file=%04d.ppm r=0 r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0.0  b1=1.0;
stop when=end;
end;
```

RUNNING BEATBOX AND INTERPRETING THE OUTPUTS

Running minimal.bbs

```
565 13:27:21 vnb$ Beatbox_SEQ minimal.bbs
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop $
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=======================================================
566 13:27:26 vnb$
```

Command line: the name of the executable and the name of the script. This can also take contain options and command-linearguments (considered later)

Running minimal.bbs

```
565 13:27:21 vnb$ Beatbox_SEQ minimal.bbs
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop $
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=====================================================
566 13:27:26 vnb$
```

Version of the executable reported. This is important for troubleshooting.

Running minimal.bbs

```
565 13:27:21 vnb$ Beatbox_SEQ minimal.bbs
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-------------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop $
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=====================================================
566 13:27:26 vnb$
```

Timing of the run.

Also, which of the devices caused the termination of the run. In this case this is device stop which is expected and normal.

Running minimal.bbs

```
565 13:27:21 vnb$ Beatbox_SEQ minimal.bbs
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop $
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=====================================================
566 13:27:26 vnb$
```

Reports the name of the script, arguments and options. Currently all options are default.

Running minimal.bbs

```
565 13:27:21 vnb$ Beatbox_SEQ minimal.bbs
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop $
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=======================================================
566 13:27:26 vnb$
```

Then go reports of the state and device commands, starting with state which reports the grid size, in the format xmax x ymax x zmax x vmax. This is a 2D run since zmax=1.

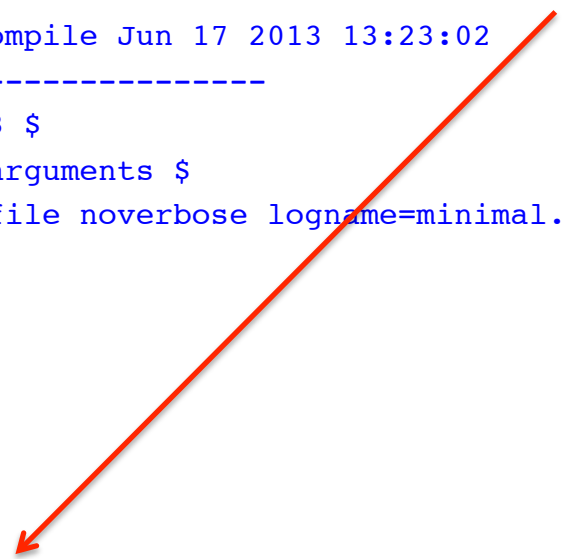
Running minimal.bbs

```
565 13:27:21 vnb$ Beatbox_SEQ minimal.bbs
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop $
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=====================================================
566 13:27:26 vnb$
```

The reports on other devices are even more concise. Still, they are needed: a failure at parse-time is easier to diagnose when you know in which device it happened.

Running minimal.bbs

```
565 13:27:21 vnb$ Beatbox_SEQ minimal.bbs
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop$
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=======================================================
566 13:27:26 vnb$
```



Summary of the devices created. Check this out in case of any unexpected behaviour: missing a semicolon in the input script means omitting the whole device!

Running minimal.bbs

```
565 13:27:21 vnb$ Beatbox_SEQ minimal.bbs
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop$
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=======================================================
566 13:27:26 vnb$
```

Messages from devices and run-time debug information would go here. In this example, only the stop device reports.

Understanding minimal.log

```
568 14:11:10 vnb$ cat minimal.log
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-----------------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose
logname=minimal.log
state xmax=102 ymax=102 vmax=3 /* Grid size (102 x 102 x 1 x 3) */$
k_func nowhere=1 name="schedule" pgm={} $
k_func when="begin" x0=1 x1=100 y0=1 y1=100 name="ic" pgm={} $
diff v0=0 v1=2 hx=0.5 D=1 $
euler v0=0 v1=1 ht=0.03 ode="fhncub" par={eps=0.2 bet=0.8 gam=0.5
Iu=@2 } $
ppmout when="output" file="%04d.ppm" file="%04d.ppm",0="0000.ppm" r=0
r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0 b1=1 $
stop when="end" $
end of input file $
Loop of 6 devices created:
   (0)schedule (1)ic (2)diff (3)euler (4)ppmout
   (5)stop $
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds.
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=================================================
569 14:11:15 vnb$
```

As you can see, in the present example the log file is almost a copy of the standard output, with some more detail.

Understanding minimal.log

```
568 14:11:10 vnb$ cat minimal.log
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-----------------------------------------------------------------
execution begin at Mon Jun 17 13:27:24 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile noverbose
logname=minimal.log
state xmax=102 ymax=102 vmax=3 /* Grid size (102 x 102 x 1 x 3) */ $
k_func nowhere=1 name="schedule" pgm={} $
k_func when="begin" x0=1 x1=100 y0=1 y1=100 name="ic" pgm={} $
diff v0=0 v1=2 hx=0.5 D=1 $
euler v0=0 v1=1 ht=0.03 ode="fhncub" par={eps=0.2 bet=0.8 gam=0.5
Iu=@2 } $
ppmout when="output" file="%04d.ppm" file="%04d.ppm",0="0000.ppm" r=0
r0=-1.7 r1=1.7 g=1 g0=-0.7 g1=0.7 b=2 b0=0 b1=1 $
stop when="end" $
end of input file $
Loop of 6 devices created:
   (0)schedule (1)ic (2)diff (3)euler (4)ppmout
   (5)stop $
STOP AT 2000[0]
TIMING INFO:
This run took 2.157187 seconds.
BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 13:27:26 2013
=================================================
569 14:11:15 vnb$
```

Namely, it reports the values of the device parameters as they were read and understood. Check these out in case of any troubles. Say misspelling a parameter name may go unnoticed otherwise.

Running it with the –verbose option

```
576 14:32:40 vnb$ Beatbox_SEQ minimal.bbs -verbose > /dev/null
577 14:32:43 vnb$ cat minimal.log
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-----------------------------------------------------------------------
execution begin at Mon Jun 17 14:32:41 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile verbose logname=minimal.log
def int narg 0$def str 0 minimal$state anisotropy=0! xmax=102 ymax=102
zmax=1! vmax=3 /* Grid size (102 x 102 x 1 x 3) */$
def real begin 0$def real output 0$def real end 0$
k_func when="always"! nowhere=1 row0=0! row1=0! col0=0! col1=0! color=15!
name="schedule" pgm={
begin=eq(t,0);
output=eq(mod(t,100),0);
end=ge(t,2000);
} $
k_func when="begin" nowhere=0! x0=1 x1=100 y0=1 y1=100 z0=0! z1=0! v0=0!
v1=0! row0=0! row1=0! col0=0! col1=0! color=15! name="ic" pgm={
u0=-1.7+3.4*gt(x,50);
u1=-0.7+1.4*gt(y,50);
} $
diff when="always"! nowhere=0! x0=1! x1=100! y0=1! y1=100! z0=0! z1=0!
v0=0 v1=2 row0=0! row1=0! col0=0! col1=0! color=15! name="diff"! hx=0.5
D=1 $
…
```

Specifying a –verbose option on the command line does not change the standard output, but adds more details to the log file.

Running it with the –verbose option

```
576 14:32:40 vnb$ Beatbox_SEQ minimal.bbs -verbose > /dev/null
577 14:32:43 vnb$ cat minimal.log
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-----------------------------------------------------------------
execution begin at Mon Jun 17 14:32:41 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile verbose logname=minimal.log
def int narg 0$def str 0 minimal$state anisotropy=0! xmax=102 ymax=102
zmax=1! vmax=3 /* Grid size (102 x 102 x 1 x 3) */$
def real begin 0$def real output 0$def real end 0$
k_func when="always"! nowhere=1 row0=0! row1=0! col0=0! col1=0! color=15!
name="schedule" pgm={
begin=eq(t,0);
output=eq(mod(t,100),0);
end=ge(t,2000);
} $
k_func when="begin" nowhere=0! x0=1 x1=100 y0=1 y1=100 z0=0! z1=0! v0=0!
v1=0! row0=0! row1=0! col0=0! col1=0! color=15! name="ic" pgm={
u0=-1.7+3.4*gt(x,50);
u1=-0.7+1.4*gt(y,50);
} $
diff when="always"! nowhere=0! x0=1! x1=100! y0=1! y1=100! z0=0! z1=0!
v0=0 v1=2 row0=0! row1=0! col0=0! col1=0! color=15! name="diff"! hx=0.5
D=1 $
…
```

Say, it causes the k-programs in the k_func devices to be spelled out.

Running it with the –verbose option

```
576 14:32:40 vnb$ Beatbox_SEQ minimal.bbs -verbose > /dev/null
577 14:32:43 vnb$ cat minimal.log
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-----------------------------------------------------------------
execution begin at Mon Jun 17 14:32:41 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend nodebug graph noprofile verbose logname=minimal.log
def int narg 0$def str 0 minimal$state anisotropy=0! xmax=102 ymax=102
zmax=1! vmax=3 /* Grid size (102 x 102 x 1 x 3) */$
def real begin 0$def real output 0$def real end 0$
k_func when="always"! nowhere=1 row0=0! row1=0! col0=0! col1=0! color=15!
name="schedule" pgm={
begin=eq(t,0);
output=eq(mod(t,100),0);
end=ge(t,2000);
} $
k_func when="begin" nowhere=0! x0=1 x1=100 y0=1 y1=100 z0=0! z1=0! v0=0!
v1=0! row0=0! row1=0! col0=0! col1=0! color=15! name="ic" pgm={
u0=-1.7+3.4*gt(x,50);
u1=-0.7+1.4*gt(y,50);
} $
diff when="always"! nowhere=0! x0=1! x1=100! y0=1! y1=100! z0=0! z1=0!
v0=0 v1=2 row0=0! row1=0! col0=0! col1=0! color=15! name="diff"! hx=0.5
D=1 $
…
```

It also reports the values of device parameters that were taken as default values, so you can check if this is what *you* wanted. NB: the default values are followed by exclamation marks! – so you can recognize them as such.

Running it with the –debug option

```
583 14:42:36 vnb$ Beatbox_SEQ minimal.bbs -debug stdout > minimal.out
584 14:42:43 vnb$ cat minimal.out
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#---------------------------------------------------------------------
execution begin at Mon Jun 17 14:42:41 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend debug=stdout graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop $
t=0: 0(schedule) 1(ic) 2(diff) 3(euler) 4(ppmout)
t=1: 0(schedule) 2(diff) 3(euler)
…
t=99: 0(schedule) 2(diff) 3(euler)
t=100: 0(schedule) 2(diff) 3(euler) 4(ppmout)
t=101: 0(schedule) 2(diff) 3(euler)
…
t=1999: 0(schedule) 2(diff) 3(euler)
t=2000: 0(schedule) 2(diff) 3(euler) 4(ppmout) 5(stop)STOP AT 2000[0]
STOP AT 2000[0]
```

The –debug option requires also giving the name of the file where to put the debug information. The name stdout means standard output, so debug messages will be intermingled with the normal messages.

Running it with the –debug option

```
583 14:42:36 vnb$ Beatbox_SEQ minimal.bbs -debug stdout > minimal.out
584 14:42:43 vnb$ cat minimal.out
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#--------------------------------------------------------------------
execution begin at Mon Jun 17 14:42:41 2013 $
Input file minimal.bbs without additional arguments $
with options: noappend debug=stdout graph noprofile noverbose logname=minimal.log
state /* Grid size (102 x 102 x 1 x 3) */$
k_func $
k_func $
diff $
euler $
ppmout $
stop $
end of input file $
Loop of 6 devices created: (0)schedule (1)ic (2)diff (3)euler (4)ppmout (5)stop $
t=0: 0(schedule) 1(ic) 2(diff) 3(euler) 4(ppmout)
t=1: 0(schedule) 2(diff) 3(euler)
…
t=99: 0(schedule) 2(diff) 3(euler)
t=100: 0(schedule) 2(diff) 3(euler) 4(ppmout)
t=101: 0(schedule) 2(diff) 3(euler)
…
t=1999: 0(schedule) 2(diff) 3(euler)
t=2000: 0(schedule) 2(diff) 3(euler) 4(ppmout) 5(stop)STOP AT 2000[0]
…
```

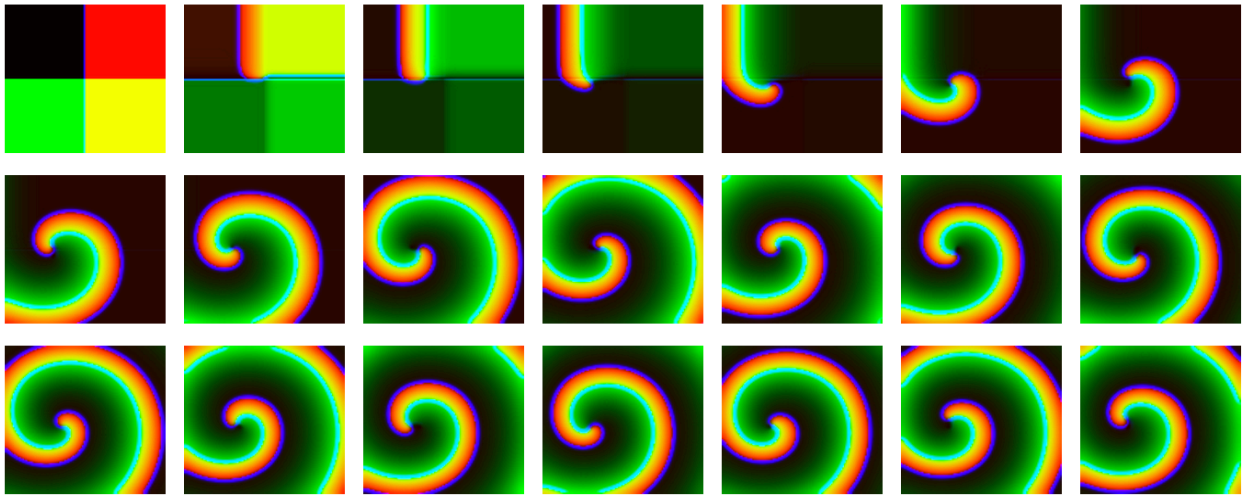
This is the debug information. At every time step t, the numbers of the devices with their names (here is where the names are needed!) are printed in order of execution. If a crash happens, you will know on whose watch it was!

Running it with the –profile option

```
588 15:04:23 vnb$ Beatbox_SEQ minimal.bbs -profile
#! BeatBox 1.3 revision 663M, sequential compile Jun 17 2013 13:23:02
#-------------------------------------------------------------------------
-------
execution begin at Mon Jun 17 15:04:29 2013 $
…
STOP AT 2000[0]

TIMING INFO:
This run took 2.055643 seconds.

Profiling summary:
No  Device name          # calls     %      ms per step      ms per call
  0 schedule                2001   0.4          0.00359          0.00359
  1 ic                         1   0.7          0.00680         13.60400
  2 diff                    2001  25.1          0.25700          0.25687
  3 euler                   2001  72.9          0.74646          0.74608
  4 ppmout                    21   1.0          0.01005          0.95690
  5 stop                       0   0.0          0.00000          0.00000

BeatBox 1.3 finished at t=2000 by device 5 "stop"
Mon Jun 17 15:04:31 2013
=========================================================
589 15:04:32 vnb$
```

This option allows timing of individual device instances, so you could know what took it so long to do the job.

The image output from minimal.bbs



- Files 0000.ppm, 0001.ppm … 0020.ppm are created in the current directory.
- Viewers for ppm files are available on many platforms. Alternatively, one can convert ppm files into a more common image format using netpbm, e.g.

```
>  for (( i=0 ; $i<=20 ; i=$i+1 )); do n=`printf %04d $i`; pnmtopng $n.ppm > $n.png; done
```

END