MapBuss

A MapBuss contains named values and MapDefs that listen for changes in those values. MapDefs scale the data, remember the results and can attach responder functions to those outputs.

create a MapBuss and add MapDefs to it using .addDef. MapBuss updates cause dependent MapDefs to update their memory.

Create the MapBuss:

```
m = MapBuss.new;
```

.addDef -- a MapFunc will be added to the global MapDef library.

arguments go like this .addDef(name, input, output, mapItem, environment_init) where:

name - a useful, possibly descriptive name. You can access the mapdef in code by name like this: MapDef(name)

input - channel on MapBuss that will be the input source. array for many inputs to a MapDef function.

output - channel on MapBuss that will be the output destination. an array for many output from a MapDef function.

mapItem - either [[values],[lengths]], an Env, or a function

environment_init - optional. set the environmantal values for function mapItems. A key, value pairs array sets ~key = value wne given [\key, value]

There are two ways to create a one to one MapDef from an Envelope. Either pass 1) an array [[values],[lengths],curve] or 2) an actual envelope

example 1) add a mapping with an input and an output. input should range from 0-1

```
m.addDef(\test_xy, \test_inxy, \test_outxy, [[0,0.5,5],[0.75,0.25]]);
```

MapBuss.addDef(somename) creates a MapFunc in the global MapDef Library at \somename

```
//this is the mapping that was just built
MapDef(\test_xy);

//supply some input to the MapBuss
m.set(\test_inxy, 1);

//the output is written to the MapDef environment
MapDef(\test_xy).environment
```

example 2) make a mapping with an Envelope. input should range from 0-1

```
m.addDef(\test_env, \test_inenv, \test_outenv, Env.triangle(1,1));
m.set(\test_inenv, 0.5);

//see the output in the def environment
MapDef(\test_env).environment

//get the output
MapDef(\test_env).at(\test_outenv);
```

There is a lot of flexibility in creating MapDefs with custom functions. one-one, one-many, many-one, many-many are all possible

one to one mapping. the value of \tested_in is automatically passed as an argument and the result is placed at the output on the MapBuss

```
m.addDef(\tested, \tested_in, \tested_out, { |vv| vv * 2 });

//update the mapbuss
m.set(\tested_in, 0.75);

//check the updated output
MapDef(\tested).at(\tested_out);
```

a mapdef can have many outputResponders to react to updates in the environment
.addOutputResponder(uniquename, variable in environment, function)

```
MapDef(\tested).addOutputResponder(\tested_tested_in, \tested_out, { |vv|
      [\say, vv].postln;
});

m.set(\tested_in, 1.75);
```

MapBuss

one to one mapping using a custom function. Once the input is initialized, the function update in the back ground on it's own clock.
the default, once started, is to update 20 times a second. use .start and .stop to control the regular updates.

this will generate random numbers without an input using built in clock to trigger the funciton \none is an input that is by default always set to zero

```
m.addDef(\testFunc, \none, \testFunc_out, { |vv| 10.0.rand });

MapDef(\testFunc).start; //start updating

 //execute this line many times and you can see the output changing over time
MapDef(\testFunc).at(\testFunc_out);

MapDef(\testFunc).stop; //stop updating

//updates are still triggered when the input channel is updated.
m.set(\none, 0);
```

a more sophisticated MapDef that has memory using the optional environment_init agument, one can initialize values used in the function by adding name, value pairs array. keys like \asdf will appear like ~asdf in the mapItem function. the variables are locally scoped variables inside the mapItem function.

important to note that this means the mapItems have persistent memory. is possible to create scenarios where values decay over time or that accumulate, average, integrate, etc. with previous values

```
m = MapBuss.new;
m.addDef(\testFunc2, \testFunc2_in, \testFunc2_out,
      { |vv| vv * ~mul + ~add + 0.1.rand2 },
      [\mul, 5, \add, 10]
);
MapDef(\testFunc2).environment; //peek into the MapItems environment and see that
the default values are set.
m.set(\testFunc2_in, 1);
MapDef(\testFunc2).at(\testFunc2_out);

//some responders
MapDef(\testFunc2).addOutputResponder(\tf2_mul, \mul, { |vv| [\mul, vv].postln; });
MapDef(\testFunc2).addOutputResponder(\tf2_add, \add, { |vv| [\add, vv].postln; });
MapDef(\testFunc2).addOutputResponder(\tf2_out, \testFunc2_out, { |vv|
[\testFunc2_out, vv].postln; });

//the auto responders are here
MapDef(\testFunc2).outputResponders;
m.set(\testFunc2_in, 1);
```

MapBuss

an example creating one to many mapping using custom function.
as usual, the first output is the result of the las line of the function.
all other output values need to be set in the function.
set ~testFuncom_x = some value and \testFuncom_x on the MapBuss will be updated.

```
m.addDef(\testFuncom,
      \testFuncom_in,
      [\testFuncom_out, \testFuncom_x],
      { |vv| ~testFuncom_x = vv * ~mul; vv * ~mul + ~add + 0.1.rand2 },
      [\mul, 5, \add, 10, \testFuncom_x, 0]
);

//set the input and both output values are updated as a result
m.set(\testFuncom_in, 3);

MapDef(\testFuncom).at(\testFuncom_out); //output 0
MapDef(\testFuncom).at(\testFuncom_x); //output 1
```

create a many to many mapItem function
by passing arrays of channel names as input and output.
the mapItem function will not general output
until all input channels have been initialized

```
m.addDef(\testFuncmm,
      [\testFuncmm_in, \testFuncmm_inx],
      [\testFuncmm_out, \testFuncmm_x],
      { |vv| ~testFuncmm_x = vv * ~mul; vv * ~mul + ~add + 0.1.rand2 },
      [\mul, 5, \add, 10, \testFuncmm_x, 0, \testFuncmm_inx, 0]
);

//since there is no value at \testFuncmm_in
//a warning is issued and the function is not executed.
m.set(\testFuncmm_inx, 33);

//now both inputs are available and the function will execute
m.set(\testFuncmm_in, 4);

MapDef(\testFuncmm).at(\testFuncmm_out);
MapDef(\testFuncmm).at(\testFuncmm_x);
```