INSTITUTE OF SONOLOGY

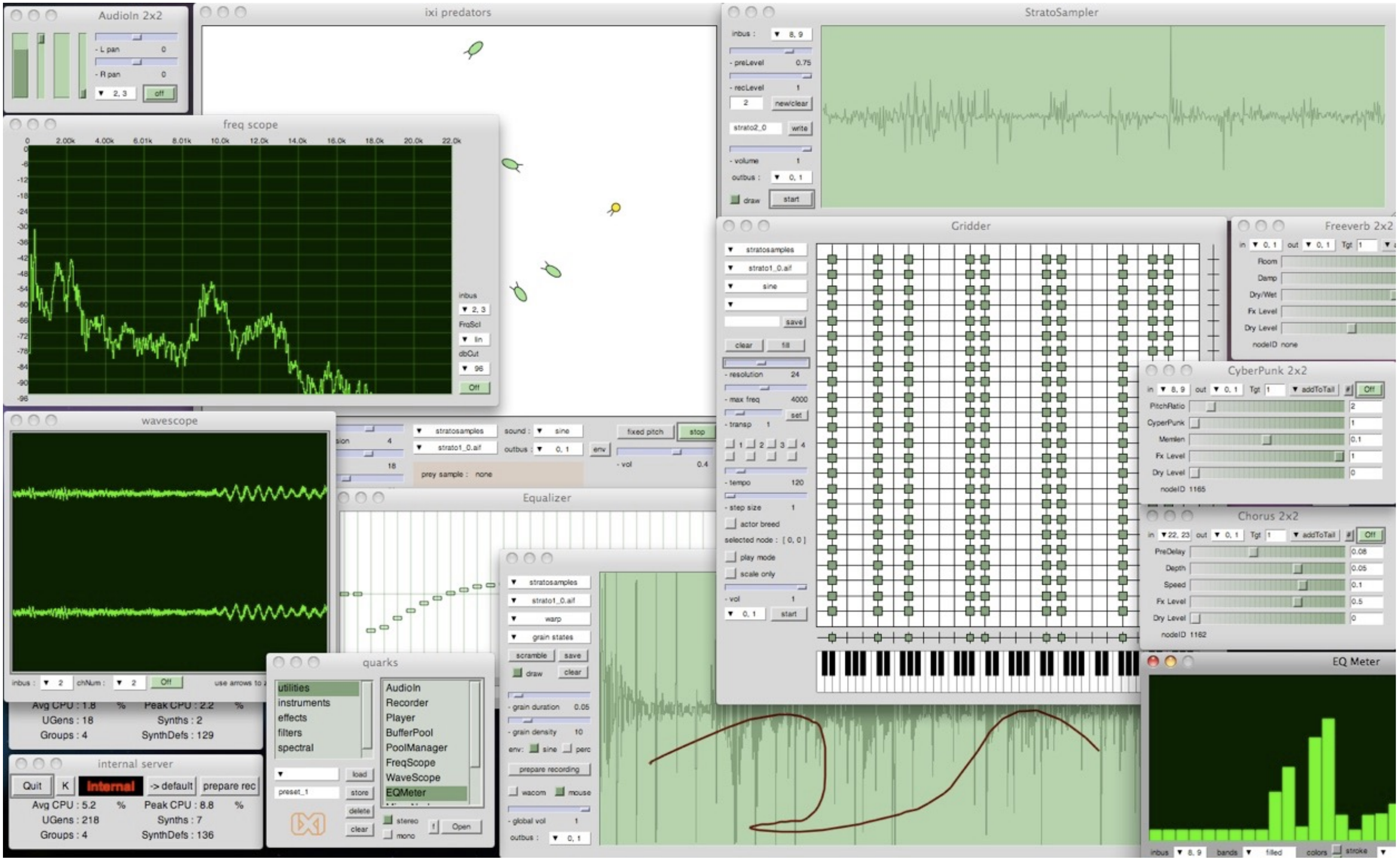
Royal Conservatoire The Hague

Interfaces

Programming and Music
http://www.bjarni-gunnarsson.net

GUI

SuperCollider



Why use a GUI ?

More visually intuitive.

Learning how to use a system is quicker.

Access to parameters is quick.

Multitasking is more straightforward.

Lets one focus on available controls instead of code.

Visible controls and screen space force abstraction.

Code potentially changes more frequently than a GUI.

SC GUI

Originally, SuperCollider was a ***MacOS application***. GUI elements had class names such as SCWindow and SCSlider. These elements looked like ***native*** MacOS GUI elements.

A focus on making ***cross-platform*** GUIs led to the use of more generic class names such as Window and Slider. Methods could be used to choose the type of GUI.

Currently (SC 3.6.3 with new IDE), SuperCollider has migrated to using the ***QT cross-platform framework*** and the use of Macintosh native elements (such as SCWindow) is no longer possible.

Various other features related to creating GUIs are also in transition.

SC GUI

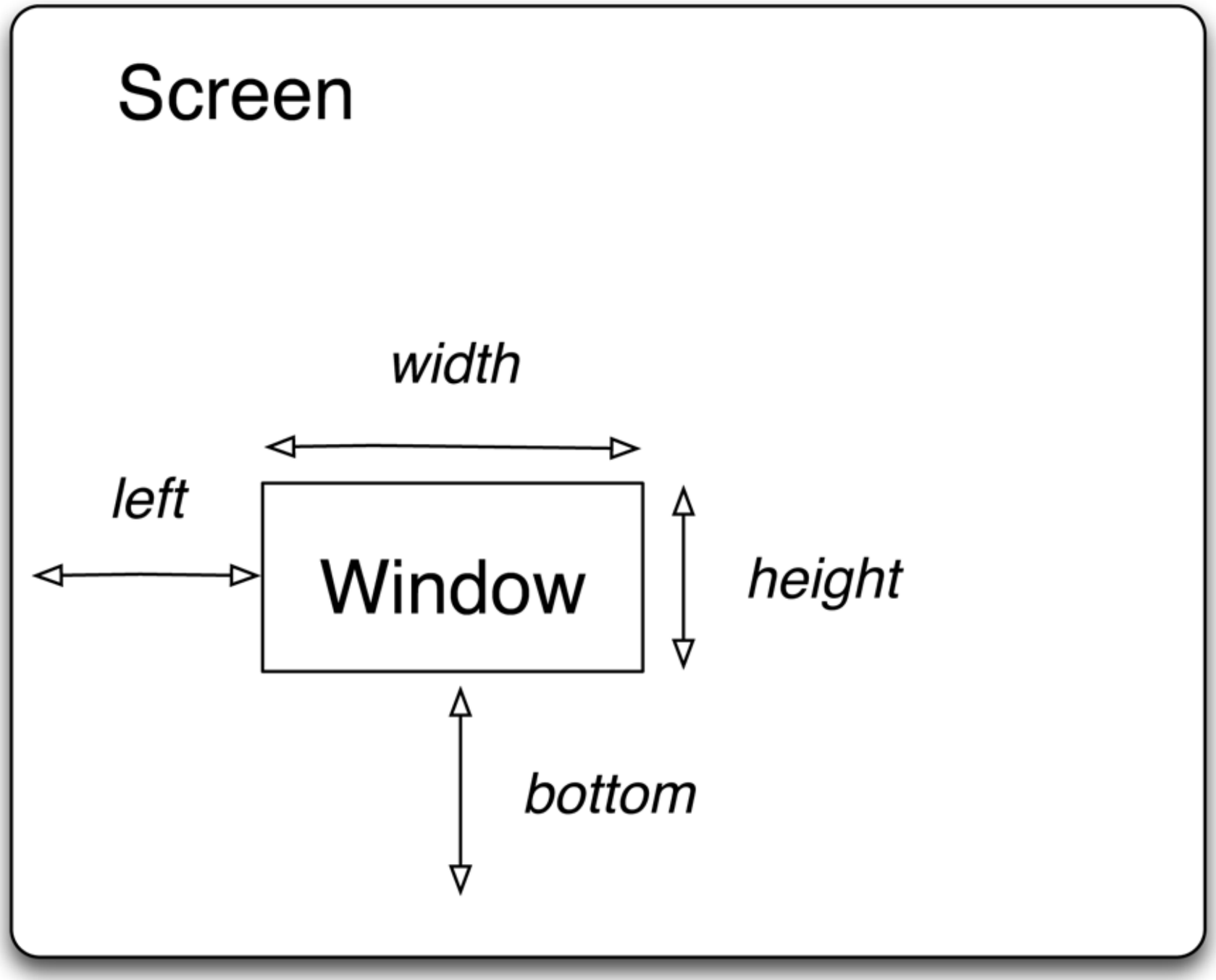
Some information about creating a GUI can be found in the Document Browser, under the category: GUI. This documentation may not be up to date.

In most cases, a user will need to:

* Create a ***window***
* Insert ***items*** such as sliders, buttons, etc.
* Specify ***what*** each of these items does

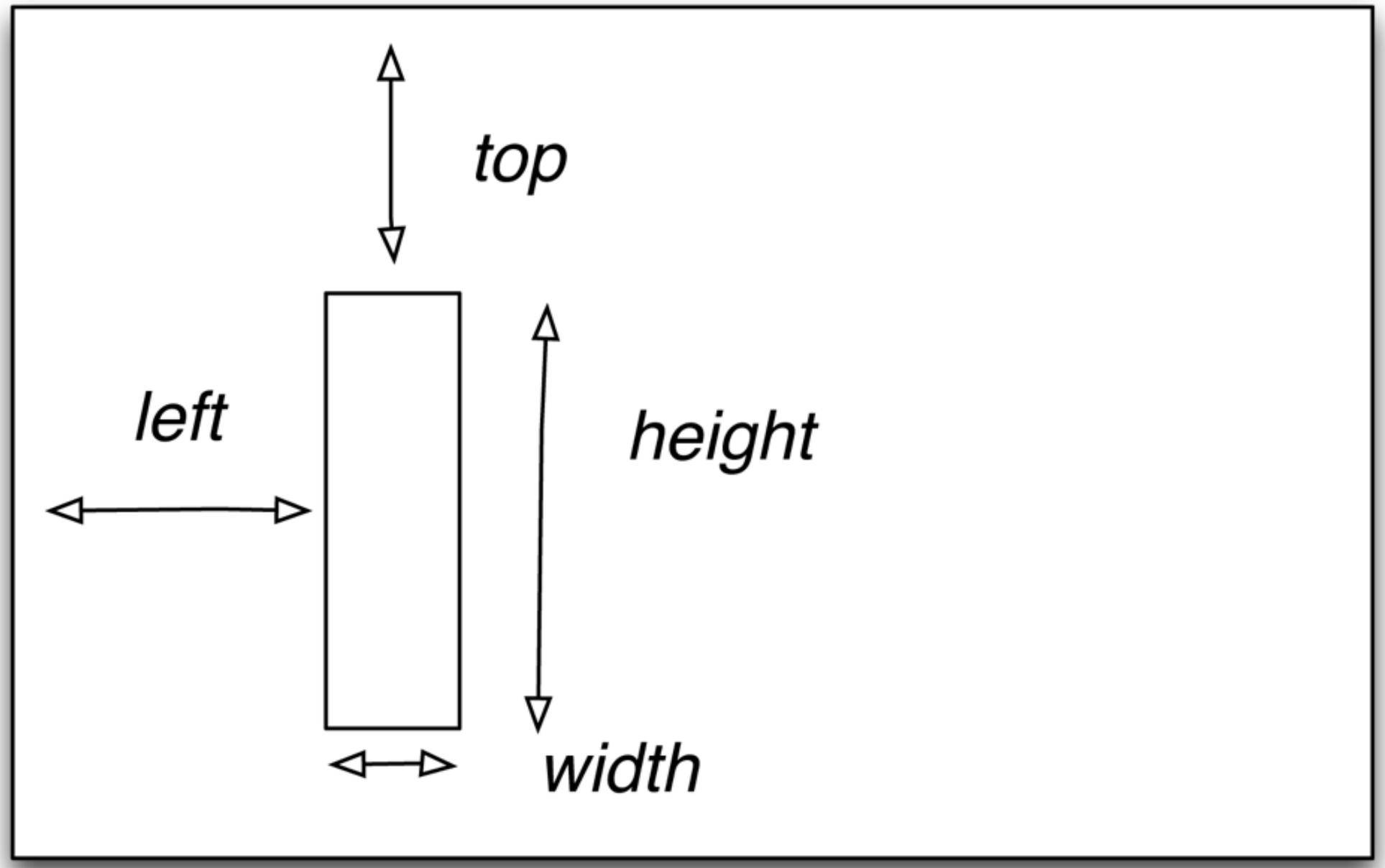
There is also an option to automatically 'decorate' a window which may make things easier.

Position of window in a screen



Rect(left, bottom, width, height)

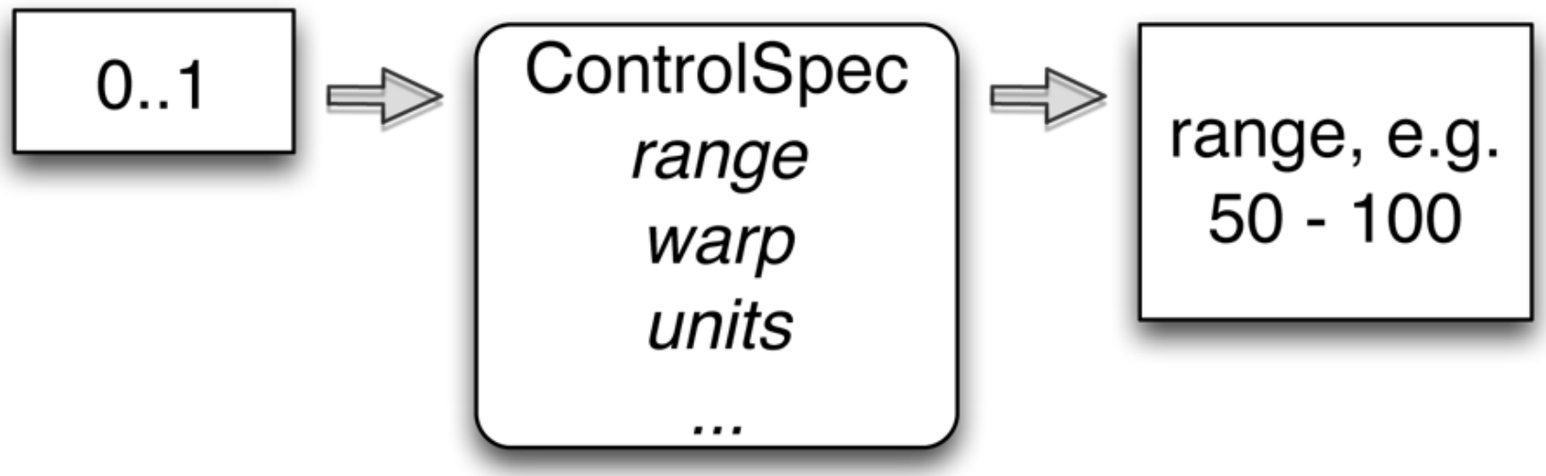
Position of object in a window



Rect(left, top, width, height)

Mapping with Control Spec

Map



Unmap

Basics

Windows and Basic Controls

Creating Windows

```
var win = Window("Title", Rect(left, top, width, height));

win.front; // Display the window
```

Essential Controls

StaticText: Display text (titles, labels, dynamic content)

Button: Interactive control with multiple states

Slider: Continuous value control (0.0 to 1.0)

NumberBox: Display and edit numerical values

Key Pattern

Connect controls with .action:

```
slider.action = {|s| numberBox.value = s.value };
```

Layouts

FlowLayout - Auto-wrapping elements

- Use: win.view.decorator = FlowLayout(bounds, margin, gap)
- Elements flow left-to-right, wrap to next line
- Good for: Multiple similar controls (buttons, knobs)

HLayout - Horizontal rows

- Use: win.layout = HLayout(element1, element2, ...)
- Elements stretch to fill available width
- Use nil for stretchable space

VLayout - Vertical columns

- Use: win.layout = VLayout(element1, element2, ...)
- Elements stretch to fill available height
- Stack controls vertically

Windows and Basic Controls

Controlling Synths

Create synth with gate for proper cleanup

Connect slider to synth parameters via .set

```
Free on close: win.onClose = { synth.free }
```

Value Mapping

Scale slider values (0.0-1.0) to useful ranges:

```
var freq = slider.value.linexp(0, 1, 200, 2000);


.linlin - linear to linear
.linexp - linear to exponential
```

Routines and defer

GUI updates must happen on the AppClock, but audio routines run on different clocks.

Solution: **defer{}**

Wrap GUI updates in defer{} when calling from Routines:

```
Routine({
   10.do {|i|
      defer{ textView.string = i.asString };
      0.5.wait;
}}).play;
```

Animation with UserView, Enable real-time drawing:

```
view.animate_(true);
view.frameRate_(60);
view.drawFunc_({ /* drawing code */ });
```

Any GUI update from a Routine or Task needs defer{}.

Exercises

```
(
NF(\iop, {|freq=78, mul=1.0, add=0.0|
    var noise = LFNoise1.ar(0.001).range(freq, freq + (freq * 0.1));
    var osc = SinOsc.ar([noise, noise * 1.04, noise * 1.02, noise * 1.08],0,0.2);
    var out = DFM1.ar(osc,freq*4,SinOsc.kr(0.01).range(0.92,1.05),1,0,0.005,0.7);
    HPF.ar(out, 40)
}).play;
)

(
NF(\dsc, {|freq = 1080|
    HPF.ar(
        BBandStop.ar(Saw.ar(LFNoise1.ar([19,12]).range(freq,freq*2), 0.2).excess(
        SinOsc.ar( [freq + 6, freq + 4, freq + 2, freq + 8])),
        LFNoise1.ar([12,14,10]).range(100,900),
        SinOsc.ar(20).range(9,11)
    ), 80)
    ;
}).play;
)
```

```
var <>pindex, <>cindex;

initialize {
    if(pindex.isNil, { pindex = 1000 });
    if(cindex.isNil, { cindex = 2000 });
}

clearProcessSlots {
    pindex = 1000;
    (this.pindex - 1000).do{|i| this[this.pindex+i] = nil; }
}

clearOrInit {|clear=true|
    if(clear == true, { this.clearProcessSlots() }, { this.initialize() });
}

transform {|process, index|
    if(index.isNil && pindex.isNil, {
        this.initialize();
    });

    pindex = pindex + 1;
    this[pindex] = \filter -> process;
}

control {|process, index|
    var i = index;

    if(i.isNil, {
        this.initialize();
        cindex = cindex + 1;
        i = cindex;
    });

    this[i] = \pset -> process;
}
```

```
(
NF(\depfm, {|freqMin=5, freqMax=20, mul=20, add=80, rate=0.5, modFreq=2100, index=0.3, amp=0.2|
    var trig, seq, freq;
    trig = Dust.kr(rate);
    seq = Diwhite(freqMin, freqMax, inf).midicps;
    freq = Demand.kr(trig, 0, seq);
    HPF.ar(PMOsc.ar(LFCub.kr([freq, freq/2, freq/3, freq/4], 0, mul, add),
    LFNoise1.ar(0.3).range(modFreq,modFreq*2), index) * amp, 50)
}).play;
)
```

Exercises

Exercise 1: Basic Window

Create a window with a StaticText that displays "Hello GUI".

Make the text centered, size 24, and red color.

Exercise 2: Slider and NumberBox

Create a window with a slider and a number box.

When the slider moves, update the number box with its value.

Exercise 3: EZSlider

Create a window with an EZSlider for frequency.

Use the \freq ControlSpec and set labelWidth to 80.

Exercise 4: Routine with defer

Create a window with a StaticText.

Use a Routine to count from 0 to 10, updating the text every 0.5 seconds.

Remember to use defer{} to update the GUI.

Exercises

Exercise 5: Slider to Synth

Create a window with a slider that controls the frequency of a playing synth.

Use a simple sine wave SynthDef.

Exercise 6: Multi-Parameter Synth Control

Create a window with three EZSliders controlling frequency, amplitude, and filter cutoff of a filtered saw wave synth. Use appropriate ControlSpecs for each parameter.

Exercise 7: Start/Stop Synth Button

Create a window with a button that starts and stops a synth.

The button should have two states: "Start" (green) and "Stop" (red).

Make sure to free the synth when stopping.