project 2

在max/msp中自定义采样器

在max/msp中自定义采样器

INTRO

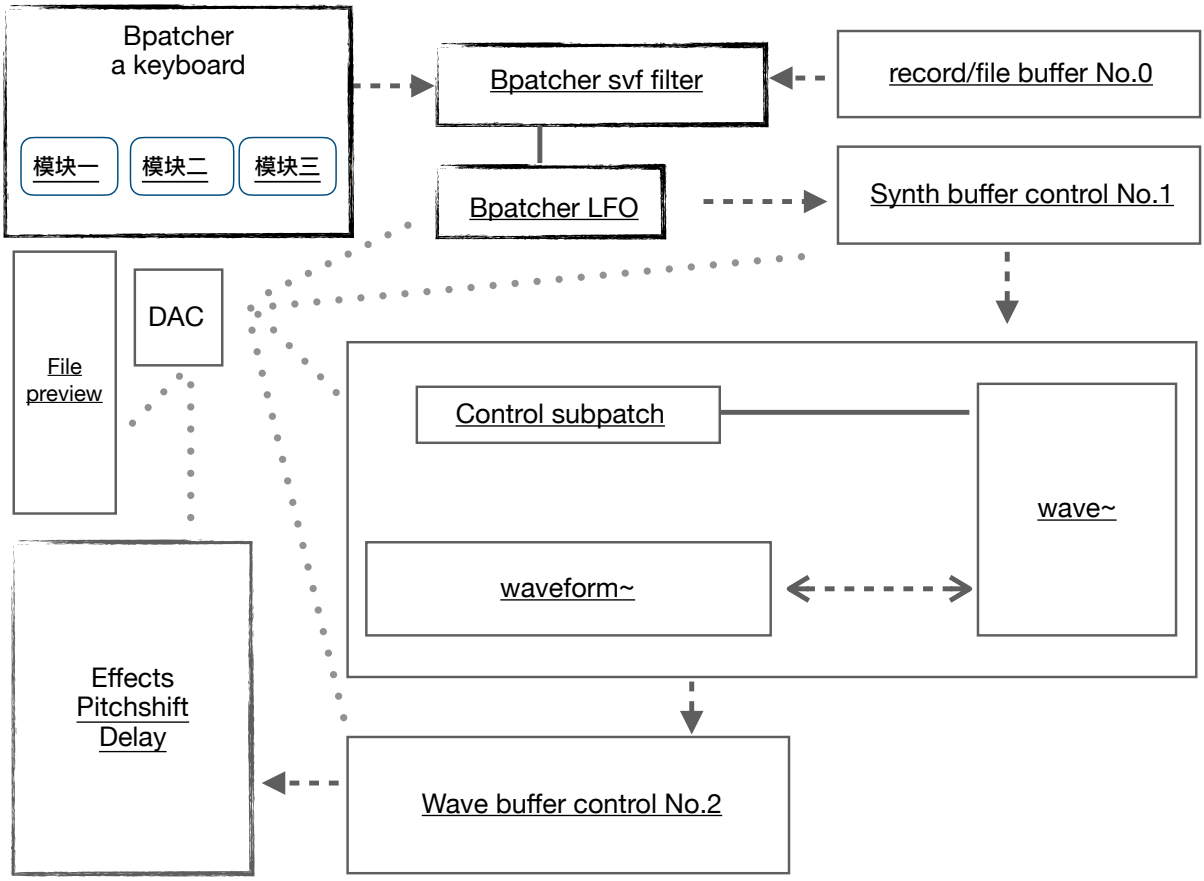
本实验是在max/msp中设计一个简易的自定义采样器，旨在尝试使用个性化的元件和线路处理声音，制作音色。尽管音乐工作者都拥有丰富的处理器/效果器和不计其数的预制效果，但创作的理想状态依然是对材料和参数的完全控制，从原始音源到每种效果的每个参数。自定义设计的最大优点在于随时更新、小巧轻便，完全根据使用者的需求设计需要的线路、剔除多余元件，甚至不受参数上限的控制。在拥有无限可能的同时，又不必苦恼于庞大的软件和海量插件带来的负担。
声音处理按照先后次序，包括基本的滤波、低频振荡器，模拟波表合成效果，写入本地文件。额外的效果器有变调和延时。真正的波表合成器是对真实乐器采样声音的动态调制，而这个采样器尚不具备真实波表的功能，但通过wave~模拟了这种调制方式所产生的音色效果。音源采集不限于录制，也可选用合成声音。制作完成的音色通过变调处理，可模拟音阶的效果。

I. WORKFLOW

一，采集声源。可通过虚拟键盘[a keyboard]模块构建振荡器基本音色，其他支持的声源采集方式包括麦克风外录和读取本地文件。
二，调制处理、写入buffer。合成器基本音色首先进入filter和LFO模块，默认直通。这个模块也是接收其他声源的必要流通线路，为保持简洁准确，只使用一个核心buffer作为声音处理的承载。
三，运用波表方式的处理，制作音色。
四，完成音色并录入最后一个buffer，回放检查后写入文件。
五，可选用效果器预览音阶效果和延时效果。

以下结构图是按照patch的大体结构（非演示模式），保持相应的位置和比例关系而制作的，以最简洁的方式呈现整体架构和信号流通顺序，其中较粗的框架代表这个模块是一个bpatch，虚线箭头代表信号的无限发送目的地，实线代表直接的连接关系。原点虚线专门表示DAC接收无线发送的声音。整个patch仅使用一个[ezdac~]输出音频。

II. OUTLINE



III. STRUCTURES

以下按不同功能的结构块分别说明。

注：在这一max文件中出现的所有subpatch、abstraction，其输入输出口都添加了comment；所有标签化的按钮都添加了hint。

○ subpatch - [a keyboard]

这个patch的结构中包含三个模块：读取电脑键盘；midi信息处理；基本波形合成。

模块一，键盘读取

- **基本逻辑**

用一对**[key][keyup]读取键盘按键**并转换成midi音符号，通过打包形成音符开关信息。键盘按键规则按max内置的keymidi设置。为了方便随时关闭对键盘按键的读取，**对[key]添加了gate控制**。

- **技术实现**

【key线路】

通过**[select]筛选**特定的按键并连接到[funnel]。[funnel]对象默认从0起输出与inlet对应的一个list:

(no. of inlet, given value)。通过**[funnel]的第二个argument设定起始序号偏移量**，即可从所需序号起输出。而这里[funnel]没有接收到对应值，因[select]的输出全部为bang，因此[funnel]输出值均为零。此处仅需要它的output list中的第一个元素，即序号，作为midi音符号信息。
解包后获得相应的midi音符号。**音符号与手动设定的midi音量打包成基本的midi note信息** (pitch, velocity) 传递给[kslider]。

【keyup线路】
原理同key线路，keyup监测键盘按键**抬起状态，作为"音符关"信息的来源**（pitch, 0），此信息可直接传递给[kslider]，但为了让音符信息有可扩展的空间（八度升降），仍**通过解包再打包的形式将(pitch, 0) 传递给[kslider]**。

【八度升降】<GUI>
midi音符八度调节仅需以12的倍数升降即可。**GUI对象[incdec]可用于逐级升降的操作**，基本的连接方式是与数字形成一个循环圈，再从数字端输出。升降后形成的**正负n*12通过加法器赋值给midi pitch**。

模块二，midi 信息/复音键盘

使用[poly]建立一个复音键盘并将音符信息传递给下面的合成器模块。（[a keyboard]原本作为一个独立虚拟键盘，配合自定义的sequencer使用。分为midi mode和osc mode两种。针对此项目--简易采样器，没有用到音序器，于是在这一版中经过重新整理，去掉了一些控制信息，尤其省略了连接外部设备的接口，假定为仅使用电脑键盘作为可弹奏的虚拟midi键盘。初始默认开启osc mode。）

• 基本逻辑

复音形式通过[poly]形成，但音符信息并不直接进入[poly]而是经过[midiformat]格式化为midi event，因为要使用[midiformat]更多的控制码功能。midi输出使用[midiout]并指定了port。在建立复音的线路中再次强制接收同一端口（目前初始默认IAC Driver Bus），确保信息传递的准确对等，同时也留出可选择端口的下拉列表在用户界面中。

• 技术实现

【midi端口】
输入输出的**端口选择均由[midiinfo]配合[umenu]**使用，通过loadbang使文件在初始化时读取当前设备的midi信息。但目前添加了"强制化"指令，使其默认为IAC Driver。在**osc mode中，要确保midi信息不能从mac系统的内置合成器AU DLS端口输出声音**。在midi mode中，则要确保midi声音通过AU DLS输出midi音色。针对不同模式触发相应的port。
<trouble shoot> 通过无线发送和接收选择模式的序号，在输出端口处触发相应的信息指令切换port时，出现了[midiout]没有即时更新信息的问题。尽管与它连接的port指令已经接收到信息，并且查看event probe时，显示信息已经更新，但[midiout]仍停留在上一次的选项上。尝试再次触发port指

令，[midiout]便随之更新。因此在midi设备下拉列表的输出端添加了一个bang使其在输出选项信息后随即输出一个bang，问题仍未解决。由此考虑到是系统延迟问题，于是为这个bang作了200ms的延迟，之后操作选择midi模式或osc模式，输出端能够顺利更新。问题解决。

【虚拟键盘与音符信息】

模块一中形成的基本音符信息进入kslider，这一对象主要作为可视化的GUI使用，其显示模式设定为polyphonic。**midi信息通过[midiformat]形成，并传递给midiout输出**。[midiformat]整合了程序改变、控制改变等重要的控制信息，此处隐藏了部分内容但**保留了一个关键控制码123** (midievent 176 123 127)，并将此操作显示在presentation中，作为重置选项，以防出现意外。在**触发123全部音符关时，同时触发键盘的flush指令**，以清除鼠标点按留下的未抬起按键。

【复音线路】

针对osc mode，**[notein]接收的音符信息通过[poly]收集成为复音信息**，此处略为简化，只做了五个复音。开启steal模式。[poly]输出的复音信息主要在于多了一个序号，五个复音就会出现1～5序号开头的的音符信息list。而**复音音符信息的关键传递由下一个模块的[spray]处理**。

模块三，基本波形合成

midi音符转换成为其**音高对应的频率并通过振荡器形成基本波**，成为合成器音色音源。

- **基本逻辑**

从[spray]获得的音符信息，进入一个单独的合成模块，通过转换频率给到振荡器，形成基本音色。

- **技术实现**

【获取音高】

带有序号的音符信息(list)从上一模块中被送出，由[spray]接收，与复音对象相应有五个输出口。[spray]的默认模式（listmode=0）为将input list拆包、输出单个值，但在此处**开启listmode**，**[spray]按照序号依次送出 (pitch,velocity) list**，它将成为subpatch - **[basicwave]的输入信息**。在[spray]下添加的[unpack]仅为了单独获取音高并将其显示出来。

【生成基本波及示波】abstraction - [basicwave]

[basicwave]的三个inlet分别接收音符信息、scope开关、[radiogroup]波形选择。

在这一patch中，midi音符信息被拆包，**pitch通过[mtof]转换为频率，velocity通过[scale]进行数学映射，使其转换为0～0.9的振幅值**。

频率：频率值输入给基本波振荡器，**音频信号经过[selector~]（用户端通过[radiogroup]筛选）**最终的信号经过一个乘法器后输出。

振幅：振幅值经过**[line~]处理，**对于zero to non-zero或non-zero to zero，都有一个**20ms的渐变，防止click**。

[basicwave]从两个outlets分别输出音频信号，其中一个用于连接[scope~]显示波形。但有时使用者可能不需要看到每个波形，或不想被显示干扰，因此**添加[gate~]控制这一端的输出**。
从[basicwave]送出的信号**整合到一个乘法器中经过一次pre-attenuate，之后再连接到作为GUI的[gain~]**。整合信号另有一路进入了略大一点的[scope~]中，用于显示合成后的波形，同样受到scope开关的控制。
在用户选择osc mode时，默认开启音频开关和scope，midi mode时这些将全部关闭。

【送出基本音源】
尽管在制作和测试时，合成出来的声音是连接到[ezdac]的，但在测试完成后就断开连接了。这一图标仅作为音频开关和图示。真正的**信号通过[send~]送出，成为一个全局关键字。[a keyboard]在“简易采样器”中**是一个集成工具，**仅送出信号**，不播放声音。声音的控制需要从它的外部 —— 即采样器这一层进行控制。

○ subpatch - [filter_sampler]

这个patch以[svf~]为核心。普通模式已经过整理，只留GUI对象，其他打包在[p svf_filter]中。

• 基本逻辑

[svf~]由于结合了四种常用的滤波形式并且可以同时使用，较为方便高效，操作也很简单。

• 技术实现

信号首先经过[gate~]，在用户界面中控制信号的输出通路：直通或者进入[svf~]的信号输入端口。**用户界面中用[live.slider]设置截频，使用对数型显示，exponent level 5。resonance显示为百分数，实际取值为0～1**。最终的整合信号经过一个乘法器衰减，输出。
在采样器patch中，**这一模块的输入信号有两路可选，一是来自虚拟键盘**，接收的关键字“osc_sound_little_keyboard” 来自[a keyboard]。**另一路来自0号buffer组**。两路信号由[selector~]接收。用户界面则使用[umenu]进行选择。
Filter和LFO模块是信号必经线路，默认直通，音量默认开。

○ subpatch - [LFO_sampler]

结构设计类似svf filter，普通模式已经过整理，只留GUI对象，其他打包。

• 基本逻辑

LFO由四个基本振荡器直接生成，用户界面的频率操作使用[live.dial]，范围限定在LFO通常的**低频区域0～30，使用对数型显示，exponent level 2**。通过[selector~]进行选择，用户界面使用[umenu]。

• 技术实现

在[selector~]之后，对振荡器生成的波形进行了偏移，使其作为信号的乘数保持0以上。LFO amount在用户界面显示为百分数，实际取值0～1。通过连接低频振动器的乘法器，直接调节amount会影响整体音量，因此**需要数值补偿，使其不对原音量产生影响**。补偿方式是**通过补数加回去**。amount值与1之间的差额就是补偿值。由此，调节amount导致的减少值在下一条线路中会被加回去。最终作为调制波连接在信号的乘法器上，输出。

○ Buffer group - record/synth/wave

[buffer~] [record~] [play~]相互配合共同构成一个组，在本采样器中共有三个这样的套组，结构一致，依据功能需求的不同略有微调。

• 基本逻辑

0号为可选的record buffer组，针对外录和读取本地文件的情况；**1号为核心组synth buffer，0号组声音最终仍进入核心组进行处理和录制**；2号为波表声音制作后的完成版本。录音按钮可自动关闭，当buffer时长被重置后，自动关闭的时间也随之更新。

• 技术实现

【record buffer】

如使用录制或打开本地文件，则首先在record buffer组处理。**0组的声音由调制组直通输出**，调制组默认直通。这一组为必经线路，即便不做任何处理，也**需要从这一组录制并写入核心buffer**。

0组与其他组的不同主要是多了ADC外录和本地文件读取。adc信号进入[record~]之前通过乘法器(作为一个“前置放大器”) 进行增益，振幅范围0～20。连接在乘法器后的[meter~]作为用户界面的提醒。

打开本地文件的方式：可以**拖拽文件，通过[textedit]读取路径。**文件路径传输给[buffer~]的**read指令进行读取**；也可以**使用replace指令打开系统对话框**。

【synth buffer】

如使用虚拟键盘创建**合成声音，信号直接从调制模块输出，通过录制，写入buffer-mysynth，即进入核心buffer。**这一组功能比较简单，但是一个核心线路。

【wave buffer】

2号是完成组，比其他组多了写入按钮。主要用来做最后的检查以及写入文件，保存到本地。

【自动关闭录音】

常规设计思路是按下录音toggle的同时启动delay。时间到后delay输出bang返回到toggle即关闭。但如果用户提前手动关闭toggle，就会由于delay周期仍在继续，导致出现额外的bang使toggle被意外开启的情况。解决办法：可以增加一个stop指令给delay，线路略繁琐，也需增加select等对象。这里尝试了另一种逻辑。**利用[record~]的sync out输出监测。**这一输出口通过连接[edge~]可以很方便

的监控信号变化。**一旦录音开始，sync out信号从zero变化到non-zero，[edge~]左侧的outlet输出bang，触发delay的开始。delay到时间后输出的bang不用来控制录音开关，而是给到[edge~]右侧的bang，这个bang触发一个0给到开关，便能使其关闭。**如果用户手动提前关闭录音，sync out将变化为零。这时，尽管delay仍未停止，但不必理会，因为时间到了后输出的bang只会令已经处于关闭状态了的开关再接受一次0，对实际状态不产生任何影响。如果在极短时间内，用户再次开启了录音开关，这将直接更新[delay]，开始新一轮延迟周期。

【播放控制】
三个buffer组的播放控制是完全一致的，使用[play~]的基本控制指令，**用户界面全部使用[textbutton]**。其中**start, stop所连接的按钮使用button模式**，输出bang直接触发信息框指令。**pause和resume两个指令通过一个toggle模式的按钮完成**，toggle模式的[textbutton]输出1或0，所以使用[select]对象筛选。默认初试值1，用户按下按钮后，便会输出0，pause指令会被触发。
如果用户按了一次暂停键，之后又点击stop按钮停止了音频回放，这会导致暂停键仍在“standby”。下一次点开播放的时候，这个按钮就有问题了。因此，**为停止键增加了一个指令，使其触发stop的同时也将暂停键恢复为默认状态。**

○ wave control

波表音色制作的核心区。

· 基本逻辑

[wave~]与[waveform~]搭配，通过**读取同一buffer名称保持内容一致**；通过无线**发送与接收起始与结尾点信息，保持截取位置一致**。[wave~]的基本作用在于循环读取buffer中的某个段落，而对截取位置的控制是通过在0～1间循环往复的信号对输入信号进行样本扫描。**信号读取使用锯齿波和余弦波**控制，能够形成单向读取和来回读取两种效果，用户界面中使用下拉列表进行选择。根据buffer时长计算并**预设原速播放所需的频率作为初始值**。

· 技术实现

【[wave~]与[waveform~]的关联】
[waveform~]初始设定为读取buffer-mysynth，因此在1号**核心buffer组录制时，波形即已随之更新**显示。**初始默认选中整个段落**，方便回放全部buffer内容，这与[wavecontrol]的默认设定也有关。
[waveform~]中间的两个inlets可设置循环选段的起点和终点，其数据的改变将直接呈现在波形的高亮选择上，而波形的改变信息会直接从下方的两个outlets输出，由此可以形成一个数据循环。**上方inlets可接收set指令，设定起末时间点**。所以**下方输出直接给到set，成为set指令的变量**。与此同时，从outlets输出的一对**时间点被无线发送至[wave~]的table location输入端口**，由此**形成相互关联**。因此不需要面向[wave~]的操作，完全通过GUI对象[waveform~]进行截取操作。

【用户界面波形操作及显示信息】

波形**右侧的slider用于调整纵向上的zoom in/out。**[waveform~]的**vertical zoom这一参数**值越大，波形越缩小，这在视觉操作上和人的一般感觉是反过来的，因此**通过scale进行了反向数学映射。**演示模式中，[waveform~]上方为操作按钮，下方为信息显示和音量调整。**采样率信息和总时长来自[info~]，**通过读取buffer名称mysynth，[info~]可输出相应的信息。**触发这一输出的指令来自上一个模块中的核心buffer组。**利用连接在[record~]sync out端口的[edge~]，**录制结束时，[edge~]右侧outlet输出的bang将触发[info~]。**

【 subpatch - wave control 锯齿波和余弦波控制】

[phasor~]能够生成幅值在0～1间的锯齿波，常用作精确的样本读取工具。线性上升的斜线和干脆利索的回落使它成为wave控制的第一选择。由于**波形是线性上升的，所以可以匀速、单向的读取样本。**频率为正时，锯齿波正向前进，因此是顺序读取样本。反之，频率为负时，就可以逆向读取。因此，[phasor~]可实现**正向或逆向的单向样本读取。**[cycle~]基本逻辑与锯齿波类似，但因其波形为**余弦曲线，对样本的扫描不是线性的，因此声音会形成动态的变化感。**

实际工作中可能需要一个读取“基准”，因此为控制参数设定了默认值，也就是**phasor的默认频率。**这一频率**取决于buffer的长度。**比如mysynth默认为5秒，因此**使用一个完整周期波原速读取的话，这一周期波的cycle长度就需要5秒，则频率为1/5秒。**当mysynth的buffer**时长被重置时**，新的时长将**通过无线发送，在[expr]中重新计算。**这一频率计算的逻辑同样适用于余弦波，虽然在“back and forth”这一模式下，并没有严格意义上的“原速播放”，基于曲线的变化，声音是被实时拉伸的。但仍然需要一个相对的“参考基准”，使其在初始播放时，完整回放的时长与原时长等同。由于cosine在一个周期内将声音样本**来回各读取了一遍**，而且声音会发生变形，为了“听清楚”这一模式下的原速回放，cosine的频率是**将两个周期作为一个周期来设定，其中半个周期为一遍完整回放**，所以频率为phasor的四分之一。

以上**“原速播放”默认基于一个完整的buffer**，因此在段落选择上也设定了默认选中整个buffer (在[waveform~]端设定)。

控制波的振幅意味着对选定样本段落进行读取的范围（0～1），因此用百分比来表示逻辑更清晰。

控制端的对象基本都打包在subpatch中，仅将GUI对象留在外面。

【图示】

波表的音频信息从[wave~]的outlet中输出，这一音频信号经过音量调整后传输给[ezdac~]。这一输出同时也连接到**两个[spectroscope~]，分别显示彩色声谱图和频谱图。**在“back and forth”模式下，声音被拉伸的过程在声谱图上较为明显。**这一输出也连接到了[snapshot~]，以2ms的间隔读取样本值，通过[multislider]显示为滚动的图示，主要作为音频回放时的视觉提示。**

○ effects - pitch shift

音色制作完成后，采样器的工作已经结束。这里额外添加了一点效果器，用于试听已经制作完成的音频。这一版块的核心对象为[groove~]，但并未将其作为looper编辑器使用，而是**利用它读取buffer以及使用time stretch和pitch shift**并输出音频信号。timestretch属性默认开启。[groove~]除基本的播放控制外，不设其他选项。

变调的目的是**模仿一个简单的自然音阶**，试听已完成音色的音阶效果。以原声作为音阶第一音，用户通过电脑键盘上的**数字键，模拟简谱的do,re,mi…，数字1作为do，即未变调的原声**。数字键盘的识别与变调计算打包在[p pitch_control]中。

[p pitch_control] - 电脑键盘的**数字键通过[key]识别，之后经由[select]筛选数字键的ASCII码(49~57)**。筛选后输出的bang将触发[funnel]，**使用[funnel]的序号输出功能**，将序号偏移量设置为49，从而输出每个数字键的ASCII码。经过减法转化为0～8的九个数字。**按照12平均律的频率比，计算每个音符(钢琴白键)的频率关系**。由于并不输出每个半音，所以不能统一计算，只能**按照音程关系为每个音符单独计算pitchshift factor**。

○ effects - delay

delay模块使用bpatch - [delayline_sampler]，普通模式已经过整理，逻辑线路打包在[p delayline]中。

• 基本逻辑

进入[p delayline]的输入信号被分成三路，一个直通，另两个用作delayline的干湿信号。

除了延时线路，信号输出前还会经过一个abstraction - [cosDryWetBal]，用于干湿信号mixing。

用户界面可选择bypass；干湿信号的控制使用一个简单的滑轨，添加了快捷恢复按钮；clear按钮用于清除feedback。

• 技术实现

【延迟反馈】

延迟的核心对象为[tapin~][tapout~]，针对短小的音色，2000ms的延迟容量能够适用大部分情形。使用“**单个tap反馈**”的基本方式，**反馈线路的增益经过了补偿和smoothing**。补偿的基本思路是，反馈信号实时变化，通过计算其均方根值，比较与目标音量的差异，将这个**差异值加回给原信号rms值，其计算结果不断趋近目标值**，即用户设定的反馈量。将这一控制信号经过smooth后传递给反馈循环中的乘法器，作为增益控制。feedback level的最大值设定为0dB，理论上反馈循环的振幅可始终在1以下。

【清除残留信号】

高反馈情况下，如feedback达到最高值0dB，延迟信号会有大量微弱的尾部反馈长时间停留在音频系统中，因此添加了自动清除的线路。通过**对均方根值进行布尔判断，在满足条件时自动触发延时器的clear指令**。经过测试，0.0001是比较稳妥的阈值。为防止信号不稳定而反复bang的情况，**使用[onebang]进行触发**。通过select 1，输出bang。**当信号不再小于0.0001，布尔值变为0，从而触发[select]从右侧outlet输出bang并重置[onebang]**以便为下一次输出bang做准备。

湿信号的输出经过[onepole~]进行低通。

【控制端】

低通cutoff frequency的范围是全频段(20~20k)，使用对数型显示，exponent level 5；feedback范围设定为 -60~0dB，同样使用对数型，但这里为负数计算，exponent level 1/5。

【干湿信号混合 - [cosDryWetBal]】

*** 参考了Andy Farwell在《设计声音》中使用PureData设计的一个线路。**

干湿信号的混合使用**余弦值控制，能够避免线性控制会出现的音量损失**，均衡在中间的时候，可以获得3dB的音量增长，这是两个声音叠加本应带来的声能加倍。

[cos~]的输入端接收0～1之间的phase值，1等同于一整个周期，因此这一输入值意味着一个周期2π的占比。如输入0.5时，相当于计算π的余弦函数值。

在这个mixer中，均衡在极左时（控制值0），需要干信号获得完整振幅，而湿信号降为零；极右时，二者相反；在中间（控制值0.5）时，需要二者均衡。cosine0等于1，而在**-π/4 ～π/4区间，刚好获得围绕y轴对称的半个周期曲线**，可用于控制干湿信号。

0 ～π/4为递减区间（控制干信号），与此同时-π/4 - 0为递增区间（控制湿信号）。（控制值0，干信号振幅100%；控制值0.25，干信号振幅0；控制值-0.25，湿信号振幅0；控制值0，湿信号振幅100%）

用户端输入的0～1之间的控制值，首先经过乘法器缩减为0～0.25，这一区间可以直接作为干信号的控制值。缩减后的值偏移 -0.25 ，获得-0.25～0的区间，这一区间作为湿信号的控制值。当用户输入为0.5时，首先被缩减为0.125（1/8周期，即π/4），此时，干信号控制值cos(π/4) 和湿信号控制值cos(-π/4)，均为0.707左右。由此实现了比较理想的平衡控制。

○ preview and listen

这一版块是独立的，在导入音源文件之前，可能会预听一些潜在有用的音频。逻辑上属于音源的步骤，所以在演示模式中，和0号、1号buffer组处于同一层。

• 基本逻辑

声音的播放通过**[sfplay~]直接读取文件**，与buffer无关。连接了[playbar]以方便回放操作。[sfplay~]的文件读取**配合[umenu]以方便加载文件夹**。

• 技术实现

【打开文件】

直接使用**[sfplay~]的read指令**打开系统对话框选择单个文件。

【加载文件夹】

文件路径的读取关键利用**[umenu]的prefix的指令**，prefix的参数如果是一个有效的文件夹路径，在**开启autopopulate**属性的前提下，umenu可以自动清理现有列表并将文件夹中的所有文件填充进去。在这里，umenu的file type属性已设置为“Mp3 WAVE M4a CAF AIFF”，因此会过滤其他格式的文件。用户通过下拉列表选择音频文件后，**完整的文件路径将传递给[sfplay~] open指令，成为其参数变量**，从而读取相应的文件。
接收文件夹信息的是GUI对象[textedit], 其右侧outlet监测变化，发生变化时将输出bang，这个bang返回给输入端，触发文本内容的输出。通过**[route]去掉输出标签“text”**，即可成为prefix指令的参数。
利用[textedit]输入端bang的功能，**添加了“reload”按钮，以方便重载**。

【文件夹读取层级】
[umenu]的**depth指令可以实现向下或向上读取文件夹层级**，上下逐级变化的情况非常适合**用[incdec]来完成**。在用户界面使用的则是[textbutton]，在这个项目中绝大部分按钮都是应用[textbutton]。用>>和<<符号作为标签，按钮输出的bang用来**触发[incdec]的inc指令和dec指令**。**[incdec]与作为depth指令参数的数字框相连**，便可以使用户**手动读取更深一层文件夹或往回读一层文件夹**。但decrement的指令是没有限制的，即，如果用户在数字已达到0后，仍一直触发dec指令，[incdec]会继续输出小于0的数字。尽管**与depth相连的数字框限制了最小值0，但[incdec]中暂存的数字却可以是远小于0的负数，因此设定了一个boolean控制**，如果[incdec]中暂存的数字**小于等于0，将触发set指令，将其重置为0**。如果不做这一重置，在用户往回按了许多下的情况中，比如[incdec]内存数字达到了-10，用户这时意识到要按相反方向的按钮，那么就不得不按十次才能回到非负状态。因此，重置为零可以灵活解决这一问题。
[umenu]中的文件个数从其右侧outlet中输出，这一信息也呈现在用户界面中。

SUMMARY

msp对象大多是高度集成的，可以非常方便的设计音频处理工具以及添加足够丰富的功能，即便不使用任何第三方库，仅内置对象就可以满足大多数需求。在设计上，更重要的在于线路排布、信号流通等结构关系。在这个采样器中，我也更多地关注了细节处的逻辑构建，尽可能使其成为易用的、逻辑顺畅的工具。
目前，这个小工具能够满足简单的音色制作，但仍有很大改进空间。在音源方面，基本波形声音的构建比较单一；对真实声音的采样，还需要更多的预处理和编辑功能；波表合成方式也可以做更多扩展。作为一个自定义工具，根据实际需求随时更新迭代是它的最大优势，因此这个工具可以成为比较理想的工作助手。