CtrlBuss

CtrlBuss manages CtrlDefs that link MapBuss and MatchBuss. Responders send updated mapping output to MatchDef environment and synths.

//make an instance of a CtrlBuss and add a CtrlDef.

```
c = CtrlBuss.new;

//add a MatchDef with a couple MapDefs
c.addDef(\tester, [\asdf], { |mk| [mk.name, \start, \outx, ~outx].postln; }, { |mk|
[mk.name, \stop].postln; }, [
      [\mapx, \inx, \outx, [[0,0.5,5.0],[0.75,0.25]]],
      [\mapy, \iny, \outy, [[0,0.5,5.0],[0.75,0.25]]]
]);

//the CtrlDef name \test is a lnink between the MatchDef named \test and the
MapDefs \,apx and \mapy
CtrlDef.all

//add a symbol to the MatchBuss and the MatchDef \test starts
c.add(\asdf);

//remove a symbol to the MatchBuss and the MatchDef \test stops
c.remove(\asdf);

//inpect some data that has accumulated in the MatchDef \test local memory
MatchDef(\tester).environment;

//input some control into the MapBuss
c.set(\inx, 1)

//see the result of the mapping - the code refactor makes this not functional.  no
values are written back to the buss automatically
c.get(\outx)
MatchDef(\tester).environment; //BUT \outx is written to the MatchDef environment
//because a CtrlDef is a mapping function from a MapBuss to client side 'node' or
MatchFunc

//inpect the contents of the MapBuss
c.map.buss //only the inputs are kept on the MapBuss.  The scaled output is sent to
the MatchFunc based on the mapItem in the MapFunc

//notice that MatchDef \test has been informed by the mappings
MatchDef(\tester).environment;
```

[server control]

CtrlBuss has wrappers for some basic scsynth server control

.onBoot(func) – sets a function that will execute upon scsynth server start
.boot – boots the scsynth server
.waitForBoot(func) – sets the onBoot function, starts the scsynth, executes the onBoot

.waitForNetworkBoot(func) – wait for all CtrlMesh sc3 hosts to boot scsynth,
then execute this function. This is a convenience wrapper for
CtrlMesh.bootAllServers(func) see [CtrlMesh]

CtrlBuss

[addMIDI]

.addMIDI constructs a basic messaging scheme for noteOn, NoteOff and CC messages

.addMIDI(chan) – chan is an optional channel for sending midi data to a CtrlMesh. see [CtrlMesh]

Incoming midi data from all sources are translated into spcialized MapBuss and MatchBuss messages.

[midi NoteOn MapBuss]
A midi noteOn will set the value of the note on the MapBuss at midiNote_[chan]n[num] where

value is the value of the midi message
chan is the channel of the midi message
num is the num of the midimessage

a message on channel 0 num 81 with a value of 127 will set \midiNote_c0n81 on the buss with the value of 127

[midi NoteOn MatchBuss]
A midi noteOn will add a symbol to the MatchBuss at midiNote_[chan]n[num] where

value is the value of the midi message
chan is the channel of the midi message

a message on channel 0 num 81 with a value of 127 will add \midiNote_c0n81 to the MatchBuss

a message on channel 0 num 81 with a value of 0 will remove \midiNote_c0n81 to the MatchBuss

[midi NoteOff MapBuss]
A midi noteOff will set the value of the note on the MapBuss at midiNote_[chan]n[num]

a message on channel 0 num 81 with a value of 127 will set \midiNote_c0n81 on the buss with the value of 0

[midi NoteOff MatchBuss]
A midi noteOff will remove a symbol to the MatchBuss at midiNote_[chan]n[num]

a message on channel 0 num 81 with a value of 0 will remove \midiNote_c0n81 to the MatchBuss

[midi CC and MapBuss]
A midiCC will set the value of the note on the MapBuss at midiCC_[chan]n[num] where

value is the value of the midi message
chan is the channel of the midi message
num is the num of the midimessage

a message on channel 0 num 81 with a value of x will set \midiCC_c0n81
on the buss with the value of x

CtrlBuss

[midi CC and MatchBuss]
A midiCC will add a symbol to the MatchBuss at midiNote_[chan]n[num] under certain conditions where

value is the value of the midi message
chan is the channel of the midi message

a message on channel 0 num 81 with a value of 127 will add \midiCC_c0n81 to the MatchBuss

a message on channel 0 num 81 with a value of 0 will remove \midiCC_c0n81 to the MatchBuss

with a synth controlled by EV-1 and fs-6

Midi is written to the MapBuss automatically. Values have names that indicate the protocol, type, channel and number. So for Midi CC a name like \MidiCC_c0n11 contains the value for Midi Continuous Control messages on channel 0 at number 11.

```
(
c = CtrlBuss.new(s);

c.waitForBoot({
      SynthDef(\test, { |freq = 440, gate = 0, amp = 0.25|
            Out.ar(0, SinOsc.ar(Lag.kr(freq,0.01)) * amp *
EnvGen.ar(Env.adsr(0.1,0,1, 0.25), gate, doneAction: 2) )
      }).add;

      //add the default midi to mapbuss scheme
      c.addMIDI;

      //add a MatchDef with a couple MapDefs
      c.addDef(\tester, [\midiCC_c0n81], { |mk| [mk.name, \start, \outx, ~outx,
\freq, ~freq].postln; }, { |mk| [mk.name, \stop].postln; }, [
            [\frx, \midiCC_c0n11, \freq, [[110,440,880],[0.75,0.25]]],
            [\mapx, \inx, \outx, [[0,0.5,5.0],[0.75,0.25]]],
            [\mapy, \iny, \outy, [[0,0.5,5.0],[0.75,0.25]]]
      ]);

      MatchDef(\tester).addSynth(\testsynth, \test, [\freq, {rrand(400,500)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.25], s, \addToHead);
});
)

//the function starts when you click the footpedal on the left
//just in case her in code
c.add(\midiCC_c0n81);
c.set(\freq, 0.25) //set fr on the buss to change the frequence of all the nodes
//remove a symbol to the MatchBuss and the MatchDef \test stops
c.remove(\midiCC_c0n81);
```

CtrlBuss

CtrlKeyWin - window converts keypresses like a into \k_a and add/remove them on the buss

CtrlKeyWin will call .add(\key_symbol) and .remove(\key_symbol) on whatever object is passed as the second argument. Anything with .add .remove methods can be used (Array, MatchBuss, CtrlBuss, etc)

A specialized CtrlKeyWin is added to a CtrlBuss using

.addKeyWin(chan) where chan is an option symbol specifying which CtrlMesh channel for sending data. see [CtrlMesh]

```
c = CtrlBuss.new;  //instance a very plain CtrlBuss
c.addKeyWin; //add key window observe that the window displays symbols on the buss

//these are the keys mapped by default
c.keyWin.keymap.keysValuesDo({ |kk, vv| [kk, vv].postln }); "";

//add an additional keyboard character to the map. + becomes \k_plus on the buss
c.keyWin.keymap.put($+, \k_plus); //add a symbol for the + key
```

```
(
c = CtrlBuss.new(s);

c.waitForBoot({
	SynthDef(\test, { |freq = 440, gate = 0, amp = 0.25|
		Out.ar(0, SinOsc.ar(Lag.kr(freq,0.01)) * amp *
EnvGen.ar(Env.adsr(0.1,0,1, 0.25), gate, doneAction: 2) )
	}).add;

	c.addKeyWin;

	//add a MatchDef with a couple MapDefs
	c.addDef(\tester, [\k_a], { |mk| [mk.name, \start, \outx, ~outx, \freq,
~freq].postln; }, { |mk| [mk.name, \stop].postln; }, [
		[\frx, \fr, \freq, [[110,440,880],[0.75,0.25]]],
		[\mapx, \inx, \outx, [[0,0.5,5.0],[0.75,0.25]]],
		[\mapy, \iny, \outy, [[0,0.5,5.0],[0.75,0.25]]]
	]);

	MatchDef(\tester).addSynth(\testsynth, \test, [\freq, {rrand(400,500)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.25], s, \addToHead);

});
)

//now press the 'a' key
```

CtrlBuss

ctrlkeywin - to start stop a couple sounds and to start stop a couple maps

if it is important to have the node not get data directly streamed in, the us a different target (not the MatchDef default). then the node will start with values from the environment (as updated by the mappings) but will not receive the data while running on the server

```
(
//when given a Server, CtrlNode will build a local Group for each MatchDef.

c = CtrlBuss.new(s);
c.waitForBoot({
	SynthDef(\sin, { |freq = 440, gate = 0, amp = 0.25|
		Out.ar(0, SinOsc.ar(Lag.kr(freq,0.01)) * amp *
EnvGen.ar(Env.adsr(0.1,0,1, 0.25), gate, doneAction: 2) )
	}).add;

	SynthDef(\pmo, { |freq = 440, mod = 220, gate = 0, amp = 0.25|
		var sig;
		//sig = SinOsc.ar(Lag.kr(freq,0.01)) * amp *
EnvGen.ar(Env.adsr(0.1,0,1, 0.25), gate, doneAction: 2);
		sig = PMOsc.ar(Lag.kr(freq,0.05), Lag.kr(mod,0.05), 0.5, 0.5) * amp *
EnvGen.ar(Env.adsr(0.1,0,1, 0.25), gate, doneAction: 2);
		Out.ar(0, sig);
	}).add;

	c.addKeyWin;
	c.addDef(\test_a, [\k_a], { |mk| [mk.name, \start, \outx, ~outx, \freq,
~freq].postln; }, { |mk| [mk.name, \stop].postln; },
		[[\frax, \fra, \freq, [[110,440,880],[0.75,0.25]]]]
	);

	MatchDef(\test_a).addSynth(\asin, \sin, [\freq, {rrand(400,500)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.25]);

	c.addDef(\test_s, [\k_s], { |mk| [mk.name, \start, \outx, ~outx, \freq,
~freq, \mod, ~mod].postln; }, { |mk| [mk.name, \stop].postln; },
		[[\frsf, \none, \freq, { |vv| ~idx = ~idx + 0.5;  ~idx.wrap(400,600) },
[\idx, 100]],
		[\frsx, \none, \mod, { |vv| SystemClock.seconds.sin + 2 * 1000 }]]
	);

	MatchDef(\test_s).addSynth(\spmo, \pmo, [\freq, {rrand(600,500)}, \mod,
{rrand(600,500)}, \dur, {rrand(0.125,1.0)}, \amp, 0.25]);

	MatchDef(\test_s).addSynth(\spmo, \pmo, [\freq, {rrand(600,500)}, \mod,
{rrand(600,500)}, \dur, {rrand(0.125,1.0)}, \amp, 0.25], s);

	c.addDef(\test_d, [\k_d], { |mk| MapDef(\frsx).toggle }, { |mk| });
	c.addDef(\test_f, [\k_f], { |mk| MapDef(\frsf).start }, { |mk|
MapDef(\frsf).stop });

});
)

c.set(\fra, 0.9)
```

CtrlBuss

when building from a ctrlBuss, first build the MatchDef, then add the synth to it using .addSynth.

c = CtrlBuss.new

```
	c.addDef(\test_a, [\k_a], { |mk| [mk.name, \start, \outx, ~outx, \freq, ~freq].postln; }, { |mk| [mk.name, \stop].postln; },
		[
[\frax, \fra, \freq, [[110,440,880],[0.75,0.25]]]
		]
	);

	c.addSynth(\test_a, \asin, \sin, [\freq, {rrand(400,500)}, \dur, {rrand(0.125,1.0)}, \amp, 0.25]);
```