Andre Bartetzki
e-mail: abart@berlin.snafu.de
abart@gigant.kgw.tu-berlin.de
www: http.//www.kgw.tu-berlin.de/~abart

STEAM
Studio für elektroakustische Musik
der Hochschule für Musik Berlin, Germany
phon: (+49) 30 20309 2488

CMask

a stochastic event generator for Csound

CMask is an application that produces *score files* for Csound, i.e. lists of notes or rather events. Its main application is the generation of events to create a texture or granular sounds.
The program takes a *parameter file* as input and makes a *score file* that can be used immediately with Csound.
The basic concept in CMask is the tendency mask. This is an area that is limited by 2 time variant boundaries. These area describes a space of possible values for a score parameter, for example amplitude, pitch, pan, duration etc. For every parameter of an event (a *note statement pfield* in Csound) a random value will be selected from the range that is valid at this time (see [a], [c], [d], [4], [5], [7]). There are also other means in CMask for the parameter generation, for example cyclic lists, oscillators, polygons and random walks. Each parameter of an event can be generated by a different method.
A set of notes / events generated by a set of methods lasting for a certain time span is called a field.

CMask was written in the Metrowerks CodeWarrier environment in C++.
The program is available for Macs (68k and PPC), for SGI IRIX5.3 and for Windows95:

```
http://www.kgw.tu-berlin.de/~abart/CMaskMan/CMask_Download.html
ftp://ftp.kgw.tu-berlin.de/pub/cmask/
```

CMask is freeware.

The manual consists of a basics section, a reference and some examples.

July 1997, Andre Bartetzki

BASICS

Parameter and Processing

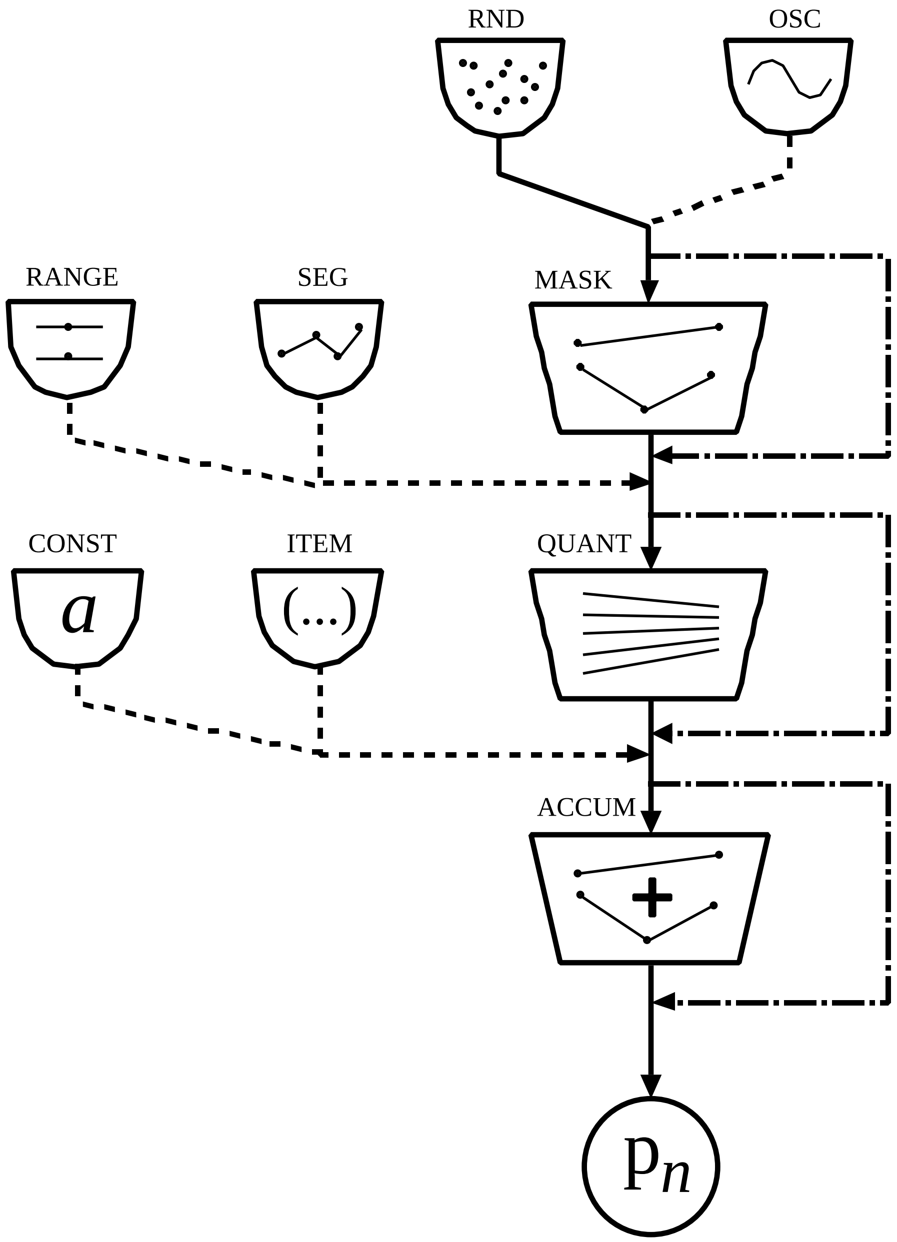
Every event or note in a Csound score is described by an *i statement* and controlled by a number of parameters,
called *pfields* `p1, p2 p3` ... These *pfields* contain instrument number, onset time (or beat) and duration of each event.
Moreover it is possible to define other parameters for the instruments numbered as `p4, p5` etc. These are usually frequencies or pitches, transpositions, filter values, amplitudes, attack times, panorama positions, table numbers and so on.

One parameter in CMask corresponds to one *pfield* of an *i statement* in Csound.
The first parameter is the instrument number `p1`.
The second parameter, in Csound actually the onset *time* (or the beat,) will be regarded as the onset *difference* (or the rhythm) between successive events. The sum of all preceeding onset differences up to a certain event gives us the absolute onset time `p2` of this event. The `p2` parameter in CMask has another special quality: it determines the event density, that is: how many notes will be generated in a certain time span.
The third is the duration `p3`. Every subsequent parameter gets its meaning from the *orchestra file*.

The instructions for the parameter generation take place in a special text file called the *parameter file* .
Normally all the events or grains have the same or similar properties so they can be grouped together from a statistical point of view. These groups or rather the parameters of their events called *fields* in CMask. The notion of fields is comparable to *clouds* or *streams* in other concepts of granular synthesis.

A field has a duration that is determined by start and end time. The values of each paramter `p1,p2,p3 ... p`*n* within this time span can be generated and processed by different means. At first, one of the generators (see below) produces a value. This might be a random or a periodical value or one from a list of fixed numbers. The value can be now processed or modified with up to 3 modules (tendency mask, quantizer, accumulator) in dependence on the generator type. (For example: a list can't be followed by a mask but an accumulator.)
With the mask module the generated values will be mapped into a time variant domain - the tendency mask.
The quantizer adjusts its input values (the values from a mask or a generator) to a time variant grid.
The last module, the accumulator, sums all values together. These 3 modifiers are optional.

Generators	Modifiers		
constant -->			[accumulator] -->
list -->			[accumulator] -->
segment function -->		[quantizer] -->	[accumulator] -->
random generator-->		[quantizer] -->	[accumulator] -->
probability generator -->	[mask] -->	[quantizer] -->	[accumulator] -->
oscillator -->	[mask] -->	[quantizer] -->	[accumulator] -->



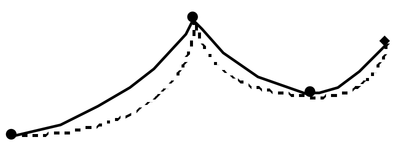
Most of the modules can be controlled by special parameters. These control parameters can be constant or time dependent. The variable parameters will be described by segment functions:

Segment Functions

Segment functions (or break point functions) serves in CMask both as generators of event parameters and as control functions for special generator and modifier parameters. Such special parameters are, for example, the mean value of a gaussian probability distribution, the quantization strength or a boundary of a tendency mask.
A segment functions consists, like the `linseg` and `expseg` unit generators and `GEN5` / `GEN7` in Csound, of a connected sequence of segments. The segments are determined by pairs of time and related function values.
Values between these points will be calculated by an interpolation. There is a linear or a power function interpolation. The interpolation type is set by an exponent (see below and in the reference section). It can also be set off - the result is a step function similar to Csounds `GEN17.`

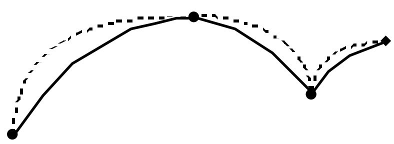
interpolation with a positive exponent (= convex)

```
(0 0 4 10 7 5 9 8 ipl 0.5)
(0 0 4 10 7 5 9 8 ipl 2)
```



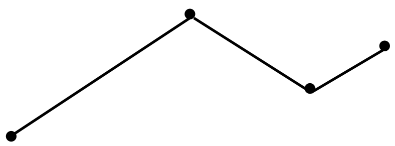
interpol. with a negative exp.(= concave)

```
(0 0 4 10 7 5 9 8 ipl -0.5)
(0 0 4 10 7 5 9 8 ipl -2)
```



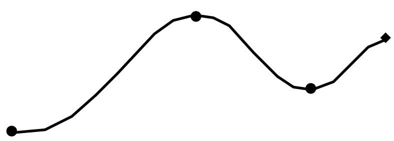
linear interpolation

```
(0 0 4 10 7 5 9 8 ipl 0)
```



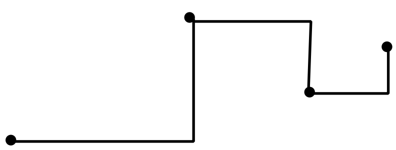
half-cosine interpolation

```
(0 0 4 10 7 5 9 8 ipl cos)
```

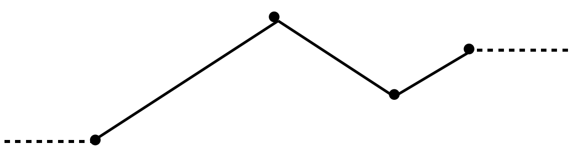


no interpolation

```
(0 0 4 10 7 5 9 8 ipl off)
```



The value of the first defined time-value pair of a segment function is also valid before this time (without interpolation). The same applies to the last defined value:



Random Generators and Probability Distributions

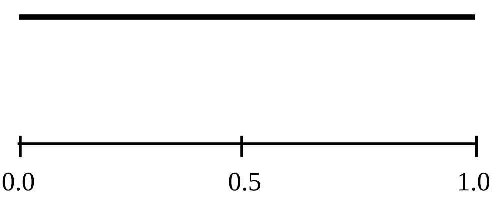
CMask generates random numbers either by a simple random generator (`range`) or by means of dynamic controllable probability functions (`rnd`).
The implemented distributions are: uniform, linear, triangle, exponential, bilateral and reverse exponential, gaussian, cauchy, beta and weibull. For a detailed description of these probabilities see in [a], [1], [2], [3].

Many of these distributions are controllable by one or two parameters. They can be set as constant or as variable by a segment function. A gaussian distribution, for example, is determined by a mean value and a standard deviation. If we vary the standard deviation the shape of the distribution (the histogramm) will be changed.
If we have the segment function pairs 0 1 and 10 3 as control values for an exponential distribution, the result is a rising exponent (lambda) from 1.0 at 0 seconds to 3.0 at 10 seconds, that is, the generator prefers more lower values at time 10.0.
If we have a cauchy distribution and control its mean value by 0 0.5 and 10 0.8, we get a mean value that changes from 50% to 80% of the range {0...1}.

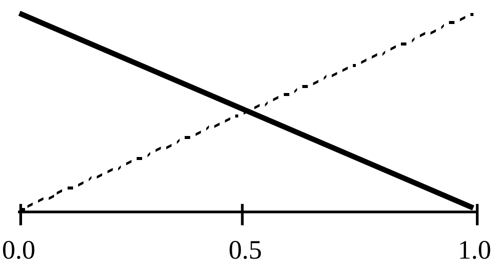
The parameters for every probability function and their ranges are described in the reference section of this manual.

The generated random numbers are limited to the range {0...1} (even if they might be lower or higher in theory, like gaussian, cauchy, exponential and weibull). This range can be stretched or compressed by a subsequent tendency mask.

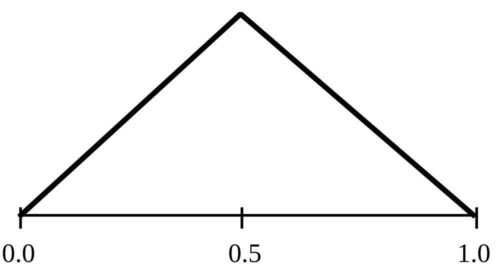
uniform distribution
`rnd uni`



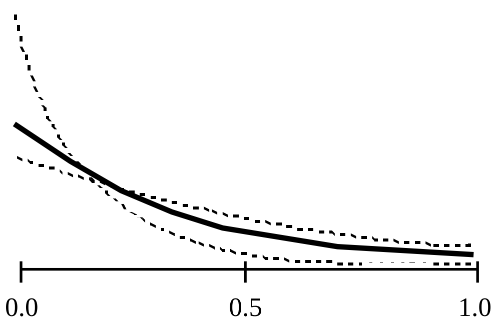
linear distribution
`rnd lin 1 (lin -1)`



triangle distribution
`rnd tri`

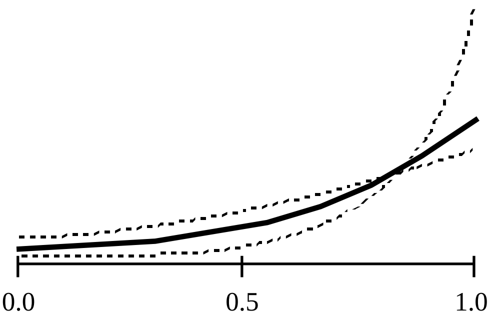


exponential distribution
`rnd exp 1 (exp 0.7, exp 2)`



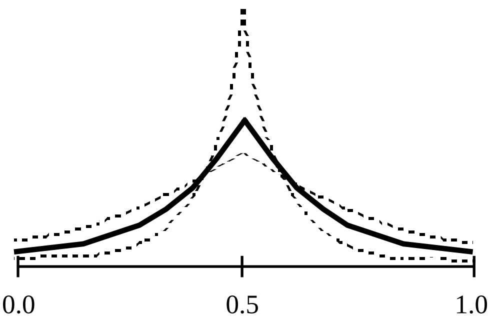
reverse exponential distribution

`rnd rexp 1 (rexp 0.7, rexp 2)`



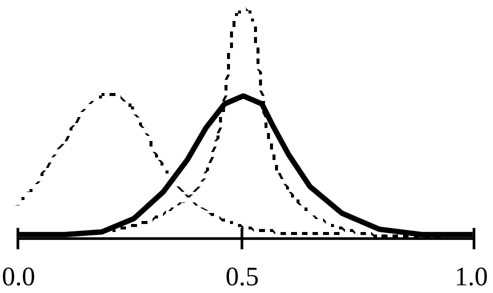
bilateral exponential distribution

`rnd bexp 1 (bexp 0.7, bexp 2)`



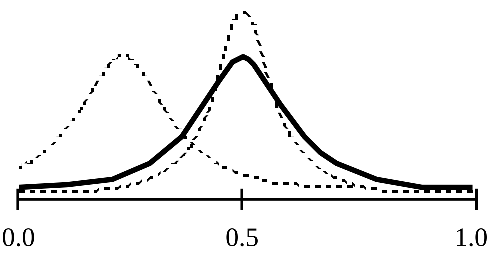
gaussian distribution

```
rnd gauss .5 .2
(gauss .5 .1, gauss .2 .2)
```



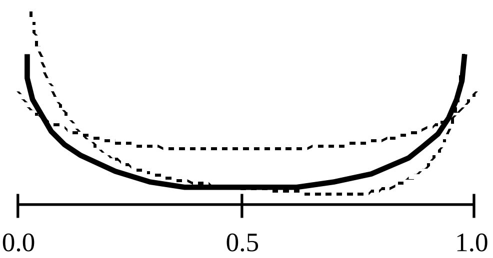
cauchy distribution

```
rnd cauchy .5 .2
 (cauchy .5 .15, cauchy .2 .2)
```



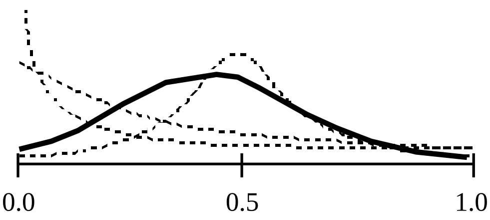
beta distribution

```
rnd beta .2 .2
(beta .6 .6, beta .1 .3)
```



weibull distribution

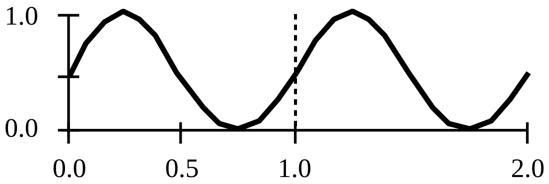
```
rnd wei .5 3
(wei .5 1, wei .5 .3)
```



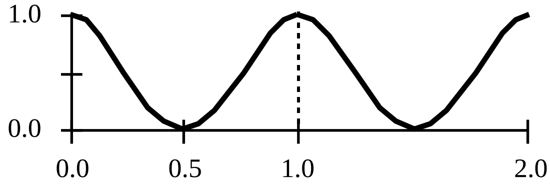
Oscillators

There are not only random generators in CMask. It is also possible to use the following periodic functions as parameter generators: sine, cosine, saw tooth up and down, square, triangle and periodic power functions.
Every function has a frequency value measured in cps, constant or time variant, and a constant phase (normalized, between 0 and 1).
These functions produce values in the range {0...1}, like the random generators, i.e. their amplitudes keep constant.
A subsequent mask can serve as an amplifier to achieve other ranges.

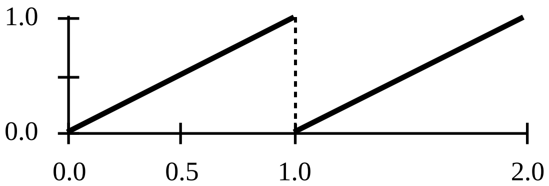
sine function
`osc sin 1`



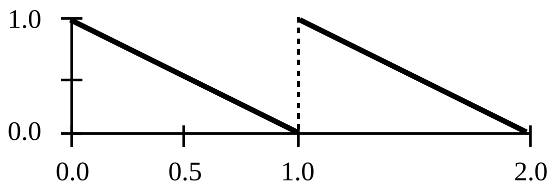
cosine function
`osc cos 1`



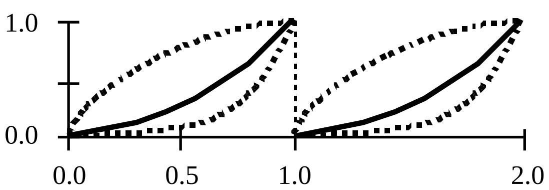
saw up function
`osc sawup 1`



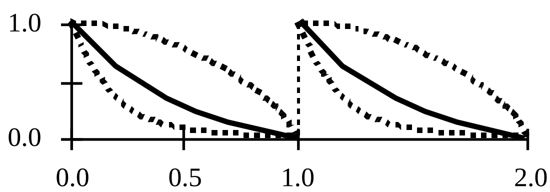
saw down function
`osc sawdown 1`



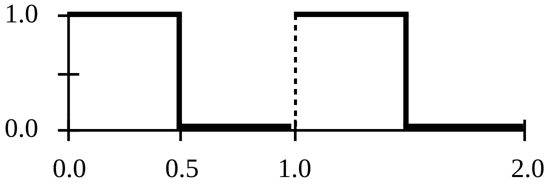
power up function
`osc powup 1 0 1`
`(powup 1 0 2, powup 1 0 -1)`



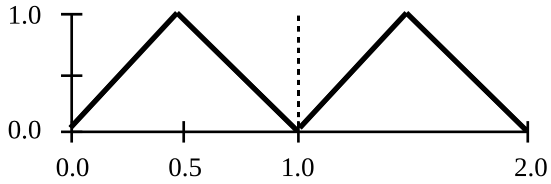
power down function
`osc powdown 1`
`(powdown 1 0 2,powdown 1 0 -1)`



square function
`osc square 1`



triangle function
`osc triangle 1`



Lists

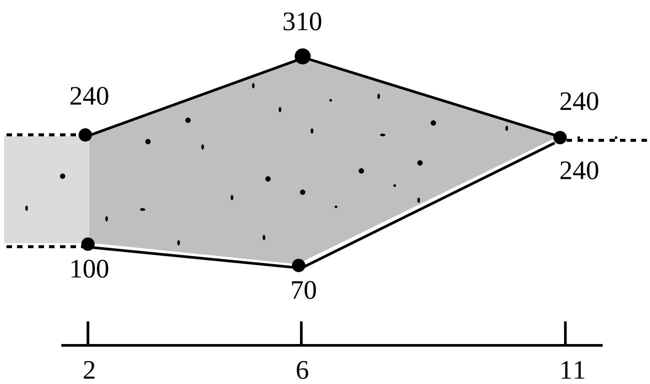
Instead of random or oscillating generators one can use simple lists.
A list contains a set of numbers, for example (5 1 8 10.2) or (1.5 0.01 0.01 1234) etc.
The elements of the list can be selected by different methods: in a forward loop (`cycle`), in a forward and backward loop, like a pendulum or a palindrome (`swing`), as random permutations, like a stack of cards (`heap`) and by chance (`random`). There is in contrast to segment functions no time dependence at all - the elements will be read one by one corresponding to the mode.
The example shows the four modes with the same list:

```
item cycle (1 2 3 4)  ->                    1 2 3 4 1 2 3 4 1 2 3 4
item swing (1 2 3 4)  ->                    1 2 3 4 3 2 1 2 3 4 3 2
item heap  (1 2 3 4)  ->                    3 2 4 1 2 3 1 4 4 3 1 2
item random (1 2 3 4) ->                    2 4 2 1 1 3 2 3 4 1 2 2
```

Tendency Masks

A mask is an area that is limited by two (time variant) boundaries. The numbers produced by a preceding random generator (`rnd`) or oscillator (`osc`) will be mapped into this range.
Upper and lower mask limit may be constant or a segment function.

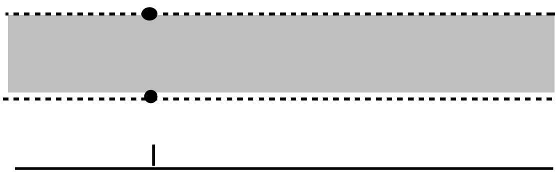
```
p4 rnd uni
mask (2 100 6 70 11 240) (2 240 6 310 11 240)
```



This example describes a uniform random generator for `p4` (a frequency ?) and a tendency mask. The masks range at 2 seconds is {100...240}, therefore the random numbers are distributed between 100 and 240. The range expands to {70...310} at 6 seconds and shrinks to a single value of 240 at 11 seconds.

The following forms are special border cases of the mask principle:

1. A mask with a *constant* lower and a *constant* upper boundary ist actually a normal (time invariant) range:



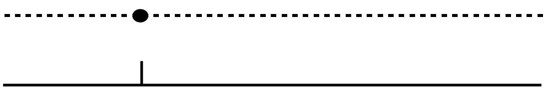
Therefore, these two parameter descriptions are equivalent:

```
p4 rnd uni
mask 10 20
```

or

```
p4 range 10 20
```

2. A mask with *constant* and *identical* boundaries describes a constant value:



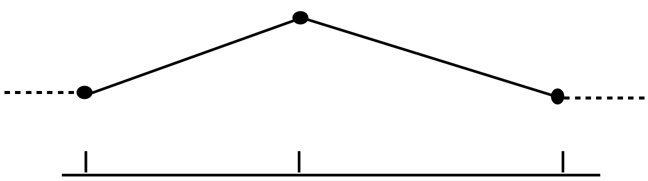
Therefore, these two parameter descriptions are equivalent:

```
p4 rnd uni
mask 10 10
```

or

```
p4 const 10
```

3. A mask with time variant but *identical* boundaries describes a normal segment function - there is no range for random or oscillating numbers:



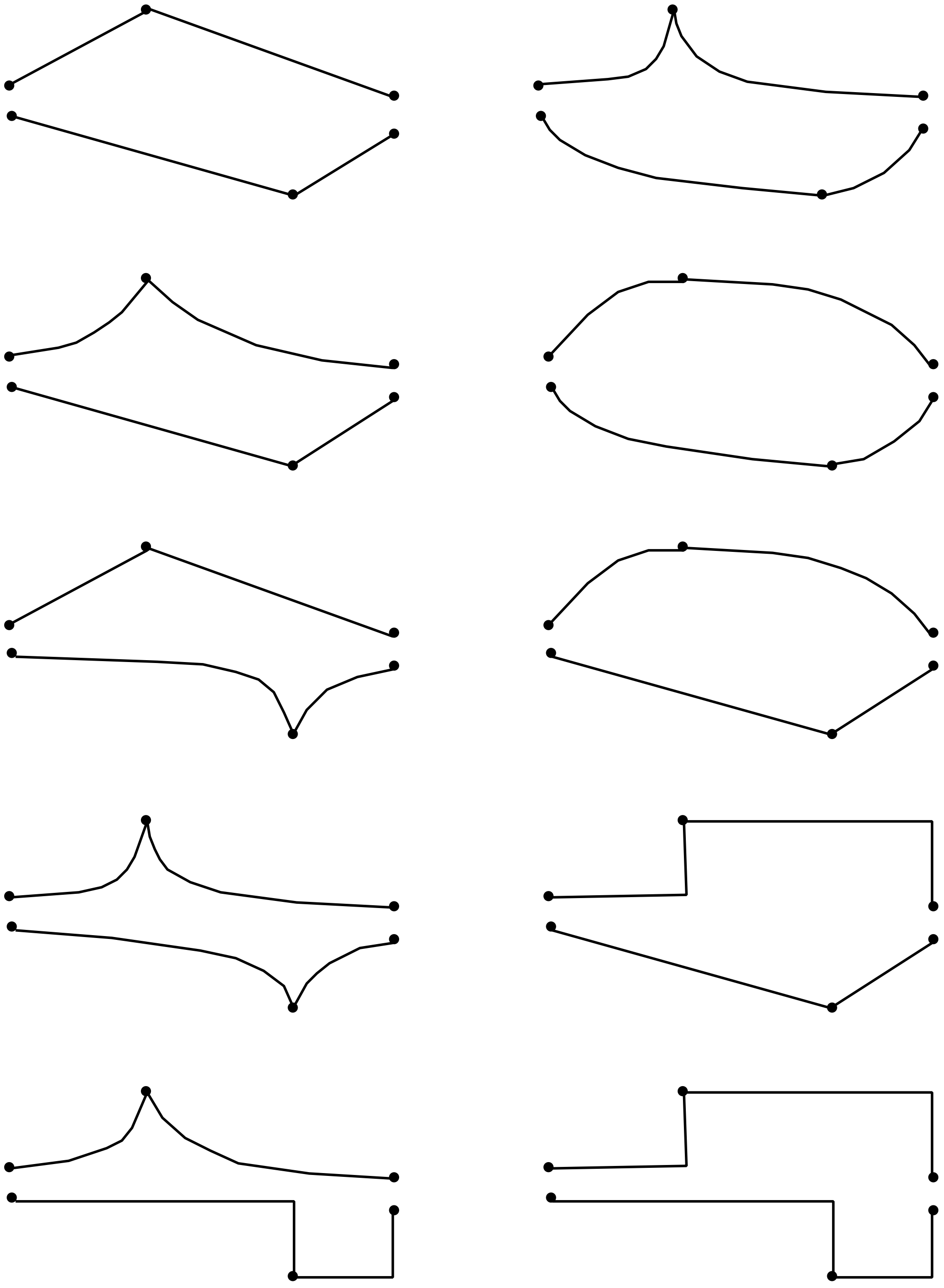
Therefore, these two parameter descriptions are equivalent:

```
p4 rnd uni
mask (3 10 5 20 7 10) (3 10 5 20 7 10)
```

or

```
p4 seg (3 10 5 20 7 10)
```

Because of the free choice of an interpolation exponent for segment functions there is a large variety of possible mask shapes constructed by the same set of break points:



Mapping

The mathematical transformation (mapping) of the random or periodic values into the mask follows a linear, a power oder a root function.
A linear mapping, probably normal for many CMask applications, expands oder compresses the values to the current mask range in a proportional (linear) way. For example, if the random values 0.0, 0.1, 0.5 and 0.8 will be linear transformed in a range between 2 and 6, the result will be 2.0, 2.4, 4.0 and 5.2. With a range of -10 ... +10 we will have -10.0, -8.0, 0.0, 6.0.
A power function changes the original proportion. A square function transforms the valuea 0.0, 0.1, 0.5 and 0.8 to 0.0, 0.01, 0.25 and 0.64. A mapping into the range [2,6] results in 2.0, 2.04, 3.0 and 4.56.
A square root function, for example, transforms the values to 0.0, 0.316..., 0.707... and 0.894.... In the range [2,6] this is 2.0, 3.264..., 4.828... und 5.577....
The mapping function in CMask is

$$f(x) = x^{2^n}$$

x is the input value and *n* the mapping exponent

n=0 gives a linear function.
n>0 gives a power function, *n=1* gives the square function.
n<0 gives a root function, *n=-1* gives the square root function.

For many synthesis parameters the linear transformation would be the right choice.
But if we want to generate uniform distributed frequencies for use in audio oscillators between 100 and 800 Hz for example, they will distributed equally in that range, but we will hear more higher pitches. In this example the half of all frequencies fall in the range 450...800 Hz, this is about 1 octave. The other half (100...450) comprises more than 2 octaves. Therefore pitches from the lowest octave will be rare in relation to the pitches from the highest octave. To solve the problem of the logarithmic frequency-pitch-relation one can either use `cent` or `oct` values instead of frequencies or use as approximation an exponential distribution instead of uniform. The third way is a square function as mapping transformation.
We have a similar case with onset differences and durations. Uniform distributed time intervals or durations would be perceived as a preference of larger intervals and durations.

Quantization

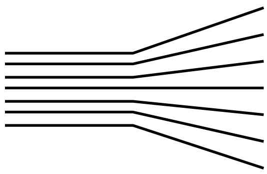
After generation and mask mapping it is possible to quantize the numbers. There are 3 parameters for quantization:
interval, *strength* and *offset* .

The quantization interval is the distance between two neighbouring points in the value range. All these points form an equidistant grid. These points attract surrounding random values like magnets. (Compare with metrical quantize in a MIDI sequencer.)

An interval of 30, for example, builds a grid:
.... -120 -90 -60 -30 0 30 60 90 120 150 180 ....

The interval can also be time variant. The segment function `(0 30 5 30 10 45)`, for example, describes a grid that is constant before 30 seconds and increasing after this time:

```
quant (0 30 5 30 10 45) 1 0
```

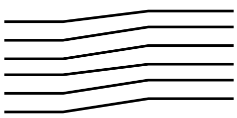


The quantization offset**,** also controllable by a segment function, is a shift of the grid.

An offset value of 10 and an interval of 30 results in the grid:
.... -110 -80 -50 -20 10 40 70 100 130 160 190....

This is an example for a dynamic offset:

```
quant 40 1 (3 0 6 20)
```

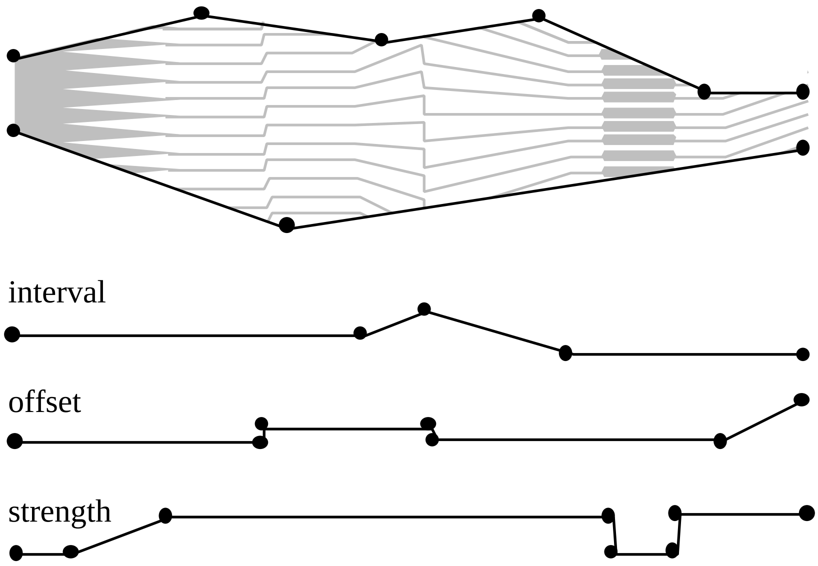


The quantization strength, a value or a segment function in the range {0...1}, i.e. 0 ... 100%, specify how the grid points attract values between them.
A strength value of 1 gives a total quantization - every random value are now precise on the grid. A strength of 0.5 (50%) means, that all random values are shifted to the half distance to the next grid point. (Compare with "Iterative Quantize" in a sequencer.)

An example for an increasing strength:

```
quant 40 (0 0 10 1) 0
```

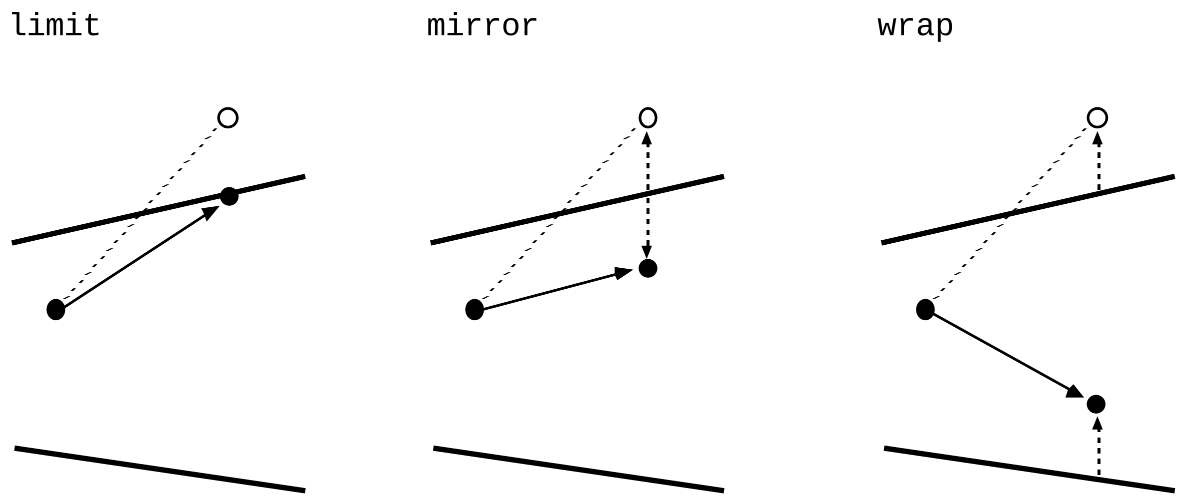
The example below shows a tendency mask combined with a quantization with dynamic interval, offset and strength (grey areas in the mask are possible areas for random or periodic values):



Quantization is useful for the construction of regular meters in rhythm oder harmonic frequency grids.

Accumulation

An accumulator adds all its input values and an initial value together.
For example, the numbers 10 2 34 5 and an inital value of 0 results in 10 12 46 51. That is, one can look upon the input values as relative values or intervals. The numbers after accumulation are absolute values.
If we use an accumulator in connection with a random generator, we can call the result a random walk. Depending on the input values, the sum may rise to a very large number. In order to limit to sum to a certain range it is possible to define lower and upper boundaries similar to a mask. The specification of the accumulation mode prescribes the behavior near to a boundary. The `limit` - mode cuts values higher than the upper limit and vice versa. The `mirror` - mode reflects the amount that crosses the limit back into the allowed range. The `wrap` - mode considers both limits the same, so that a value that is too high comes out above the lower boundary and vice versa.



An example

The lines below describe a field that contains 6 parameter in the valid CMask syntax:

```
{
f1 0 8193 10 1                   ;text in braces take place
                                 ;in the score file unchanged
i3 0 20
}

f 0 10                           ;a field from 0 to 10 seconds

p1 const 1                       ;instrument 1

p2 range .01 .2 prec 3           ;onsets between 10 and 200 ms
                                 ;precision: 3 digits after
                                 ;the decimal point

p3 rnd exp 1                     ;durations between 0.5
                                 ;and 1 seconds
mask .5 1                        ;and exponential distribution
prec 2

p4 item heap (120 125 400 355)   ;permutations of 4 values

p5 rnd uni                       ;uniform distributed numbers
mask (3 100 8 50) 200 map 1      ;between 100 and 200 at first and
                                 ;between 50 and 200 in the end

p6 osc cos (0 .5 10 5)           ;faster and faster oscillating
                                 ;values in the range {0...1}
```

Working with CMask

As usual one should design the instruments and the *orchestra file* at first. If the meaning of the pfields `p4-p`*n* is fixed one can write generators, masks etc. for all the *pfields* in a *parameter file*. For a better readability it is advisable to insert comments (after a semicolon).
The file format of the *parameter file* have to be plain ASCII text, similar to the `.orc` and `.sco` files.
Suitable text editors for Csound and CMask on Macs are BBedit or Alpha.

If the *score file* should also contain other note lists or function table statements one can write these in braces {...} as the first lines of the *parameter file*. This part of the *parameter file* will be taken unchanged into the *score file*. In this way one gets a complete score file for Csound that don't have to be further edited.

Each parameter can have its own precision. The default value for the decimal places after the point is 5. In order to get integer numbers, for example, the precision have to be 0.

CMask on the Macintosh:

After launching CMask a file dialog box appears to select the *parameter file*. Then one have to set the path and type the name for the new *score file*. The default for the file name is the name of the *parameter file* with appended '.sco'.
If an errors occurs, CMask stops and prints a message. One have to quit (command-Q) the program now. After debugging one can restart CMask. If the text was OK, CMask writes the *score file* to the disk.

CMask on SGI:

CMask expects at least on argument: the name of the *parameter file*. The optional second argument may contain the name for the new *score file*. If no *score file* name is given, CMask appends '.sco' to the *paramater file* name as the new *score file* name. If an errors occurs, CMask prints a message and stops. After debugging one can restart CMask. If the text was OK, CMask writes the *score file* to the disk.

CMask on the PC:

There is no GUI for CMask, so you have to run it in a MS-DOS-Window.
CMask expects at least on argument: the name of the *parameter file*. A special file type extension is not required. The optional second argument may contain the name for the new *score file*. If no *score file* name is given, CMask appends '.sco' to the *paramater file* name as the new *score file* name. If an errors occurs, CMask prints a message and stops. After debugging one can restart CMask. If the text was OK, CMask writes the *score file* to the disk.

REFERENCE

Symbols and numbers in a parameter file have to be limited with space, tab or return.
The values of a modul (generator, mask etc.) can be distributed over several lines.
Several moduls can also be written together in one single line.

Symbols (**f, rnd, mask, accum ...**) can be written in upper case or lower case letters.

Prescribed text

```
{ <text> }
```

The text in braces may contain function table statements, note lists and other score statements inclusive comments, which should be copied unchanged into the *score file* . This prescribed text must be written before any field desriptions.

Example:

```
{
f1 0 8193 10 1
f2 0 1024  8 0 512 1 512 0
; tables with sine wave and envelope
}
```

Comment

```
; <text>
```

Comments can be written elsewhere into the *parameter file* .
They begin with a semicolon **;** and extend to the end of the line.

Segment function (`bpf`)

(`<t1 v1> <t2 v2> <t3 v3> <..> [`**ipl** `<<val> |` **cos** `|` **off**`>]` **)**

[`<vstart> <vend> [`**ipl** `<<val> |` **cos** `|` **off**`>]` **]**

Almost every module in CMask can controlled by a constant or by time dependend values. The description of variable values is done by segment functions (also known as break-point functions or polygonals). Such a function have to be written within parenthesis as a sequence of time-value pairs or points (`<tn vn>`). The function value of the first point is also valid before its time. The same principle applies to the last point.
Values between the given points will be calculated by linear or power function interpolation. The optional interpolation exponent follows the keyword **`ipl`**. The default for interpolation is 0, that is linear. If **`ipl`** is followed by **`off`**, the result a step function instead of interpolation.

interpolation exponent	result
0	linear (similar to `linseg` in Csound)
>0	slowly rising, fast decaying (convex, similar to `expseg` in Csound)
<0	fast rising. slowly decaying (concave)
`cos`	half-cosine interpolation (smooth curve)
`off`	no interpolation (steps, similar to `GEN17` in Csound)

There is a simplified way to write functions that consists of only one segment lasting from start to end of the field. A simple segment function have brackets and only two numbers - the start and the end value.

Examples:

```
(0 3 5 0 8 -1)    ;3 points {0,3},{5,0} and {8,-1} with linear interp.
(0 3 5 0 ipl -1)  ;2 points {0,3},{5,0} and {8,-1} with concave interp.
[10 50 ipl 2.3]   ;2 points {start,10},{end,50} with strong convex int.
[10 50 ipl cos]   ;2 points {start,10},{end,50} with smooth interpolation
```

Field

```
F        <start time> <end time>
         <parameter1>
         <parameter2>
         ...
```

A *parameter file* consists of one or any number of *fields*. Fields are comparable to clouds or streams in granular synthesis systems.
A field describes at least the first 3 parameter of any Csound instrument: the instrument number, the onset time (or beat) and the duration. Depending on the structure of the instrument there can be other parameters.
Every field must have a header that is marked by the letter F. Then follow the start and the end time of the field in seconds. After the header begins the block with parameter descriptions for `p1, p2, p3` etc. The field description ends after the last given parameter p*x* or with a declaration of a next field.
Fields don't have to be declared in chronological order.

Parameter

```
pn       <generator>
         [<mask>]
         [<quantizer>]
         [<accumulator>]
         [prec <val>]
```

Each CMask parameter **p***n* will be described with one or more moduls from the processing scheme (see above).
n corresponds to the numbering in Csound *score files* `p1, p2, p3` etc.
A CMask parameter have to contain at least one of the generators (`const, item, seg, range, rnd, osc`).
Any subsequent modifying moduls (`mask, quant, accum`) are optional. In dependence on the choosen generator there only certain modifying moduls allowed.

Precision

```
prec     <val>
```

The numbers computed by the last modul of every parameter will be rounded normally to a precision of 5, that is 5 decimal places after the point. This precision in changeable with the optional **prec** - value (0...5).
`<val>` defines the desired number of decimal places for the output, 0 results in integers, negative values are not allowed.

Generators

A generator produces numbers, which may be further processed by a mask, a quantizer or an accumulator.
Currently there are the follwing generators implemented: `const, seg, range, rnd, osc` and `item.`

Constant

```
const    <val>
```

`const` generates a constant value `val`.
Possible processing with **`accum.`**

Example:
```
p1 const 2 ;constant instrument number
```

List

```
item     <mode> <list>
```

`item` reads one number at a time from `list` dependend on the `mode`.

access	mode	example
cyclic	`cycle`	`(1 2 3 4) -> 1 2 3 4 1 2 3 4 1 2 3 4`
swinging	`swing`	`(1 2 3 4) -> 1 2 3 4 3 2 1 2 3 4 3 2`
permutations	`heap`	`(1 2 3 4) -> 3 2 1 4 3 1 4 2 2 3 4 1`
random	`random`	`(1 2 3 4) -> 4 2 3 3 2 1 3 4 1 2 1 4`

Possible processing with **`accum.`**

Example:
```
p4 item heap (400 410 30.3 5000 1222) ;5 different values
```

Segment function

`seg` `<bpf>`

`seg` reads function values from the segment function given in `bpf`.
Possible processing with **`quant`** and **`accum`**.

Example:

```
p3 seg (0 1 4 .3 6 1)
```

Random number generator

`range` `<val1> <val2>`

`range` generates uniform distributed random numbers in the range {`val1,val2`}.
Possible processing with **`quant`** and **`accum`**.

Example:

```
p2 range .1 .5 ;rhythms between 0.1 and 0.5 seconds
```

Probability distribution generator

rnd `<func> [<val1 | bpf1> [<val2 | bpf2>]]`

rnd produces random numbers according to a probability distribution `func`. Some of the distributions have optional parameters to describe their shape precisely. The values of these optional parameters may be constant (`val`) or time variant (`bpf`).

Probability distributions:

distribution	name	optional parameters	default	remarks
uniform	uni	-		
linear	lin	<val>	1.0	;>0 : decrease <0 : increase
linear, increasing	rlin	-		
triangle	tri	-		
exponential, decreas.	exp	<val | bpf>	1.0	; only values >0
exponential, increas.	rexp	<val | bpf>	1.0	; only values >0
exponent., bilateral	bexp	<val | bpf>	1.0	; only values >0
gaussian	gauss	<val | bpf> <val | bpf>	0.1 0.5	; standard deviation ; and mean {0...1}
cauchy	cauchy	<val | bpf> <val | bpf>	0.1 0.5	; spread and mean {0...1}
beta	beta	<val | bpf> <val | bpf>	0.1 0.1	; a and b {0...1}
weibull	wei	<val | bpf> <val | bpf>	0.5 2.0	; s {0...1} and t {>0}

Possible processing with **mask**, **quant** and **accum**.

Example:
```
p4 rnd gauss (0 .1 10 .6) .5
```

Oscillator

osc `<func> [<freq | bpf> [<phs> [<exp>]]]`

osc generates numbers according to a periodic function `func` in dependence on a constant or variable frequency and an optional phase.

Periodic functions:

function	name	parameter	remark
sine	`sin`	`<val | bpf> [<val>]`	;frequenz in cps, phase {0...1}
cosine	`cos`	`<val | bpf> [<val>]`	
saw, increasing	`sawup`	`<val | bpf> [<val>]`	
saw, decreasing	`sawdown`	`<val | bpf> [<val>]`	
square	`square`	`<val | bpf> [<val>]`	
triangle	`triangle`	`<val | bpf> [<val>]`	
power function, increasing	`powup`	`<val | bpf> [<val>] [<val>]`	;frequenz, phase, exponent
power function, decreasing	`powdown`	`<val | bpf> [<val>] [<val>]`	;frequenz, phase, exponent

Possible processing with **mask**, **quant** and **accum.**

Example:

```
p4 osc sin (0 2 3 .5 ipl 1) ;from 2 to 0.5 Hz falling frequency
```

Modifier

A modifier processes the numbers which was generated by a preceeding generator or another modifier.
All modifying moduls are optional, but there is only one possible order (see the processing scheme above).
The implemented modifiers are: `mask, quant` and `accum.`

Tendency mask

mask <low | bpf> <high | bpf> [**map** <exp>]

The values generated by **rnd** or **osc** `can` can be mapped onto a time dependend range - the tendendency mask.
Both `low` and `high` boundaries may be constant or time variant (`bpf`).
The mapping process itself depends on the optional **map**-exponent: Values which was produced by **rnd** or **osc** are always in the range {0...1}. Within a mask they will now transformed by a (nonlinear) function:
y = x ^(2^exponent).
The default exponent is 0, this results in a linear transformation.
Positive exponents result in a power function. For example: an exponent 1 makes the squares of the input numbers, 0.5 will be 0.25, 0.3 will be 0.09, that is, all values gets smaller (except 0 and 1).
Negative exponents result in a root function. For example: an exponent -1 makes the square root of the input numbers, 0.5 will be 0.79 0.3 will be 0.669, that is, all values gets larger (except 0 and 1).

Example:
```
p4 rnd uni
mask 50 (2 100 10 200) ;lower limit 50, upper limit from 100 to 200
```

Quantizer

quant <q | bpf> [<s | bpf> [<o | bpf>]]

A quantizer attracts the results of the preceeding modul onto a quantization grid `q`. The optional strength `s` {0...1} (default 1) specifies to which extent the grid takes effect (1 is total quantization, 0 is no quantization at all). The optional offset `o` (default 0) shifts the whole grid by the given amount. Each of the 3 parameters may be constant or time variant (`bpf`).

Example:
```
p4 range 100 200
quant 20 [0 1]   ; dynam. quantization to a grid ->
100,120,140,160,180,200
```

Accumulator

accum <mode> [<low | bpf> <high | bpf>] [**init** <exp>]

The accumulator continuously sums all its input values to an inital value **init**. The inital value is optional and has a default value of 0. In order to restrict the sum to a certain range it is possible to define lower and upper limits similar to tendency masks.
The accumulators behavior near by the limits can be determined by one of three limiting modes.

method	mode	parameter
without any limiting	on	
with limiting	limit	<low | bpf> <high | bpf>
with limiting and reflection	mirror	<low | bpf> <high | bpf>
with cyclic closed limits	wrap	<low | bpf> <high | bpf>

Example:
```
p4 range -10 10
akkum mirror 100 300 init 200
```

REFERENCES

[1] Ames, Ch. 1991. "A Catalog of Statistical Distributions..." *Leonardo Music Journal* 1(1)
[2] Dodge, Ch. / Jerse, Th. A. 1985. *Computer Music* Collier Macmillan, London
[3] Lorrain, D. 1980. "A Panoply of Stochastic 'Cannons'." *Computer Music Journal 4*(1)
[4] Roads, C. 1991. "Asynchronous Granular Synthesis." in *Representations of Musical Signals*, MIT Press
[5] Roads, C. 1985. "Granular Synthesis of Sounds." in *Foundations of Computer Music*, MIT Press
[6] Truax, B. 1988. "Real-time Granular Synthesis with a Digital Sound Processor." *Computer Music Journal* 12(2)
[7] Wishart, T. 1994. *Audible Design* Orpheus the Pantomime
[8] Xenakis, I. 1992. *Formalized Music* Pendragon Press

SOFTWARE

[a] Castine, P. Litter Package (Externals for MAX)
[b] Koenig, G.M. Project I, Project II
[c] Truax, B. POD programs
[d] Xenakis, I. Stochastic Music Program

EXAMPLES

A simple FM-instrument:

```
;; bells.orc

sr      = 44100
kr      = 4410
nchnls  = 2

instr 1
        ;p2 onset
        ;p3 duration
        ;p4 base frequency
        ;p5 fm index
        ;p6 pan (L=0, R=1)

kenv    expon     1,p3,0.01
kindx   expon     p5,p3,.4
a1      foscil    kenv*10000,p4,1,1.143,kindx,1
        outs      a1*(1-p6),a1*p6

endin
;; --------------------
```

A dynamic texture made of FM-bells:

```
;; bells parameter file
{
f1 0 8193 10 1                        ;sine wave for foscil
}

f 0 20                                ;field duration: 20 secs

p1 const 1

p2                                    ;decreasing density
rnd uni                               ;from .03 - .08 sec
                                      ;to .5 - 1 sec
mask [.03 .5 ipl 3] [.08 1 ipl 3] map 1
prec 2

p3                                    ;increasing duration
rnd uni
mask [.2 3 ipl 1] [.4 5 ipl 1]
prec 2

p4                                    ;narrowing frequency grid
rnd uni
mask [3000 90 ipl 1] [5000 150 ipl 1] map 1
quant [400 50] .95
prec 2

p5                                    ;FM index gets higher from
2-4 to 4-7
rnd uni
mask [2 4] [4 7]
prec 2

p6 range 0 1                          ;panorama position
                                      ;uniform distributed
prec 2                                ;between left and right

;; --------------------
```

The same .orc file but lists and several fields:

```
;; bells parameter file
{
f1 0 8193 10 1                         ;sine wave for foscil
}

f 0 10                                 ;field 1

p1 const 1

p2 range .1 .3 prec 2                  ;density between 100
                                       ;and 300 ms

p3 range .7 1.2 prec 2

p4 item heap (300 320 450 430 190)     ;5 frequencies in
                                       ;random permutations

p5 const 3                             ;FM index = 3

p6 range 0 1 prec 2


f 2 8                                  ;field 2

p1 const 1

p2 seg (2 .01 5 .5 8 .01 ipl 1) prec 3 ;another density structure

p3 const .2

p4 item random (2000 2020 2400 2450 5300 2310 2350)

p5 seg (2 3 5 7 8 3 ipl 1) prec 1      ;FM index synchronous
                                       ;to density p2

p6 range 0 .5 prec 2                   ;panorama only in the
                                       ;left half


f 5 15                                 ;field 3

p1 const 1

p2 item swing (.3 .05 .2 .1 1)         ;a rhythm

p3 item swing (.3 .05 .2 .1 1)         ;no rest, no overlap


p4 range 100 200 prec 1

p5 seg [1 5]                           ;increasing FM index

p6 range .3 .7 prec 2                  ;only in the middle

;; -------------------
```

A version with random walks.
Note that different runs of CMask result partly in very different score files!

```
;; bells parameter file
{
f1 0 8193 10 1                          ;sine wave for foscil
}

f 0 20                                  ;field 1

p1 const 1

p2 range -.2 .2                         ;density difference
                                        ;-200 to +200 ms
accum limit .01 1                       ;absolut values between
                                        ;.01 to 1 sec

p3 range -.1 .1
accum mirror .1 1.5 init .5

p4 range -50 100                        ;tendency to higher
                                        ;frequencies
accum wrap 200 2000 init 300            ;but wrapped at (upper)
                                        ;boundary

p5 const 3                              ;FM index = 3

p6 range 0 1 prec 2
```

For more examples (timestretching and others) look in the folder "CMask examples".