MatchBuss

MatchBuss maintains a sequence of logical flags that is tested for matches by MatchDefs. Flags are ordered in sequence by the time they are inserted. MatchDefs start/stop when their match pattern occurss in the sequence. This is useful for quickly prototyping musical functions that do not need to be hardcoded to controllers, controlling many musical objects from hardware and creating modal control of many musical objects with the same hardware using flags as layers.

Create an instance of a matchbuss:

```
m = MatchBuss.new;

//add a MatchDef
m.addDef(\tester, [\test], { |self| [self.name, \start].postln }, { |self|
[self.name, \stop].postln });

//add a flag to the MatchBuss sequence
m.add(\test);
m.remove(\test);

//either start or stop the key
m.toggle(\test)

//more MatchDefs
m.addDef(\tester1a, [\scene_1, \k_a], { |self|
      [self.name, \start].postln
}, { |self|
      [self.name, \stop].postln
});

m.addDef(\tester1q, [\scene_1, \k_q], { |self|
      [self.name, \start].postln
}, { |self|
      [self.name, \stop].postln
});

m.addDef(\tester1a, [\scene_2, \k_a], { |self|
      [self.name, \start].postln
}, { |self|
      [self.name, \stop].postln
});

m.addDef(\tester2q, [\scene_2, \k_q], { |self|
      [self.name, \start].postln
}, { |self|
      [self.name, \stop].postln
});


m.addDef(\tester1a, [\scene_2, \k_a, \k_s, \k_v], { |self|
      [self.name, \start].postln
}, { |self|
      [self.name, \stop].postln
});

m.addDef(\tester1a, [\scene_2, \k_a, \k_v, \k_s], { |self|
      [self.name, \start].postln
}, { |self|
      [self.name, \stop].postln
});
```

MatchBuss

history, .within and .recent

.within test if this matchFunc has matched within x seconds

.recent tests if another matchFunc has matched within x seconds

```
m = MatchBuss.new;

//add a MatchDef
m.addDef(\test, [\test], { |self| [self.name, \start].postln }, { |self|
[self.name, \stop].postln });
m.addDef(\testing, [\testing], { |self| [self.name, \start, \test,
SystemClock.seconds - MatchDef(\test).at(\lastTime) < 10].postln }, { |self|
[self.name, \stop].postln });
m.addDef(\tested, [\tested], { |self| [self.name, \start, self.within(1),
self.recent(\test, 10)].postln }, { |self| [self.name, \stop].postln });

m.toggle(\test);
m.toggle(\testing);
m.toggle(\tested);
MatchDef(\test).environment
```

use the synth management built into MatchFunc

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

      SynthDef(\grain, { |freq = 440, dur = 0.1, curve = -8, amp = 0.25|
            Out.ar(0, SinOsc.ar(freq) * amp * EnvGen.ar(Env.perc(0.01,dur,curve),
doneAction: 2) )
      }).add;
});

m = MatchBuss.new;
m.addDef(\tester, [\test], { |self| [self.name, \start].postln }, { |self|
[self.name, \stop].postln });

//functions are evaluated once and results are put in the environment
MatchDef(\tester).addSynth(\tester_test, \test, [\freq, {rrand(400,500)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.25], s, \addToHead);
)

m.add(\test);// invoke the matchDef
m.remove(\test);// stop the matchDef

//synth arguments pulled from preset,
//then updated with values from the environment.
MatchDef(\tester).environment; //notice the args from .addSynth in the environment.

m.add(\test); //invoke it again
m.remove(\test); //stop it again
MatchDef(\tester).environment; //notice the addition of \interOnset (in seconds)
```

MatchBuss

add a second MatchDef to demo ways to use the environment

note: synths managed by a matchDef are started BEFORE the startFunc executes
changes to variables in the environment will be available on the next startFunc.

```
m.addDef(\tested, [\tester], { |self|
      ~freq = rrand(400,500);
      [self.name, \start].postln
}, { |self|
      [self.name, \stop].postln
});

MatchDef(\tested).setTarget(s, \addToHead);  //add a default group.

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

MatchDef(\tested).environment; //note the value of freq

// inpect the values at all the steps along the way
m.add(\tester); //invoke it again

MatchDef(\tested).synths.at(\test_ed).get(\freq, {|vv| [\freq, vv].postln; });

//synthe is started with \freq = 493
MatchDef(\tested).environment; //notive the addition of \interOnset (in
seconds)  //473

m.remove(\tester); //stop it again

MatchDef(\tested).environment; //notive the addition of \interOnset (in seconds)

m.add(\tester); //invoke it again

MatchDef(\tested).synths.at(\test_ed).get(\freq, {|vv| [\freq,
vv].postln; }); //473

MatchDef(\tested).environment; //notive the addition of \interOnset (in seconds)

m.remove(\tester); //stop it again


// .set will send values to all synths in the MatchDefs default group

// reboot
m.addDef(\tested, [\tester], { |self|
      ~freq = rrand(400,500);
      [self.name, \start].postln
}, { |self|
      [self.name, \stop].postln
});

MatchDef(\tested).setTarget(s, \addToHead);  //add a default group.

MatchDef(\tested).addSynth(\test_ed, \test, [\freq, {rrand(400,500)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.25]);

//MatchDef(\tested)
m.add(\tester);
MatchDef(\tested).set(\freq, 440);
m.remove(\tester);
```

MatchBuss

the MatchFunc will execute in this order: evaluate the startFunc, start the the internal synths, start the internal tasks

.set args of the synth from startFunc

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

	SynthDef(\grain, { |freq = 440, dur = 0.1, curve = -8, amp = 0.25|
		Out.ar(0, SinOsc.ar(freq) * amp * EnvGen.ar(Env.perc(0.01,dur,curve),
doneAction: 2) )
	}).add;
});

m = MatchBuss.new;
m.addDef(\tested, [\tester], { |self| ~freq = rrand(400,500); self.set(\freq,
~freq); [self.name, \start].postln }, { |self| [self.name, \stop].postln });

)


MatchDef(\tested).setTarget(s, \addToHead);
MatchDef(\tested).addSynth(\test_ed, \test, [\freq, {rrand(400,500)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.25], s, \addToHead);
MatchDef(\tested).environment; //note the value of freq

m.add(\tester)
m.remove(\tester)
```

add more synths to the same MatchDef

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

	SynthDef(\grain, { |freq = 440, dur = 0.1, curve = -8, amp = 0.25|
		Out.ar(0, SinOsc.ar(freq) * amp * EnvGen.ar(Env.perc(0.01,dur,curve),
doneAction: 2) )
	}).add;


	m = MatchBuss.new;
	m.addDef(\tester, [\tester], { |self| [self.name, \start].postln }, { |self|
[self.name, \stop].postln });
	MatchDef(\tester).setTarget(s, \addToHead);

	MatchDef(\tester).addSynth(\tester_test, \test, [\freq, {rrand(10,20)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.25]);
	MatchDef(\tester).addSynth(\tester_test2, \test, [\freq, {rrand(10,20)},
\dur2, {rrand(0.125,1.0)}, \amp2, 0.25]);
	MatchDef(\tester).addSynth(\tester_test3, \test, [\freq, {rrand(500,600)},
\dur3, {rrand(0.5,1.0)}, \amp3, 0.25]);
});
```

MatchBuss

every argument for a synth is updated with the environment. when using many synths, choose arguments intentionally because shared argument names will receive the same value

```
m.add(\tester); //start the match, each synth sources the random function for freq.
the function for \freq is consecutively over written
MatchDef(\tester).environment //note that \freq is one function. the last write
into the environment at \freq will apply to all synths using \freq
MatchDef(\tester).set(\freq, 440);  //now they all have the same freq
m.remove(\tester);


//over write those synths
MatchDef(\tester).addSynth(\tester_test, \test, [\freq, {rrand(200,300)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.25], s, \addToHead);
MatchDef(\tester).addSynth(\tester_test2, \test, [\freq2, {rrand(400,500)}, \dur2,
{rrand(0.125,1.0)}, \amp2, 0.25], s, \addToHead);
MatchDef(\tester).addSynth(\tester_test3, \test, [\freq3, {rrand(500,600)}, \dur3,
{rrand(0.5,1.0)}, \amp3, 0.25], s, \addToHead);

m.add(\tester);
m.remove(\tester);

MatchDef(\tester).environment //see that freq, freq2, freq3 are now functions that
are independant because they have their own name in the environment
```

add synths in the startFunc using .startSynth

synths from .startSynth are not added to the synthList. use the environment as arguments, but these arguments do not set the environment.

```


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

      m = MatchBuss.new;
      m.addDef(\tester, [\tester], { |self|
            [self.name, \start].postln;
            ~freq1 = rrand(220,440);
            ~freq2 = rrand(220,660);
            ~freq3 = rrand(220,880);

            self.startSynth(\test1, \test, [\freq, ~freq1, \dur,
{rrand(0.125,1.0)}, \amp, 0.25]);
            self.startSynth(\test2, \test, [\freq, ~freq2, \dur,
{rrand(0.125,1.0)}, \amp, 0.25]);
            self.startSynth(\test3, \test, [\freq, ~freq3, \dur,
{rrand(0.125,1.0)}, \amp, 0.25]);
      }, { |self|
            [self.name, \stop].postln
      });
});

m.add(\tester);
m.remove(\tester);
```

MatchBuss

using the MatchDef grain feature

```
m = MatchBuss.new;
s.reboot;

SynthDef(\grain, { |freq = 440, dur = 0.1, curve = -8, amp = 0.25|
      Out.ar(0, SinOsc.ar(freq) * amp * EnvGen.ar(Env.perc(0.01,dur,curve),
doneAction: 2) )
}).add;

m.addDef(\tester, [\test], { |md| [md.name, \start].postln;
{ 4.do({ md.grain(\tester_test); 0.075.wait }) }.fork }, { |self| [self.name,
\stop].postln });
MatchDef(\tester).addGrain(\tester_test, [\grain, [\freq, {rrand(400,500)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.075], \default, \addToHead]);

MatchDef(\tester).environment

m.add(\test)
m.remove(\test)
```

```
//grain
m = MatchBuss.new;
s.reboot;

SynthDef(\grain, { |freq = 440, dur = 0.1, curve = -8, amp = 0.25|
      Out.ar(0, SinOsc.ar(freq) * amp * EnvGen.ar(Env.perc(0.01,dur,curve),
doneAction: 2) )
}).add;

m.addDef(\tester, [\test], { |md| [md.name, \start].postln;
{ 100.do({ md.grain(\tester_test); ~wait.wait }) }.fork }, { |self| [self.name,
\stop].postln });
MatchDef(\tester).put(\wait, 0.075); //puts ~wait into the environment
MatchDef(\tester).addGrain(\tester_test, [\grain, [\freq, {rrand(400,500)}, \dur,
{rrand(0.125,1.0)}, \amp, 0.075], \default, \addToHead]);

MatchDef(\tester).environment;  //wait is in there

m.add(\test);
MatchDef(\tester).set(\freq, 440);  //set the freq environment
MatchDef(\tester).set(\amp, 0.05);
MatchDef(\tester).set(\dur, { [0.05, 0.25].choose });  //replace the dur value with
a function and it will be evaluated fore each grain
MatchDef(\tester).set(\freq, { rrand(400,500) });
m.remove(\test);
MatchDef.all

MatchDef(\tester).set(\amp, 0.05, \dur, { [0.05, 0.25].choose });
```

MatchBuss

using the MatchDef loop and task features

use the startLoop feature to create a repeating block of code

```
(
m = MatchBuss.new;
m.addDef(\tester, [\test], { |md|
      [md.name, \start].postln;
      md.startLoop({ |ii| [ii, SystemClock.seconds].postln });
}, { |md|
      [md.name, \stop].postln

});
)

//start and stop the match.  notice that by default the interonset is 0.1.
m.add(\test);
m.remove(\test);

// matchDef keeps data for the tasks in <>taskList.
// The identity dictionary keys are the names of the task.
MatchDef(\tester).taskList;
// The .startLoop method is a short cut to create loops quickly
// in is given the name \prLoop
// the tasks are kept in <>tasks
MatchDef(\tester).tasks;

//address the interonset for \prLoop is available on the MatchDef environment
MatchDef(\tester).at(\prWait);

//.setWait updates the interonset.
MatchDef(\tester).setWait(0.5);
MatchDef(\tester).taskList;  //note the updated value (is this no longer
referenced?)
MatchDef(\tester).at(\prWait); //note the updated value

//note the changed interonset
m.add(\test);
m.remove(\test);

//.setLoop can change the amount of repetition
MatchDef(\tester).setLoop(6);
MatchDef(\tester).at(\prLoopNum); //note the change in the matchdef environment

m.add(\test);
m.remove(\test);

MatchDef(\tester).setLoop(inf);
MatchDef(\tester).setWait(0.25);

m.add(\test);
m.remove(\test);
```

MatchBuss

//recompile

```
//note, in this order startFunc, synths, tasks are evaluated
(
m = MatchBuss.new;
m.addDef(\tester, [\test], { |md|
      [md.name, \start].postln;
      md.startLoop({ |ii| [ii, SystemClock.seconds].postln });
      md.setLoop(rrand(2,8));  //this value sets the loop that is declared above,
the loop is started *after* this block of code
}, { |md|
      [md.name, \stop].postln

});
)

m.add(\test);
m.remove(\test);
```

//recompile

use the task feature

```
m = MatchBuss.new;  //make an ampty MatchBuss

//simple matchDef
m.addDef(\tester, [\test], { |md| [md.name, \start].postln; }, { |md| [md.name,
\stop].postln });

//add a task called \task1 using .addTask
// .addTask args are [name, loopFunc, stopFunc, optional key value pairs
MatchDef(\tester).addTask(\task1, { |md, ii| [\task1, ii, SystemClock.seconds,
~asdf].postln; },{ |ii| [\task1, \stopTask, ii].postln }, \wait, 0.25, \loopNum,
100);

m.add(\test);
MatchDef(\tester).set(\asdf, 100.rand);
MatchDef(\tester).environment
m.remove(\test);

// .setWait with the name as the first arg will set the interonset of the named
task
MatchDef(\tester).setWait(\task1, 0.1);

m.add(\test);
m.remove(\test);

MatchDef(\tester).setLoop(\task1, 4);

m.add(\test);
m.remove(\test);
```

MatchBuss

//recompile

start a synth and change it with a task (emulate changing with external control)

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

	SynthDef(\grain, { |freq = 440, dur = 0.1, curve = -8, amp = 0.25|
		Out.ar(0, SinOsc.ar(freq) * amp * EnvGen.ar(Env.perc(0.01,dur,curve),
doneAction: 2) )
	}).add;

	m = MatchBuss.new;
	m.addDef(\tester, [\test], { |self| [self.name, \start].postln }, { |self|
[self.name, \stop].postln });

	//functions are evaluated once and results are put in the environment
	MatchDef(\tester).addSynth(\tester_test, \test, [\freq, {rrand(400,500)},
\dur, {rrand(0.125,1.0)}, \amp, 0.25], \default, \addToHead);

	MatchDef(\tester).addTask(
		\task1,
		{ |md, ii| md.set(\freq, rrand(600,800)) },
		{ |md, ii| [\task1, \stopTask, ii].postln },
		\wait, 0.25, \loopNum, inf
	);

	MatchDef(\tester).setTarget(s, \addToHead);
});

)

m.add(\test);
m.remove(\test);
```

MatchBuss

record 0.25s of noise, playback in grains

```
(
s.reboot.waitForBoot({
	b = Buffer.alloc(s, 44100 * 0.25, 2);

	SynthDef(\rec, { |buf, freq = 440, gate = 0, amp = 0.25|
		RecordBuf.ar(PinkNoise.ar(1.0!2), buf, 0, 0.75, doneAction: 2);
	}).add;

	SynthDef(\grain, { |buf, rate = 1, freq = 440, dur = 0.1, curve = -8, amp =
0.25|
		var sig;
		sig = PlayBuf.ar(2, buf, rate, loop: 1);
		sig = BPF.ar(sig, freq, 0.125);
		Out.ar(0, sig * amp * EnvGen.ar(Env.perc(0.01,dur,curve), doneAction:
2) )
	}).add;

	m = MatchBuss.new;
	m.addDef(\tester, [\test], { |self| [self.name, \start].postln }, { |self|
[self.name, \stop].postln });
	MatchDef(\tester).setTarget(s, \addToHead);

	//functions are evaluated once and results are put in the environment
	MatchDef(\tester).addSynth(\test_rec, \rec, [\buf, b, \rate, 1, \freq,
{rrand(400,500)}, \dur, {rrand(0.125,1.0)}, \amp, 0.25], \default, \addToHead);
	MatchDef(\tester).addGrain(\test_grain, \grain, [\buf, b, \rate, 1, \freq,
{rrand(400,500)}, \dur, {rrand(0.125,1.0)}, \amp, 0.25], \default);
	MatchDef(\tester).addTask(
		\task1,
		{ |md, ii| md.grain(\test_grain);  md.set(\freq, rrand(220,880)) },
		{ |md, ii| [\task1, \stopTask, ii].postln },
		\wait, 0.125, \loopNum, inf
	);

});

)

m.add(\test);
m.remove(\test);
```

MatchBuss

record 0.25s of noise, playback in grains with a pitchshift on the grains

```
(
s.reboot.waitForBoot({
	b = Buffer.alloc(s, 44100 * 0.25, 2);

	SynthDef(\rec, { |buf, freq = 440, gate = 0, amp = 0.25|
		RecordBuf.ar(PinkNoise.ar(1.0!2), buf, 0, 0.75, doneAction: 2);
	}).add;

	SynthDef(\grain, { |buf, rate = 1, freq = 440, dur = 0.1, curve = -8, amp =
0.25|
		var sig;
		sig = PlayBuf.ar(2, buf, rate, loop: 1);
		sig = BPF.ar(sig, freq, 0.125);
		Out.ar(0, sig * amp * EnvGen.ar(Env.perc(0.01,dur,curve), doneAction:
2) )
	}).add;

	SynthDef(\pitch, { |rate = 1, gate = 0, amp = 0.25|
		var sig;
		sig = In.ar(0, 2);
		sig = PitchShift.ar(sig, 0.25, rate, 0.05, 0.125, 1.5);
		Out.ar(0, sig);
	}).add;

	m = MatchBuss.new;
	m.addDef(\tester, [\test], { |self| [self.name, \start].postln }, { |self|
[self.name, \stop].postln });
	MatchDef(\tester).setTarget(s, \addToHead);

	//functions are evaluated once and results are put in the environment
	MatchDef(\tester).addSynth(\test_rec, \rec, [\buf, b, \rate, 1, \freq,
{rrand(400,500)}, \dur, {rrand(0.125,1.0)}, \amp, 0.25], \default, \addToHead);
	MatchDef(\tester).addGrain(\test_grain, \grain, [\buf, b, \rate, 1, \freq,
{rrand(400,500)}, \dur, {rrand(0.125,1.0)}, \amp, 0.25], \default);
	MatchDef(\tester).addSynth(\test_pitch, \pitch, [\rate, 1.5], \default,
\addToTail);
	MatchDef(\tester).addTask(
		\task1,
		{ |md, ii| md.grain(\test_grain);  md.set(\freq, rrand(220,880)) },
		{ |md, ii| [\task1, \stopTask, ii].postln },
		\wait, 0.125, \loopNum, inf
	);

});

)


m.add(\test);
MatchDef(\tester).set(\rate, 0.75);
m.remove(\test);
```