**CtrlMesh** - creates a mesh network of SC nodes. CtrlMesh instances assemble automatically via braodcast messaging or specific host mesh networks can be made explicitly. Each CtrlMesh maintains connections every other CtrlMesh host identified on the local network. Hosts can subscribe to channels of data that is sent via unicast. Hosts can send data to all hosts using the network broadcast layer via multicast. Remote boot, remote re-compile, remote code evaluation, remote post and ping utilities are available via methods.

CtrlMesh.new(address, ...options)

addr is a string containing a valid local network interface address. for example "10.0.0.3"

options can be added in any order but must conform to key calue pair structure. valid keys are \remotes, \broadcast, \server where:

\remotes is followed by an array of ip address strings
\broadcast is true or false
\server is a specific scsynth. Server.defualt is used otherwise

There are two methods for building a CtrlMesh network. the broadcast method is default and will use the local network broadcast address to announce hosts to the network. The explicit method does not use the network broadcast address and uses a list of ip addresses to build the CtrlMesh.

1. use broadcast announcements to opt-in to the CtrlMesh

create a CtrlHost on an IPv4 network address and announce the hosts details to the network. Other nodes respond with their details and connections are established between the nodes creating a full mesh. This method uses broadcast messages and hosts can be started asynchronously.

```
//IPv4 address of the network adapter on the subnet with other CtrlMesh nodes
c = CtrlMesh.new("10.0.0.4");

CtrlHost.hosts;  //all the hosts connected to the network
```

2. Explicitly declare a CtrlMesh network

```
c = CtrlMesh.new("10.0.0.4", \remotes, ["10.0.0.4", "10.0.0.6"]);
```

OR an explicitly constructed CtrlMesh can be constructed in piecemeal but must be manually synchronized. First start the mesh and declare the hosts on all nodes of the network

```
c = CtrlMesh.new("10.0.0.4", \broadcast, false); //must manually disable .announce
c.remotes = ["10.0.0.4", "10.0.0.6"];  //declare network host connections

// then announce details to complete the network mesh connections.
// if announce executes before all hosts are listening,
// then those hosts will not establish return connections

c.announce; //send local host details to remotes
```

CtrlMesh has several administrative network tools

.put, .at, .post, .ping, .evaluate, .recompile, .subscribe, .unsubscribe

//the following utilities operatate on the \config channel.  //all nodes, by default, subscribe to channels \config and \all

//you can put strings, symbols and numbers into the environment of the network nodes
c.put(\test, 100); //now, on all remote hosts c.at(\test) = 100
c.put(\thistest, "This is a a test");
c.put(\testsymbol, \thisisasymbol);

//read data from the shared network environment
c.at(\test)

//post a message to a remote host
c.post(\host_pcwin, "this is a test");

//ping \host_pcwin to get the round trip network latency time
c.ping(\host_pcwin)

//interpret a string as code on a remote host.  this feature is disable for security reasons.  see [evaluate] see [security]
c.evaluate(\host_pcwin, "x = 100.rand.postln");  //x is random value and post on remote host

//interpreat a string as code on a remote host and send notification back to the local post window
c.evaluate(\host_pcwin, "x = 100.rand.postln", \notify);

//recompile the remote host see [recompile] see [security] see [interpret]
c.recompile(\host_pcwin);

//boot the scsynth of a remote host
c.bootRemoteServer(\host_pcwin)

//boot scsynth for all type \sc3 hosts in hte CtrlMesh
c.bootAllServers(func)

where

func is a function that will be executed when all network servers have completed booting.

**[evaluate]**
.evaluate presents major security risk and should only be enable on a closed private network - a network that does not connect beyond the local stage/studio instrument setup. a basic host permission validation disallows unpermitted code. this feature should not be enabled.

by default, evaluate is is disabled.

there are TWO steps required to enable .evaluate to function:

1. permission to send evaluate code to a host must be set for each remote host individually.
set .permission on the local host to an array of ip address strings.

c.network.permission_([\host_Node]); //allow \host_Node to send code for evauation

2. evaluate must be enabled with an additional step

**[security]**
CtrlMesh builds scsynth server that bind to "0.0.0.0" which is a major security risk. This allows network clients to control scsynth remotely which is only useful in multiclient synthesis situations. Again, recommended use is for closed private network only.

**[CtrlMesh data network channels]**
hosts in a CtrlMesh communicate using channels to control the amount of network traffic created when sending data messages.

There are two special case channels \config and \all.

CtrlMesh network utilites like .post, .ping, .evaluate, .recompile, .subscribe, .unsubscribe use the \config channel

\all channel should not be used outside of test and debug application.

A node can receive data placed on a channel of the network by subscribing. subscriptions can be created for any node from any node on the established network.

m = CtrlMesh.new("10.0.0.3");
m.subscribe(\host_Node, \ch1); //host_Node is a remote host being subscribed to \ch1 from here

A node can stop receiving data on a channel using unsubscribe. usubscribe can be initiated from any node.

m.unsubscribe(\host_Node, \ch1); //host_Node is a remote host usubscribed from \ch1

**other default channels**
The default simple midi interface sends data on channel \midi unless otherwise configured. see .addMIDI in [CtrlBuss]

The default key window interface sends data on channel \key unless otherwise configured. see .addKeyWin in [CtrlBuss]

**Network connected MapBuss Example**

```
//
// local host (remote below)
//

m = MapBuss.new;
c = CtrlMesh.new("10.0.0.3");
CtrlHost.hosts //see the other hosts

m.addDef(\testfunc, \test_in, \test_out, { |vv| vv * 2 });
m.set(\test_in, 1.0.rand); //update the mapbuss
MapDef(\testfunc).at(\test_out); //check the updated output

//add a custom handler function to the .set method of the MapBuss to send data over
the network
m.addHandler(\set, { |kk, vv, chan| [kk, vv, chan].postln; c.send(chan, \set, kk,
vv) });

///set the buss \test_in send the data on channel \all
m.set(\test_in, 1.0.rand, \all); //update the mapbuss
MapDef(\testfunc).at(\test_in);
m.buss

c.subscribe(\host_Node, \test);
m.set(\test_in, 1.0.rand, \test); //update the mapbuss
```

```
//
//remote host
//

m = MapBuss.new;
c = CtrlMesh.new("10.0.0.4");
CtrlHost.hosts //see the other hosts
//add a custom function to the OSC '/data' responder for all \set messages
c.addHandler(\set, { |...msg| msg.pairsDo({ |kk, vv| m.set(kk, vv) }) });
m.buss //see the updated data received from the remote host
```

### Network connected MatchBuss Example

```
//
//local host (remote below)
//

m = MatchBuss.new;
c = CtrlMesh.new("10.0.0.3");

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

//input from elsewhere
m.add(\test);
m.remove(\test);

//add custom handlers to .add and .remove methods to send data over the network
m.addHandler(\add, { |kk, chan| [kk, chan].postln; c.send(chan, \add, kk) });
m.addHandler(\remove, { |kk, chan| [kk, chan].postln; c.send(chan, \remove, kk) });

//send add and remove on channel \all
m.add(\test, \all);
m.remove(\test, \all);
```

```
//
//remote host
//

m = MatchBuss.new;
c = CtrlMesh.new("10.0.0.4");

// add a custom functions to the OSC '/data' responder
// for all \add and \remove messages
c.addHandler(\add, { |vv| m.add(vv) });
c.addHandler(\remove, { |vv| m.remove(vv) });
m.buss
```

**Network connected CtrlBuss Example**

```
//
//local host
//

c = CtrlBuss.new;
c.addNetwork("10.0.0.3");//, \maxLogins, 8, \protocol, \udp, \remotes, ["10.0.0.4",
"10.0.0.6"]
CtrlHost.hosts

/*
//if you've already configured a server add it to the CtrlBuss.
c = CtrlBuss.new(s);
c.addNetwork("10.0.0.3");  //if the server has an address other than 127.0.0.1 then
you don't have to argue it here
*/


c.addDef(\test, [\k_a], { |self| [self.name, \start].postln }, { |self| [self.name,
\stop].postln });

//send mapbuss and matchbuss changes to \chan1
c.set(\test_in, 1.0.rand, \all);  //random number is on all the local and remote
busses
c.add(\k_a, \all); //\k_a is on all the local and remote busses
c.remove(\k_a, \all); //\k_a is on all the local and remote busses
```

```
//
//remote host
//
m = MatchBuss.new;
c = CtrlMesh.new("10.0.0.4");
c.addDef(\test_remote, [\k_a], { |self| [self.name, \start].postln }, { |self|
[self.name, \stop].postln });
```

**Network connected CtrlBuss(s) with MIDI example**

```
//local host
c = CtrlBuss.new;
c.addNetwork("10.0.0.3");
c.addMIDI;
CtrlHost.hosts;
c.map.buss; //look for changes on the buss
c.match.buss;
```

```
//remote host
c = CtrlBuss.new;
c.addNetwork("10.0.0.4");
c.addMIDI;
CtrlHost.hosts;
c.map.buss; //look for changes on the buss
c.match.buss;
```

```
//
// scaling example 1 - local host starts and stops synths on multiple servers
//

(
c = CtrlBuss.new(s);
c.addNetwork("10.0.0.3");
)

(
c.waitForNetworkBoot({

	~synth = SynthDef(\test_sin, { |outBus = 0, freq = 440, freq2 = 440, amp =
0.5, gate = 0|
		var sig;
		sig = SyncSaw.ar(freq2 + BrownNoise.kr(2.0), freq * 0.5 +
BrownNoise.kr(3.0)) * amp;
		sig = sig * EnvGen.ar(Env.adsr(0.1,0.1,1,0.25), gate, doneAction: 2);
		Out.ar(outBus, sig);
	});

	~synth.send(CtrlHost(\host_Node).srv);
	~synth.send(s);

	c.addDef(\test, [\k_a], { |cc| [cc.name, \start].postln }, { |cc|
[cc.name, \stop].postln }, [
		[\frx, \midiCC_c6n64, \freq, { |vv| vv = vv / 127; ~freq = 220 +
(vv*660) }],
		[\fry, \midiCC_c0n11, \freq2, { |vv| vv = vv / 127; ~freq2 = 220 +
(vv*660) * 0.1 }],
	]);

	MatchDef(\test).addSynth(\test1, \test_sin, [\outBus, 0, \freq, 100, \freq2,
100, \dur, 10.25, \amp, 0.5], CtrlHost(\host_Node).srv);

	MatchDef(\test).addSynth(\test3, \test_sin, [\outBus, 0, \freq, 100, \freq2,
100, \dur, 10.25, \amp, 0.5], s);

	c.addKeyWin;
});
)
```

```
//remote host
(
c = CtrlBuss.new;
c.addNetwork("10.0.0.4");
```

```
//
// scaling example 2 - local ctrlhosts control own servers
//

//local host
(
c = CtrlBuss.new;
c.addNetwork("10.0.0.3");
c.addMIDI;
//start the server and execute this function as soon as the server is up
c.waitForBoot({
      SynthDef(\test, { |outBus = 0, freq = 440, amp = 0.25, gate = 0|
            var sig = SinOsc.ar(freq) * amp;
            sig = sig * EnvGen.ar(Env.adsr(0.01,0.01,1,0.25), gate, doneAction:2);
            Out.ar(outBus, sig);
      }).add;
      //add a matchdef
      c.addDef(\test_local, [\k_a], { |self| [self.name, \start].postln }, { |self|
[self.name, \stop].postln },
            [[\pedal, \midiCC_c0n11, \freq, { |vv| vv = vv / 127; 220 +
(vv*660) }]]
      );
      //add a synth to the matchdef
      MatchDef(\test_local).addSynth(\ka_test, [\test, [\freq, rrand(440,660),
\amp, rrand(0.1,0.25)]]);
      c.addKeyWin(\chan, \all);
});
)
c.map.buss //observe data on the buss
```

```
//remote host
(
c = CtrlBuss.new;
c.addNetwork("10.0.0.4");
//start the server and execute this function as soon as the server is up
c.waitForBoot({
      SynthDef(\test, { |outBus = 0, freq = 440, amp = 0.25, gate = 0|
            var sig = SinOsc.ar(freq) * amp;
            sig = sig * EnvGen.ar(Env.adsr(0.01,0.01,1,0.25), gate, doneAction:2);
            Out.ar(outBus, sig);
      }).add;
      //add a matchdef
      c.addDef(\test_remote, [\k_a], { |self| [self.name, \start].postln }, { |
self| [self.name, \stop].postln },
            [[\pedal, \midiCC_c0n11, \freq, { |vv| vv = vv / 127; 220 +
(vv*660) }]]
      );
      //add a synth to the matchdef
      MatchDef(\test_remote).addSynth(\ka_test, [\test, [\freq, rrand(440,660),
\amp, rrand(0.1,0.25)]]);
});
)
```