Working with WFSCollider

Arthur Sauer and Wouter Snoei

The Game Of Life Foundation

Version 2.2.4

Working with WFSCollider

Table of contents

Starting up

Installing WFSCollider on your computer

To run WFSCollider you need a Mac with at least OSX 10.6 and an intel processor.
For running on linux and older versions of OSX you will need to install the libraries separately on an existing SuperCollider install. Most probably it does not run on Windows (it has never been tested, and saving files might be problematic).

Go to the website of The Game of Life: www.gameoflife.nl
In the menu, click on About > Working with our system.
Under Software downloads you will find the link (currently: http://sourceforge.net/projects/wfscollider/files/).

Mac:

Download the latest version of WFSCollider v2 (as a packet: .dmg)
Double-click on the 'WFSCollider x.xxx.dmg' file.
In the window that opens, drag WFSCollider.app to the Applications folder (or any other folder you like better).
Do the same with the Docs folder.

Note for **OSX 10.8** and **10.9** users: when opening WFSCollider for the first time you might get a warning about not being able to open the program because it is from an unknown developer. This is the so-called Gatekeeper speaking. To be able to open WFSCollider in this case you can open it by clicking on the app icon with the ctrl key pressed (or with the second mouse button), and choose "open" from the contextual menu that appears. This also results in a warning, but this time you can choose "ok" to open the app anyway. It only needs to be done once; after that WFSCollider can just be opened the normal way. Please also note on OSX 10.8/10.9 that the app should really be in the Applications folder. If put elsewhere it might also trigger the Gate keeper into quarantining the app, which effectively disables it forever on your machine (there are ways to get it out of quarantine to be found via Google, but that is not trivial).

Linux and windows (and mac if you want to run WFSCollider within SuperCollider)

install SuperCollider: http://supercollider.sourceforge.net/
when SuperCollider is up and running, install the following Quarks (via: Quarks.gui):
- MathLib
- NetLib
- PopUpTreeMenu
- VectorSpace
- wslib

also install the sc3 plugins: http://sc3-plugins.sourceforge.net/
Download and install the latest version of the libraries:
- https://github.com/GameOfLife/Unit-Lib
- https://github.com/GameOfLife/WFSCollider-Class-Library

after installing, to run WFSCollider, run the following line in SuperCollider:

```
WFSLib.startup;
```

Opening WFSCollider

Double-click on WFSCollider.app.

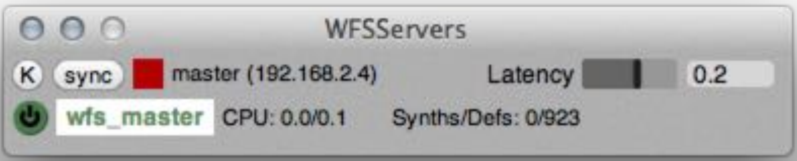
In the post window you should see something like:

```
init_OSC
empty
compiling class library...
	NumPrimitives = 769
	compiling dir: '/Applications/WFSCollider-dev/WFSCollider.app/Contents/Resources/SCClassLibrary'
	pass 1 done
	numentries = 1652997 / 37770634 = 0.044
	9181 method selectors, 4114 classes
	method table size 32860936 bytes, big table size 302165072
	Number of Symbols 22087
	Byte Code Size 957915
	compiled 901 files in 0.68 seconds

Info: 8 methods are currently overwritten by extensions. To see which, execute:
MethodOverride.printAll

compile done
Help tree read from cache in 0.01410594 seconds
Class tree inited in 0.04 seconds
[ nil, [ 58000 ], 8 ]
booting 57999
Found 0 LADSPA plugins
Number of Devices: 4
   0 : "Built-in Microph"
   1 : "Built-in Output"

"Built-in Microph" Input Device
   Streams: 1
      0  channels 2

"Built-in Output" Output Device
   Streams: 1
      0  channels 2

SC_AudioDriver: sample rate = 44100.000000, driver's block size = 512
SuperCollider 3 server ready.
Receiving notification messages from server wfs_master
Shared memory server interface initialized
```

Of course the output is also dependent on the audio interfaces you have connected to your computer, and the setup in AudioMidiSetup.

To check your settings in 'Audio Midi Setup', open 'Audio Midi Setup' in the folder /Apllications/Utilities/, and have a look in the Audio Devices window.

In WFSCollider you see a window called WFSServers. In the bottom left you should see a green button and next to it the text wfs_master.



master (192.168.2.4) 0.2
wfs_master CPU: 0.0/0.1 Synths/Defs: 0/923

If so, you are now ready to work with the program.

Soundfiles can be of almost any format except for mp3 and FLAC. See here for a complete list: http://www.mega-nerd.com/libsndfile/ . The sample rate of the system is 44.1KHz, so soundfiles that with that sample rate will sound the best (although it will playback sound files of any sample rate correctly).

About WFSCollider

WFSCollider is the driving software of the Wave Field Synthesis system of the Game Of Life Foundation (http://www.gameoflife.nl). With WFSCollider it is possible to create sound sources on the system, defining their location in the two dimensional field. Sources can be moving in real-time along a trajectory or according to a function, sit steady in the room or be manipulated live. The sound that plays on these sources can be basic sound files, but WFSCollider also offers numerous possibilities for processing and synthesizing sound. The sounds or sound processes can be placed on a timeline, and everything can be saved into project files.

WFSCollider is based on SuperCollider, an open source music-oriented programming language. While the WFSCollider interface is fully graphical, it is also fully scriptable via SuperCollider code. There are many places in the program where you might find SuperCollider functions. However, it is not required to know the SuperCollider language, or even anything about programming in general, to be able to operate WFSCollider.

Apart from driving the Game Of Life system, WFSCollider can also be used to prepare your projects at home, or even to compose for headphone, stereo or quadraphonic setups using the same features and interface. This is what we call the “offline” mode, the default mode in which WFSCollider will startup.

It is also possible to use this software to drive basically any WFS setup. It can be configured for many situations, simply via the preference panel. WFSCollider can drive both single array setups and full setups surrounding the audience. Also for this, no SuperCollider coding skills are required.

The development team of WFSCollider is always open to suggestions. If you are a programmer yourself, you are also welcome to send us pull requests for the libraries involved. Bug reports can be made here:
https://github.com/GameOfLife/WFSCollider/issues

Copyrights

The software is distributed as open source, at no charge. This means you may use it and adapt it to your liking, but not sell or resell it or parts of it yourself. The musical compositions you create with the program do not fall under this license; for those normal copyright laws apply.

Unit and WFS Libraries

WFSCollider is based on the Unit Library. This is a SuperCollider library that enables timeline based editing and patching of processes (“Units”) in a graphical way. The library was developed together with the WFS Library, especially for this project, but it is more generic; it can also be used separately without the WFS capabilities. Also, there is the VBAP library (currently part of the WFS Library), which is used to drive multi-speaker setups via the VBAP technique. Just like the WFS Library, it uses the basic infra-structure, framework and GUI interface of the Unit Library.

As a WFSCollider user it may be important to know about the distinction between these two libraries, even when you are not programming source code. In the graphic interface you will sometimes find names of objects that start with a capital ‘U’ (UChain, UGlobalEQ, Udef etc..). This means that these objects belong to the Unit Library. Also, there are objects of which the name starts with ‘WFS’ (WFSServers, WFSPath etc..). Those are specific to the WFS library.

The reason that we show these in the graphic interface is to maintain clarity about the exact names used in the programming language, so that scripting for the system blends in seamlessly with working in the GUI (graphical interface). This also goes for counting values; SuperCollider (as many programming languages) always starts counting at 0 (zero), not 1. Therefore most of the settings and values that are in whole numbers in the interface also start at zero (for example the number of an event in the score editor, or the index of a specific loudspeaker on the WFS system), even though most humans start counting at 1...

Basic elements

In WFSCollider we define a number of basic objects and concepts.

Score (UScore)

A Score holds a timeline, along which Events can be placed. Playing a Score will start and stop these events at the right moment. The Score can be saved and read as a file (.uscore extension). The system can open and play multiple scores at the same time, and there are also ways to manipulate the timeline while playing. Scores have an analogy to what is sometimes called "arrangement" in other DAW software.

Note that Score pause in WFSCollider means something else than in most DAW software. When setting a Score to pause, only the timeline is paused. Any events that were already playing will keep playing. If you really want to stop all sound, use the stop button instead.

Event (UEvent)

Events are objects to be activated by a score. They can also be activated individually (via supercollider code, via OSC messages or via the 'power' button in the editor). There are currently three types of events: **Chains** (UChain), **Markers** (UMarker) and ... **Scores** (UScore). Yes, the Score itself is also an Event, and it can be placed on the timeline of another score. In that case, we call it a Folder. Events have distinct colors in the editor, which are assigned automatically but can also be changed by the user.

Chain (UChain)

A Chain holds a series of Units (U). Together they form one processing chain, where usually a sound is generated, processed and sent to the outputs of the system. Via the editor the user can design the signal path and contents of the Chain. A Chain is a self-enclosed environment. Sounds and processes in a Chain cannot interfere with sounds in another Chain; multiple Chains operate in parallel to each other. If you want sound processes to interfere with each other, it all needs to happen within one Chain (more about this in the "Parallel architecture" topic below).

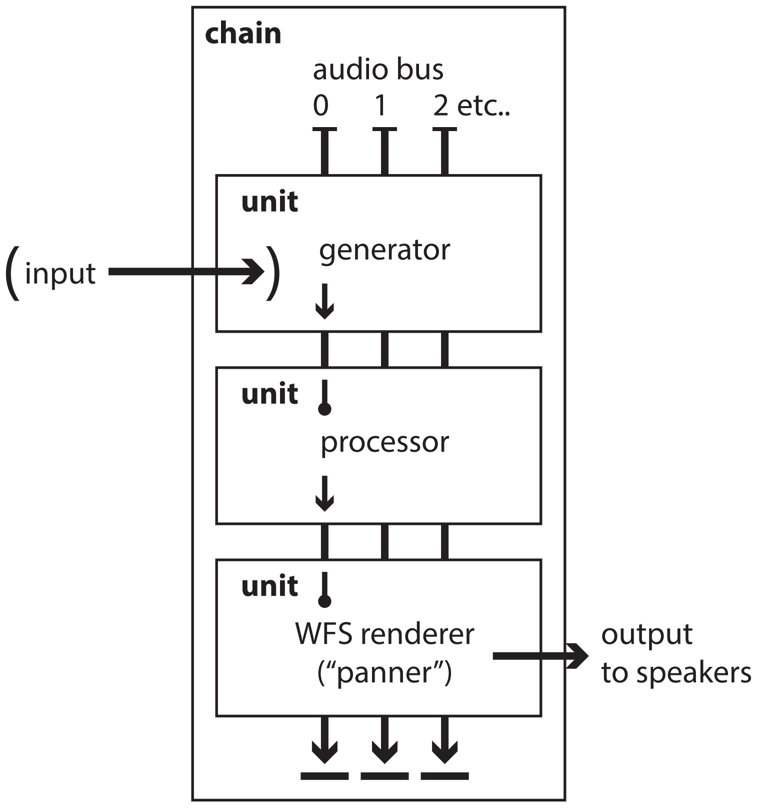
The duration of a Chain can be either a fixed time or 'infinite' (inf). Where a fixed duration Chain will automatically stop playing after it's duration has elapsed, an infinite duration will cause the Chain to keep playing indefinitely, until it is stopped by the user. If a Score includes an infinite duration chain, the duration of the score will also become infinite. Infinite chains are typically useful for live performances. Please keep in mind though that even if the duration of a Chain is set to infinite, that doesn't mean all the units in the chain produce sound infinitely. For example if there is a soundfile involved it might need to be set to 'loop', so that it will also play indefinitely.

Chains also have a 'releaseSelf' setting which defines their behavior inside a Score. If 'releaseSelf' is enabled (default) the Chain will take care of its own ending. If the Score is paused after the Chain was started, the Chain will keep playing and end itself after it's duration has passed. This is analogous to the "one shot" setting on some commercial sampler plugins and machines. If 'releaseSelf' is disabled however, the Score becomes in charge of when the sound ends. If the Score is paused while the Chain is playing, the Chain will keep on playing until the Score is unpaused again and passed its original end time.

Unit (U)

A Unit is a single processing entity. Units can only function as part of a Chain. The concept of a Unit could be compared to that of a plugin in commercial DAW's, both instrument and effect. But also the output stage is a Unit WFSCollider. For example, there can be a Unit that plays a soundfile, in a Chain together with a Unit that makes a WFS source. The signal will be routed from the soundfile Unit to the WFS Unit inside the Chain. If there would be no WFS Unit (or any other Unit that outputs sound), there will be no sound; the sound is generated but doesn't reach the outputs. Vice versa if there would be only a WFS Unit but no soundfile or other type of sound generating Unit in the Chain, there is also no sound. Then the WFS Unit will only output silence.

The Unit is the spine of the Unit Library (which makes it's name not an entire coincidence). They enable users to have full control over the signal flow in a modular way, and keep track of what is going on in both an intuitive graphical representation and a just as intuitive coding environment (this document will focus mainly on the graphical environment).



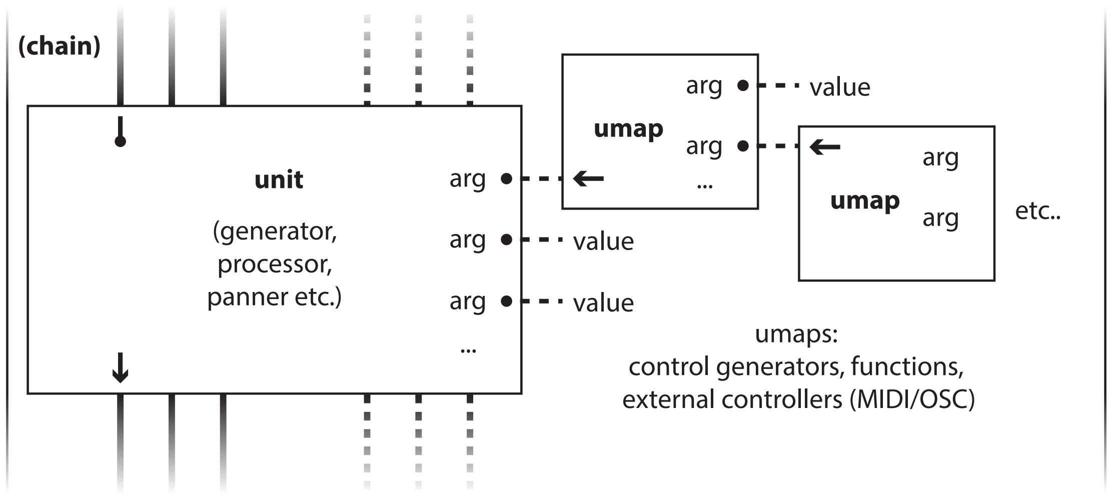
Udef (Udef)

Udefs hold definitions of what a Unit can (and will) do. Each Unit is based on one of the available Udefs. The Udef can be regarded as the "blue-print" of a Unit. There are Udefs for generating sound, for processing sound and for outputting sound. And there are special Udefs that create the more complex WFS panning engines. Udefs are stored as SuperCollider code files. The default set of Udefs that comes with WFSCollider is inside folders named “UnitDefs”, inside the Unit Library and WFS Library folders (located in the Resources folder inside the applications bundle). The currently loaded Udefs can be viewed in the Udefs window (menu View->Udefs), from where there are links to the code files. Udefs are recognized by their (exact) name. Loading a new Udef will replace any already existing Udef by the same name. More Udefs can be added by coding them and putting them in those folders, or they can be placed in the same folder as a .uscore file that uses it.

Normally, the default set of Udefs is enough to do many different things in WFSCollider. More specific info about all available Udefs can be found in the Interface (or GUI) section: Udefs Window. To change the Udef of a Unit, or add a new Unit based on a certain Udef, the Udefs can be dragged and dropped from the Udef window into the UChain editor window. Udefs are organized along a number of categories, such as ‘synthesis’, ‘soundFile’, ‘effect’. The WFS specific Udefs are in categories that have ‘wfs_’ in the name (‘wfs_panner’, ‘wfs_io’). Also usually their individual names are preceded by ‘wfs’ (i.e. ‘wfsStaticPoint’).

UMap / UMapDef

UMaps are a special type of Units. They can be used to automate and modulate parameters (“args”) of a Unit, map them to MIDI/OSC controllers or apply other functions to them. Units can receive UMaps for almost any parameter. The system automatically recognizes which UMap can be assigned to which parameter, and the output of the UMaps is automatically mapped to the value range of the parameter it is connected to. The parameters of the UMaps can themselves again be mapped to another UMap, creating endless modulation and control possibilities. UMaps have their own definition system, analogous to the Udef system for normal Units. Common examples of UMaps are LFO’s, envelope generators, MIDI controllers and randomizers. There are special UMapDefs for Point and range type parameters, including a trajectory generator.


(chain)
unit
(generator, processor, panner etc.)
arg
arg
arg
...
value
value
umap
arg
arg
...
value
umap
arg
arg
etc..
umaps:
control generators, functions, external controllers (MIDI/OSC)

Marker (UMarker)

Markers are a different type of Events that can be placed on a Score's timeline. Markers don't play sound, but they can perform tasks such as pausing the timeline, jumping forward or backward or even basic automation. Markers can also hold text for example for comments on things in the score. Each marker can have a name to be displayed in the score editor (names don't need to be unique), and with the score editor back/next buttons the user can jump between the positions of markers in the score. Basically markers fire a SuperCollider function when they are played. There is a number of presets available with various useful functions ready for use, but they can also be edited directly from the graphic interface. A Marker event doesn't have a duration, only a start time (i.e. the duration is always 0).

EQ (UGlobalEQ)

There is a global equalizer available that works on all sound outputs. It can be edited in the EQ window (menu View->EQ). It has a low shelf and high shelf filter, and 3 peak bands. It also includes a gain parameter. This EQ also holds the system's built-in limiter and DC offset removal. These cannot be edited; they are there to protect the system's speakers. The limiter is designed as a soft clipper, which will kick in when the level is more than 12dB above 0. This will cause audible distortion. The limiter is set up behind the EQ, which is why there is a gain parameter in the EQ.

Level (UGlobalGain)

The large level slider at the right side of the screen is the global level. It is applied to all outputs of the system. The nominal level is 0dB.

Bar/beat vs. time (TempoMap/BarMap)

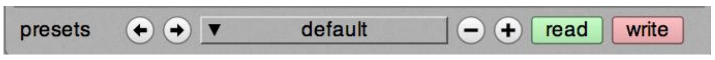
The Unit Library features a tempo and signature mapping system. The user can decide if the timing in the Score should be based on seconds, or on beats/bars in a certain tempo. There can be tempo changes and signature changes as well, which are stored in the TempoMap object and saved automatically with the score.

Folders / nested Scores

On a Score's timeline there can be UChains and UMarkers, but there can also be whole Scores. This is what we call "folders", or nested Scores. They can be created from selected events with the 'folder' button, and unpacked with the same button. Double-clicking the event will open up the folder and give access to its contents. Folders are a way of grouping events; moving a folder across the timeline will move all events in it. The folder objects function as completely separate scores, which can also be started/stopped individually. Using the [^] button will open the folder score into a new window, giving the possibility to save it as a separate score. Please note that folders have their own timeline and tempo map, independent of that of their parent score.

Presets (PresetManager)

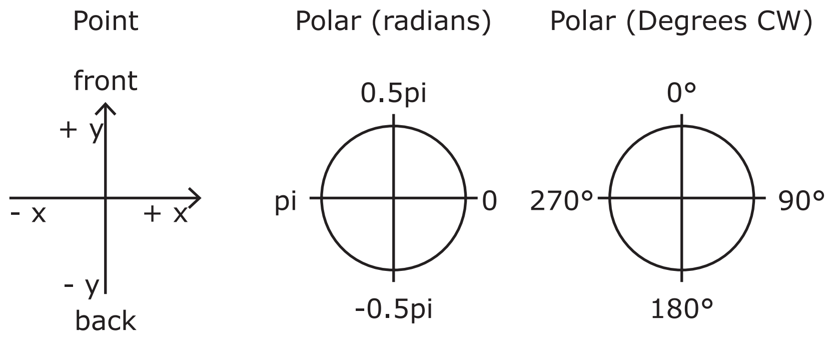
WFSCollider features a global preset system, which is used in various places. There are PresetManagers for EQ, UChains, UMarkers, WFS Speaker configurations and WFS options. It works the same everywhere; a preset can be chosen from a flip menu at the bottom of the page. This changes the settings for the objects to the stored preset. In case of UChains, it will replace the Units in the Chain with those from the preset. The arrow buttons at the left side of the menu let the user jump back and forth between the preset and the last state of the object before the preset was applied. The + / - buttons at the right side can remove or add presets to the list, based on the current state of the object. If you add a preset and give it the same name as one of the presets that is already in there, it will be replaced. The [read] and [write] buttons allow the user to save the current bank of presets to a file, or read them back from an earlier saved file. By default the PresetBank will show a predefined set of presets, mostly showing examples of what can be done, or forming starting points for new objects.



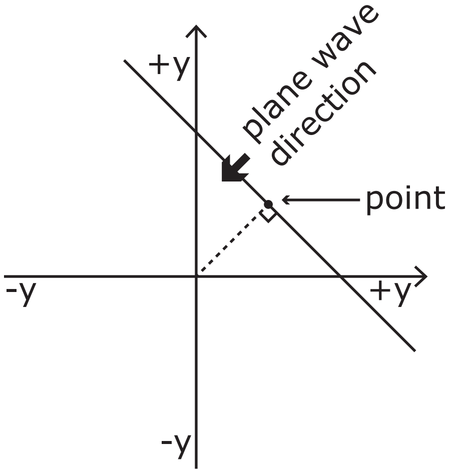
Point (Point)

Point objects are used in WFSCollider to define the position of a source. They represent the location of a point source, but also the location and direction of a plane wave. The system's two-dimensional space is represented in the following way:

- x axis (in meters) from left to right; negative to positive
- y axis (in meters) from back to front; negative to positive
- point (0,0) is the center of the room
- with normal polar conversion, a radian angle of 0pi means straight right, and from there angles move counter-clockwise
- in the graphic interface we also use "deg_cw" as units for angle; they are defined as 0 to 360 degrees clockwise, where 0 means straight front.



For plane waves, the defined point means the point on the plane that is nearest to the center of the room.

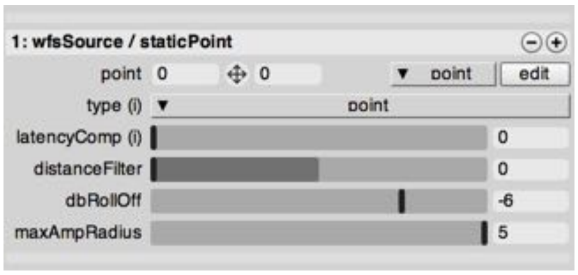


Static and dynamic sources

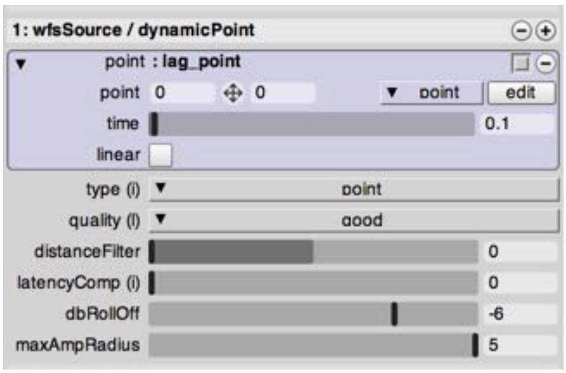
WFSCollider makes a distinction between static and dynamic WFS sources. Static sources are not allowed to move in real-time. This limitation gives way to great optimization, so that the system is able to handle many more static sources simultaneously then it could with dynamic ones. But, be sure to know, changing the position of a static source while playing it will not cause it to change audibly; the event needs to be stopped and started again to hear the effect. Dynamic sources, on the other hand, can move. They use real-time optimization, which makes them quite efficient, but only about half as efficient as static sources.

The ‘wfsSource’ Unit is used for both dynamic and static sources. By default the unit is static. The unit becomes dynamic when a UMap is applied to the ‘point’ parameter. This UMap can be anything from the ‘point’ and ‘trajectory’ categories, or an ‘expand’ (which provides the ability to connect individual UMaps to the x and y positions of the point). As soon as the UMap is removed, the unit becomes static again. The pictures below show a static source, and a dynamic source with a ‘lag_point’ UMap attached. The ‘lag_point’ is typically used for situations where the point is controlled live, by hand or controllers, and smoothens sudden changes in position to prevent Doppler shift artifacts and clicks.

The dynamic source offers two quality modes: ‘good’ (default) and ‘better’. The ‘good’ mode uses linear interpolation for the delays involved in the WFS algorithm. They sound ok in most cases but can cause some artifacts. In ‘better’ mode the interpolation is done via cubic splines (Hermite), which results in a better sound quality and more precise positioning. The ‘better’ mode requires apx. 1,5 times more CPU power from the system when compared to the ‘good’ mode.


1: wfsSource / staticPoint

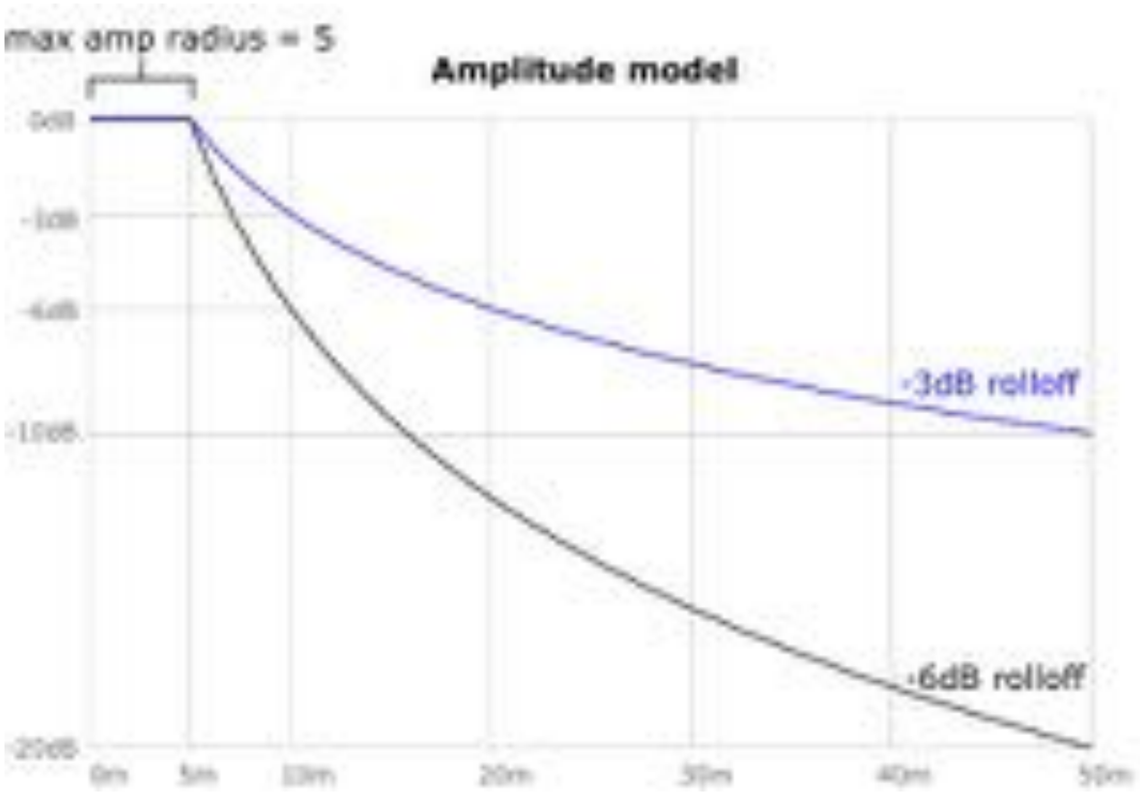
point	0, 0 ▾ point edit
type (i)	▾ point
latencyComp (i)	0
distanceFilter	0
dbRollOff	-6
maxAmpRadius	5


1: wfsSource / dynamicPoint

▾ point : lag_point	
point	0, 0 ▾ point edit
time	0.1
linear	☐
type (i)	▾ point
quality (i)	▾ good
distanceFilter	0
latencyComp (i)	0
dbRollOff	-6
maxAmpRadius	5

Amplitude model

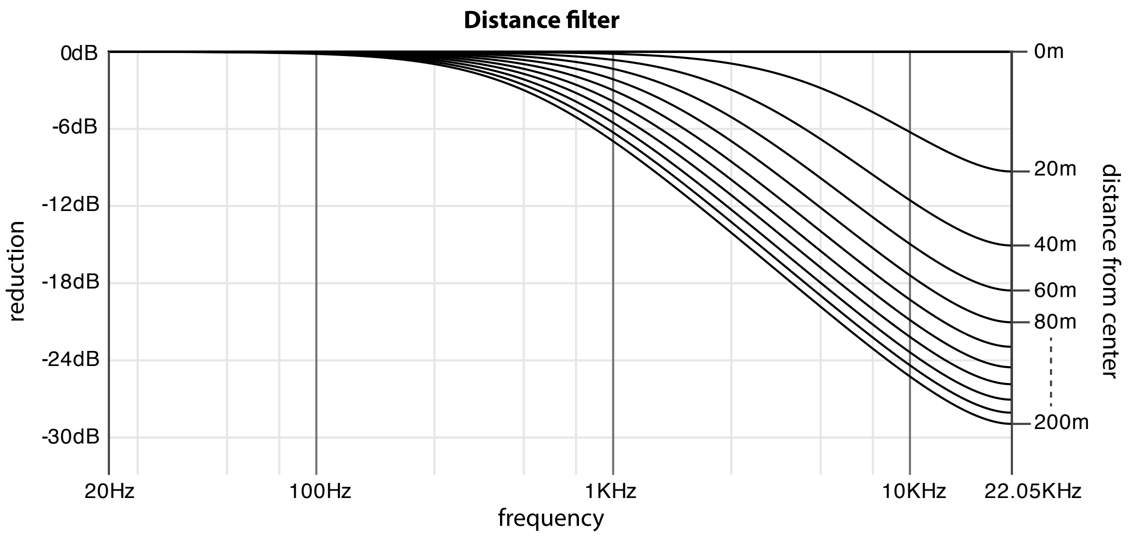
In WFSCollider we use a unified amplitude model, that is a little different from that of most other WFS rendering software. The overall amplitude (level, volume) of a source is calculated from the distance of the source to the center of the room. In the center of the room there is an area which we call the 'maxAmpRadius', where the level is always at it's maximum (relative to the gain of the source and the global level). When outside this area, the level drops with a user definable amount of dB's per distance doubling. In the default settings, the maxAmpRadius is 5m, and the dB roll-off is -6dB per distance doubling (which is the expected roll-off in an anechoic environment). In many cases however, a roll-off setting of -3dB per dd. will sound more natural to our ears. For plane wave sources, the default roll-off is 0dB (because ideal plane waves don't lose energy over distance).



Distance filter

WFSCollider features a filter for emulating distance in the point source rendering. This filter is designed according to a formula by Richard W. Furse (http://www.muse.demon.co.uk/vspace/model.html). It implements a 1-pole low pass filter with a cutoff frequency that is related to the distance of the source from the center of the room. The formula is as follows:

cutoff = 100000 / distance



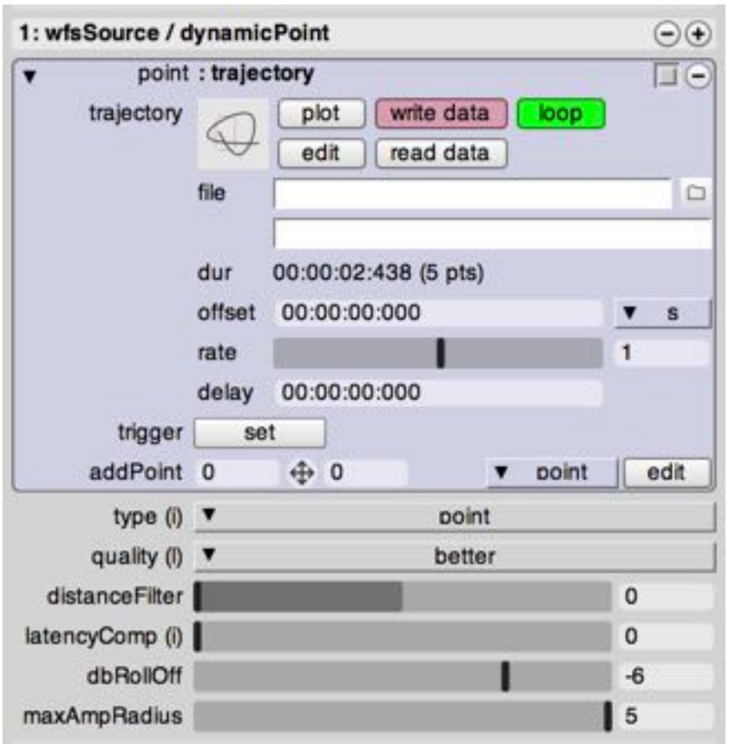
This filter is implemented as a separate Udef that can be inserted in the chain, but it is also included in the wfsSource panner. There it can be enabled via the 'distanceFilter' slider. The amount of the slider is a cubed multiplier for the used distance; when the slider is set to '0' the filter will never be audible, when the value is 1 the response will be that of the normal function as defined by Furse, and when the value is higher the effect will be exaggerated. When the value is 2 that means that the reference distance for the filter is multiplied by 8 (2 cubed; 2 to the power of 3).

Latency compensation / Doppler reduction

When calculating wavefronts in WFS we introduce a delay in the sound. This is caused by the distance of the source to the listener and the speakers, and would be there in the physical world as well. This means that if two sources are timed together, but in different places in the room, they will not be in sync. It also means that if a source is moving, the delay is modulated, creating an audible shift in frequency (known as the Doppler shift). Now, it is possible to (partly) reduce these effects, in case are not musically desired. This function is named 'latencyComp', and is featured on all the wfs_panner Udefs. The latencyComp effectively reduces the larger part of the induced delay. When fully effective there is only the delay differences between the speakers left, and not the delay caused by the distance of the source from the center of the room. This eliminates a large part of the Doppler effect, and makes sure that sources can be in sync with each other regardless of their placement in the room. The latencyComp function can be controlled via a slider, which ranges from 0 (no compensation) to 1 (maximum compensation). Latency compensation also works on static sources, where it influences the timing of the event. Note that this value is only used at the start of an event; changing it during playback will not give audible results until the source is started again. Also note that some Doppler shifting can always be heard, caused by the delay differences between the individual speakers.

Path /Trajectory (WFSPath)

In WFSCollider it is also possible to make sources move along a certain spatial trajectory. These are called WFSPaths. They consist of a collection of Points, with transition times in between. WFSPaths also hold information about the way the values in between the points are calculated. WFSPaths can be played by 'trajectory' UMaps, which can be assigned to a 'wfsSource' unit's 'point' parameter.



1: wfsSource / dynamicPoint
point : trajectory
trajectory
file
dur 00:00:02:438 (5 pts)
offset 00:00:00:000 s
rate 1
delay 00:00:00:000
trigger
addPoint 0 0 point
type (i) point
quality (i) better
distanceFilter 0
latencyComp (i) 0
dbRollOff -6
maxAmpRadius 5

The duration of a Path is not coupled to the duration of an Event. Paths can also be looped, started with an offset and played at different speeds. There are two ways to store WFSPaths:

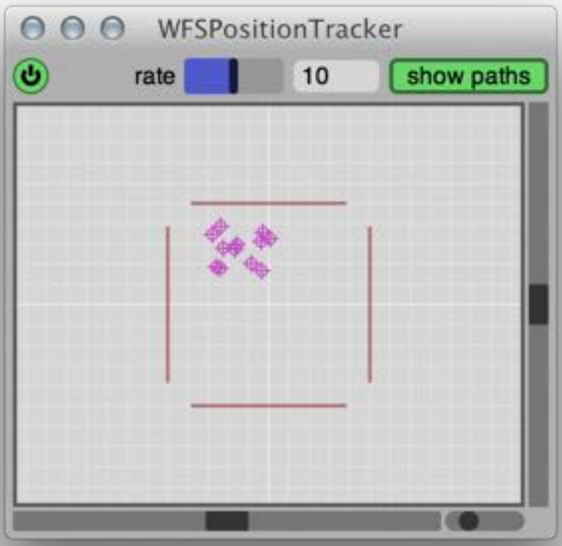
- in the Score
- as separate "raw data" files (.wfspath extension)

To save a WFSPath as raw data you need to click the "write data" button in the Chain editor, and specify a location on your hard drive where the file should be saved. If you save it in the same folder as your Score file, it will be located automatically next time you open the score, even if the whole folder is moved to another place or machine. If you don't do "write data", then the WFSPath will be saved inside the score file itself. Note that Paths with raw data files play back more reliably than the ones saved in the Score, because those have to be sent completely via network every time they are played back. This makes them vulnerable to network latency due to high traffic load, and also increases network latency for other Events played at the same time. In general it is recommended to use the "write data" function.

WFSPaths can be edited and designed via the WFS Path Editor. This editor is called up via the [edit] button in the trajectory UMap GUI in the Chain editor.

WFS Position Tracker (WFSPositionTracker)

The WFS Position Tracker can be found in the menu Views -> WFS Position Tracker. This is a global monitoring system for all active WFS sources. Each source will be shown as a point or plane in the window, and will have the same color as the event to which it belongs. The information comes from the actual rendering servers. It is not possible to edit positions from this window, only to watch them. After the position tracker is enabled (via the ‘power’ button in the upper left corner) each new started event will be registered to the tracker. Any events that were already playing will not be tracked. The ‘rate’ slider sets the frame rate of the tracking for dynamic sources. Note that higher frame rates will result in more network traffic and CPU use.

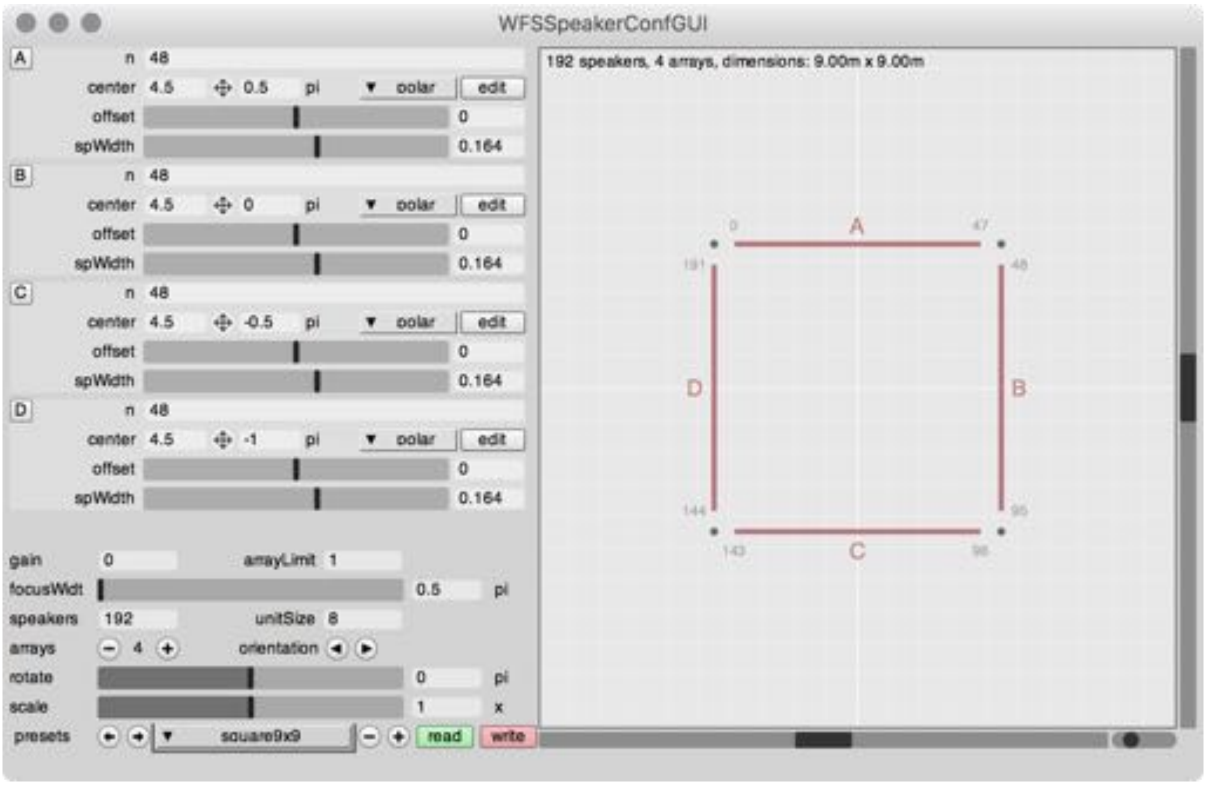


WFS Speaker Configuration (WFSSpeakerConf)

The WFSSpeakerConf holds the current setup of the speakers of the system. To be able to perform proper WFS rendering, WFSCollider needs to know the exact positions of all the loudspeakers. These are defined in the WFSSpeakerConf. This can be edited in the menu WFSCollider –> preferences... (or View -> WFS Preferences... on Linux/Qt), [edit speaker configuration] button. The default speaker configuration is that of the Game Of Life system in its widest setup: a 10m x 10m square with 192 loudspeakers (4 x 48). The speaker configuration is displayed in all WFS point, plane and path editors, as a reference for the dimensions of the system. It is also shown when running in offline mode.

WFS Speaker Configurations always consist of one or more arrays of loudspeakers (shown as red/brown lines in the edit windows). Each array is pointing to the center of the room. Systems can consist either of a single array, or a set of arrays which together form a closed shape (rectangle, square, hexagon etc..). The configuration doesn’t need to be symmetric. In general, using less and larger arrays is better. Also there cannot be two arrays with the same angle next to each other, or in front of each other.

If a WFS Speaker Configuration consists of a single array, there are some settings elsewhere to be taken care of. These are 'useFocusFades' and 'tapering', both found in the main preferences window. Any change to these settings will require a rebuild of the system's SynthDefs, which can be done via the button right below the settings (the rebuild will take about 30 seconds). The useFocusFades setting needs to be off for single array systems to be able to use the whole field in front of the loudspeakers. The tapering setting is advised to be 0.1 (10%) for single array systems, and 0 (zero) for any other configuration.



```
A  n       48
   center  4.5   0.5   pi   polar
   offset  0
   spWidth 0.164
B  n       48
   center  4.5   0     pi   polar
   offset  0
   spWidth 0.164
C  n       48
   center  4.5   -0.5  pi   polar
   offset  0
   spWidth 0.164
D  n       48
   center  4.5   -1    pi   polar
   offset  0
   spWidth 0.164

gain       0          arrayLimit  1
focusWidt  0.5 pi
speakers   192        unitSize    8
arrays     4          orientation
rotate     0 pi
scale      1 x
presets    square9x9

192 speakers, 4 arrays, dimensions: 9.00m x 9.00m
```

WFS Preview Mode (WFSLib.previewMode)

WFSCollider offers the possibility to work on WFS material at home (i.e. with no actual WFS system present). This works via the previewMode. By default, if no changes to the settings are made, WFSCollider will start up with previewMode set to 'headphone'. This means that all WFS sources will be made audible using a headphone emulation of the effect. The doppler shift and level differences are the same as in a real WFS system. The previewMode can be set via the preferences panel (menu 'WFSCollider –> preferences...' or 'View -> WFS Preferences...' on Linux/Qt). There the previewMode can be selected via a popup menu. You need to press the [apply] button to make the change effective, and if you want to save your change, press the [save] button.

There are a number of previewModes available:
- 'headphone': a simple headphone emulation, using inter-aural time differences. Note that this preview is not "binaural", meaning that it doesn't make a difference between sounds coming from the back and sounds coming from the front.
- 'stereo': an emulation optimized for stereo loudspeaker setups.
- 'quad': an emulation for classic "quadraphonic" speaker setups.
- 'hexa': an emulation for circular 6-speaker setups
- 'octo': an emulation for circular 8-speaker setups

If you set the previewMode to ‘off’, that means that WFSCollider will not be in previewMode, but generate actual WFS rendering, based on the current WFS Speaker Configuration.

Server(s)

The Server-Client model used in WFSCollider is part of how SuperCollider (the language in which WFSCollider was written) works. A Server is a computer program named ‘scsynth’, which runs in the background. The Servers do the actual audio rendering. WFSCollider communicates with servers via network messages (which can also take place within a single computer) via the OSC format. In WFSCollider there can be multiple Servers active at the same time. In normal installations outside WFS systems we usually only use a single server called “wfs_master” (also referred to as the “master server”). This server needs to be running to be able to hear sound from WFSCollider.

Normally the server starts up automatically when you start up WFSCollider, and quits again automatically when you exit the program. In some cases this can go wrong (for example if WFSCollider crashes) and the server program may keep running. Also it may happen that WFSCollider looses its connection to the server (for example when an audio interface is unplugged during operation), and is unable to start a new server. In such occasions, where the server won’t start, you can use the [K] button in the WFSServers window. This ends all scsynth processes on your machine, making way for new ones. After pressing the [K] button, you can start up the server again with the power button next to the “wfs_master” label. If the power button and the “wfs_master” label are both green, that means the Server is running.

On WFS systems, WFSCollider usually uses multiple Servers. These don’t necessarily need to be on the same machine. For example on the Game of Life system, there are 17 server applications in use; one master server (on the system’s control laptop) and 8 Server on each of the WFS rendering computers (which confusingly are also called Servers). These can all be monitored from the WFSServers window. The reason for having 8 Server processes per machine is that these machines have 8-core CPU’s. Each Server process can only use one core at a time, so with 8 cores we make full use of the machine’s capabilities. WFSCollider intelligently divides the load among these Servers to ensure optimum performance.

On the Game of Life system the master server is used as a master for synchronization. But it is at the same time a fully functional audio engine, which can also generate sound, and is connected to its own audio interface. There are certain Udefs available which force your UChain to play on the master server instead of the WFS rendering servers. Examples of this are ‘wfsMasterIn’ and ‘wfsMasterOut’, which can be used to route live inputs to the system’s rendering machines, or to play sound on the analog outputs of the master audio interface in sync with the rest of the system.

Parallel architecture

The main reason for the UChain/Unit design choice is the fact that we use parallel processing on the WFS rendering machines. A Chain with all its enclosed Units forms one series of interconnected objects. It cannot interfere with signal paths or sounds in another Chain, because all chains run in parallel to each other (i.e. you cannot be sure that another Chain is runned on the same processor core, or even on the same machine). This means that if you want sounds to be able to interfere with each other, they need to be in the same Chain. The benefit of this parallel design is that we can get much more processing power from modern multi-core machines. The Game Of Life system currently uses 8-core processors, and the parallel design of WFSCollider makes use of these as if they were 8 different machines.

When running WFSCollider in offline mode, it only uses one core of your machine. This is ok because the ‘headphone’ previewMode takes far less processing power compared to the actual WFS calculations. The limitations of the parallel design remain, mimicking the behavior of the real system.

Synchronization

One of the challenges of designing and building a WFS system is in the synchronization between machines. To ensure correct wave field rendering all machines need to run in sample-tight sync. WFSCollider uses a dedicated system for this. It involves a patched version of the SuperCollider source (i.e. it will not work with normal SuperCollider installs), which is enabled to measure differences between sample clocks of machines, and use those to schedule events at exactly the same moment on multiple machines. This synchronization system is currently only used for the Game of Life system. When using WFSCollider in offline mode at home it is not needed, since only one machine is involved there. On the Game of Life system the synchronization can be done via the “sync” button in the WFSServers window. It needs to be done only once, at the start of the session. The numbers that appear in the boxes next to the Server labels are the measured sample differences between the machines and the master server (which is the sync reference). If these numbers are unstable (i.e. if they change after pressing “sync” for the second time) that means there is a problem with the wiring of the system. It is generally advised to press the sync button every now and then to check if it is still in sync, and also to ensure the best possible latency. Please check **Appendix B**: Working with the Game of Life system to read further about this.

The “sync” button is also available in the offline mode. There it can be used to emulate the sample-synchronous timing that is used on the large systems. After pressing “sync” once, the red indicator becomes green and we are in sample-sync mode. Please note that event start times will be rounded to the program’s buffer size; 128 samples.

Latency

The latency slider in the upper right corner is also about synchronization. The system is operated via network messages. There is certain unpredictability in how long these messages take to arrive at their destination. If a message arrives after the time an event was scheduled to play, the event will start too late, and not in sync with the rest of the system. For this reason we need to give the messages some time to arrive, which we call the latency.

The default latency is set to 0.2 seconds. This is usually more than enough time for everything to arrive. However, for more complex scores where many events are started at the same moment, the network traffic increases and the 0.2 seconds may prove to be too short. In such occasions you can simply increase the latency time. That will also mean that the system will respond slower to changes. The time between hitting play and hearing the first sound is roughly the same as the latency. In the score editor, during playback, a second playback position line shows the playback time corrected by latency. The gray area in between the two playback lines means that events in that area are already started, but not yet sounding.

In some situations it could also be desirable to lower the latency. When playing only one or two events at the same time, the latency could be set as low as apx. 0.05 seconds. This means that the system will respond almost instantly to any change, which is obviously most useful in live situations.

Note that some parameters (such as the ‘amp’ in most Udefs) don’t use latency. They will always be changed as fast as possible. This is only used on parameters that don’t need to be changed in perfect sync.

Also note that the latency setting is not about live sound input. Live sound input is discussed in the “More about Units and UChains” section of this manual, and its latency depends on (usually fixed) system and soundcard settings.

The Menus

Note that the terms Chain and UChain, Score and UScore, Def and Udef, etc. have a one on one relation, but in the interface they are both used. The names with the U's in front are the names of the actual classes, the names without a U, are the names of the interface elements. And some interface elements are named after their classes.

Score

<u>New</u>
Create a new Score. You can create as many Scores as you want. The Score is like a Timeline in a Sequencer or DAW.

<u>Open...</u>
Load a Score from a disk.WFSCollider can only open uscore files via this menu item (i.e. not directly from the Finder or with File->open)

<u>Save</u>
If this is the first time you save this Score, a Save window will open, otherwise the Score will be saved.

If the Score is saved, an extension, .uscore, will be added to the name of the file. These are text files that hold the generated code to re-create the score. They can be edited in a normal text editor as well (not in WFSCollider).

To be able to use your scores on the Game Of Life system, you need to make sure that the soundfiles you use are in the same folder as the .uscore file. WFSCollider will then write relative file references in the saved uscore file, so that the whole folder can be moved anywhere.

<u>Save as...</u>
Save the Score under a different name.

<u>Export as audio file...</u>
Export the Score as an .aiff file, depending on previewMode the settings in the Preferences menu.

<u>Add Event</u> (Sh-Opt-Cmd-A)
Adds a new Event to the Score.

<u>Add Marker</u>
Adds a new Marker Event to the Score. Markers are special events that can contain text notes about the score, and perform simple functions such as pausing the score at a certain moment, or jumping to a different position.

<u>Edit</u> (Opt-Cmd-I)
Edit one or more selected Events.

<u>Delete</u> (Opt-Cmd-R)
Delete one or more selected Events.

<u>Copy</u> (Sh-Opt-Cmd-C)
Copies one or more selected Events.

<u>Paste</u> (Sh-Opt-Cmd-P)
Pastes one or more selected Events at the position of the Timeline in the score that is currently in front.

<u>Select Al</u>l (Opt-Cmd-A)
Select all Events.

<u>Select Similar</u>
Selects Events that have the same Udefs in the same order in their Chains. Or that are otherwise of the same type.

<u>Sort Events</u>
Sort the Events in the current Score by their start time. This means the indices of the events (numbers shown in each event of the Score Editor) will be re-assigned according to the order of events on the time line.

<u>Overlapping events to new tracks</u>
Overlapping Events are moved to an empty track.

<u>Remove empty tracks</u>
Removes empty tracks between the Events

<u>Disable selected</u> (Opt-Cmd-M)
The selected Event will be disabled. Disabling is different from muting. Disabling an event will make the score player ignore it, i.e. not play the event at all. It will not immediately stop playing if it already is, and will also not immediately start playing if it is enabled during playback. This is different from the "mute" function, which can be found in the Mixer and in the UChain editors.

<u>Enable selected</u> (Opt-Cmd-U)
The selected Event will be enabled.

<u>Enable all</u>
All Events will be enabled.

<u>Enable selected and disable all others</u> (Opt-Cmd-P)
Only the selected Events will be enabled.

<u>Add Track</u>
Add an empty Track to the Score. Tracks are used only for visual purposes.

<u>Remove Unused Tracks</u>
Removes added Tracks that are empty. There is always a minimum of 12 Tracks.

Session

(The Session window is partly functional, but it is still under construction. There are some bugs as well).

New (Opt-Cmd-N)
Create a new Session. You can create as many Sessions as you want. The Sessionis like a kind of Playlist.

Open...
Load a Session from a disk.

Save (Opt-Cmd-S)
If this is the first time you save this Session, a Save window will open, otherwise the Session will be saved.

If the Session is saved, an extension, .usession, will be added to the name of the file. It holds all information present in the session, including the scores.

Save as... (Sh-Opt-Cmd-S)
Save the Session under a different name.

Add > New > UChain
Add a new Chain to the Session.

Add > New > UChain Group
Add a new Chain Group to the Session.

Add > New > UScore
Add a new Score to the Session.
(Bug: you cannot delete the UScore with the Delete button)

Add > New > UScoreList
Add a new Score List to the Session.
(Bug: does not add anything)

Add > Current score
Add the Score that is the topmost Score of all Score windows to the Session.
(Bug: do not use the shortcut, it will add an Event to the Score).

Add > Current score duplicated
Add a copy of the Score that is the topmost Score of all Score windows to the Session.

Add > Selected events
Add one or more Events that are selected in the topmost Score of all Score windows to the Session. If you select an Event, a UChain will be added. If you select a Folder, a UScore will be added.

Add > Selected events flattened
Add one or more Events that are selected in the topmost Score of all Score windows to the Session. When you select a Folder, the Folder will be added as separate UChains.

Add > Selected events into a UChainGroup

View

EQ
Opens the Master EQ, or brings it’s window to front.

Level
Opens the Master Volume, or brings it’s window to front.

UDefs
Opens the Udef window, or brings it’s window to front.

Meter
Opens the Level Meters, if not already open. The number of meters is dependent on the confituration in 'Audio Midi Setup'. On the Game Of Life system this only meters the master server, not the actual system servers.

WFS Position Tracker
Opens the WFSPositionTracker window. This allows you to view the current position of active events on the system. To use the tracker: click the ‘power’ button in the left top corner. From now on every new event will be tracked. The speed of the tracking (polls per second) can be changed with the slider. Faster speeds result in more fluent animation, but cost more CPU and network traffic.

Window

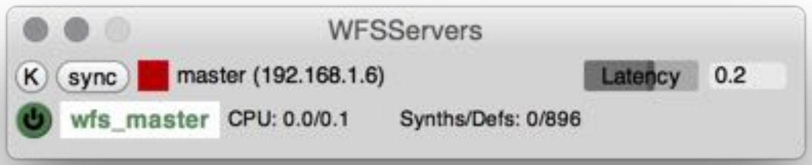
In the Window menu at the bottom you can select a window so it comes to the front.

Help

Here you can find a few help files. Look under getting started with WFSCollider: here you will find the WFSCollider V2.0 Overview. In general, these help files focus more on the scripting and coding aspects of WFSCollider and the Unit Library. In **Appendix B** of this manual you can find a full walkthrough of the help system, guided by Arthur Sauer.

Interface (or GUI)

Server window



K
: Kills all Servers.

sync
: Pressing the 'sync' button enables the sample-sync operating mode of WFSCollider. This is required for running on the Game Of Life system, but not for running in offline version. In offline mode it can be used to test the timing of the score.

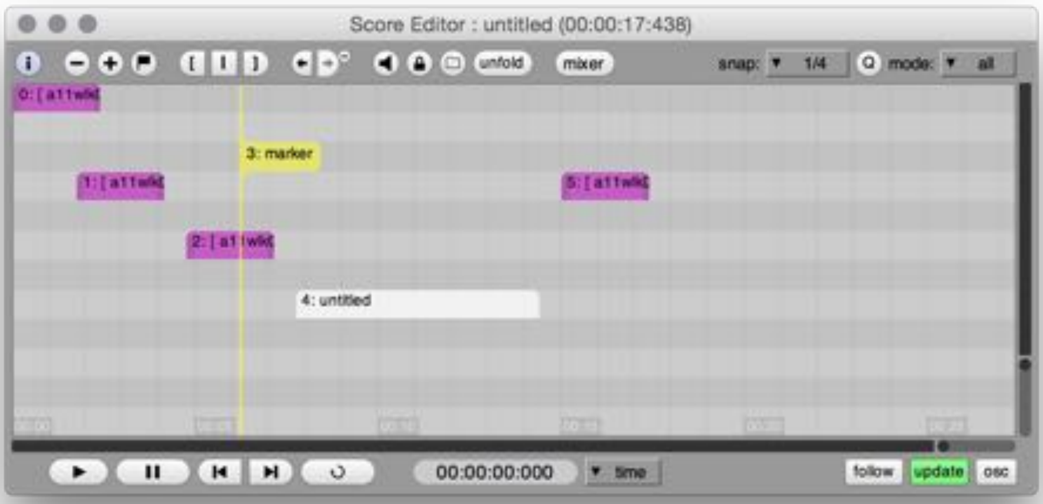
Latency
: Sets the latency of all servers. This represents the time it will take for an action to take effect. Lower latency will result in faster response, but might also cause asynchronicity. The default latency of 0.2s is good for most applications. For live use one might want to decrease this value, and very complex pieces might require a higher latency instead.

power button
: Start up the Server.

Post window

This is where all messages of SuperCollider and WFSCollider are posted.
To go to the Post window: click on it, select menu Window > Post window To Front, or Cmd-\.
To clear the Post window: select menu Window > Clear Post Window (Sh-Cmd-C).

Score window



interface elements on the top of the window

i
: Select an Event and click on the i-button to open the Chain window, where all parameters of the selected event can be edited. If no or multiple events are selected it will open the Mass Edit window, where the parameters of all events can be edited at once.

-
: Deletes selected event and delete it.

+
: If no Event is selected, a new, default Event will be added. If one or more Events are selected, they will be duplicated.

‘flag’
: Adds a Marker Event (UMarker) to the score.

[
: If no Event is selected, all Events are trimmed on the left side at the current play position. If one or more Events are selected, only the selected Events will be trimmed.

|
: If no Event is selected, all Events are split at the current play position. If one or more Events are selected, only the selected Events will be split.

]
If no Event is selected, all Events are trimmed on the right side at the current play position. If one or more Events are selected, only the selected Events will be trimmed.

<-
Undo the last edit.

->
Redo the last edit.

(-) (small minus button)
Clear undo history and disable/enable undo. Use this if your computer becomes slow after many edits

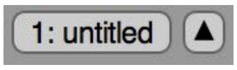
loudspeaker
All disabled selected Events are enabled and all enabled selected Events are disabled. Disabled events will be ignored during score playback.

lock
Locks or unlocks the startTimes of the selected events. Events can not be moved horizontally if their startTime is locked.

folder
Select one or more files to put them into a nested score, a.k.a. folder. To go into the folder, double-click the folder on the track. Once you are in the folder, you see the name of the folder in the toolbar, and next to it an arrow that points up. You can nest folders within folders.

unfold
Unfolds any selected folders.

after opening a folder via double click, two more gui elements will appear:



name of the folder
Click on the name of the folder to go back one level in the folder hierarchy.

up-pointing-arrow
When you are in a folder, you see an up-pointing-arrow in the toolbar. If you click on it, a new score with the contents of the folder will open.

mixer
Opens the Mixer window, where the levels and mute settings of all events in the score can be viewed and changed.

snap
When moving an Event, it will move in steps of the size of the setting in de snap pop-up menu. The available settings are: off, cf, 0.001, 0.01, 0.1, 1/32, 1/16, 1/12, 1/8, 1/6, 1/5, 1/4, 1/3, 1/2, and 1. ‘cf’ means the sample block size / control frame (128 samples) of WFSCollider, which is the actual timing resolution. When the score editor is in ‘bar’ mode, these values will correspond to fractions of a

beat. When in 'time' mode, they are fractions of seconds. Note that the times times of the events will not be rounded off to the snap value, but rather be incremented or decremented by the snap amount. When moving events using the arrow keys, the snap value is used as step size.

Q

Quantizes the startTimes of the selected events to the snap value. If no events are selected, the position line will be quantized instead.

Mode:

all

You can move, resize and add fades to Events.

move

You can only move Events, you cannot resize, or add fades to Events.

resize

You can only resize Events, you cannot move, or add fades to Events.
If you double-click on the Event, the Chain window will not open.

fades

You can only add fades to Events, you cannot move, or resize Events.
If you double-click on the Event, the Chain window will not open.

interface elements on the bottom of the window

play

Start or stop playback of the Score.

pause

Pause the playback of the Score. Soundfiles that were already playing are not stopped. Clicking on the pause button again will not play soundfiles that were already playing when pause was clicked the first time. Clicking on play twice will resume the Score where it was stopped.

back (|<)

Clicking this button will return the current play position to the start of the Score, or, if there are Markers in the score, to the previous marker position. If the score was playing when you hit this button, it will contiue playing from the new position.

next (>|)

Clicking this button will set the current play position to the next marker, or else to the end of the score. Like the back button, the score will continue playing.

loop

The loop point is the end of the last Event in the Score. The start point of the loop is the beginning of the Score.

Do not make the event to short, or the score will not loop. See the post window (Cmd-\):

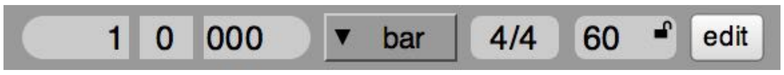
```
WARNING:
Score is too small, will not loop score. Would not have enough time to
prepare events.
```

<u>SMPTE timecode</u>

Shows the current play position. By clicking on it and moving with the left or right cursor keys on your keyboard, you can enter numbers and with the Enter or Return key you can set the current play position to this value.

<u>time / bar</u>

This flip menu decides if the score's timeline is displayed in seconds or tempo and signature aware bars and beats. When in 'time' mode, the SMPTE timecode display shows the position. When in 'bar' mode, a different set of controls appears:



<u>bar-map display</u>

Shows the current play position in bars/beats/divisions (1/1000). Each of the boxes can be edited separately, and the left/right arrow keys allow jumping between them.

<u>signature display</u>

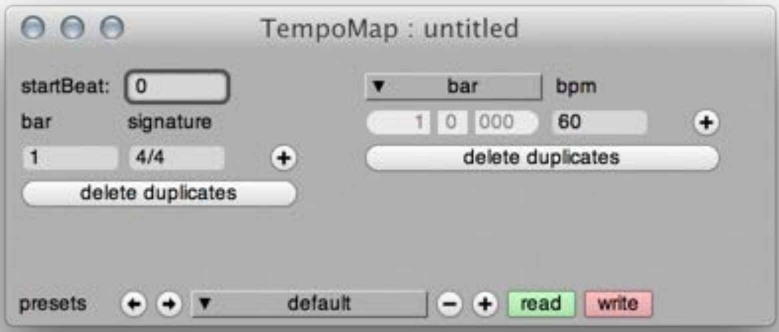
Shows the current signature. The default signature is 4/4 for the whole score. There can be signature changes. This box shows and edits the signature at the current score position. It can be edited by mouse dragging (alt or control drag for denominator change), arrow keys and by entering new signatures via keyboard.

<u>tempo display</u>

Shows the current tempo in BPM. The default tempo is 60 for the whole score, where 1 beat == 1 second. There can be tempo changes. This box shows and edits the tempo at the current score position. It can be edited by mouse dragging, arrow keys or entering new values via keyboard. The 'lock' icon in the upper right corner enables the tempo-lock mode. In this mode the startTimes of all events in the score (except the ones that are 'locked') will be altered according to changes made in the tempo.

<u>edit</u>

Opens up the TempoMap window, for editing tempo and signature changes in the score:



follow:
When enabled, this makes the score editor automatically zoom in to the current play position. This

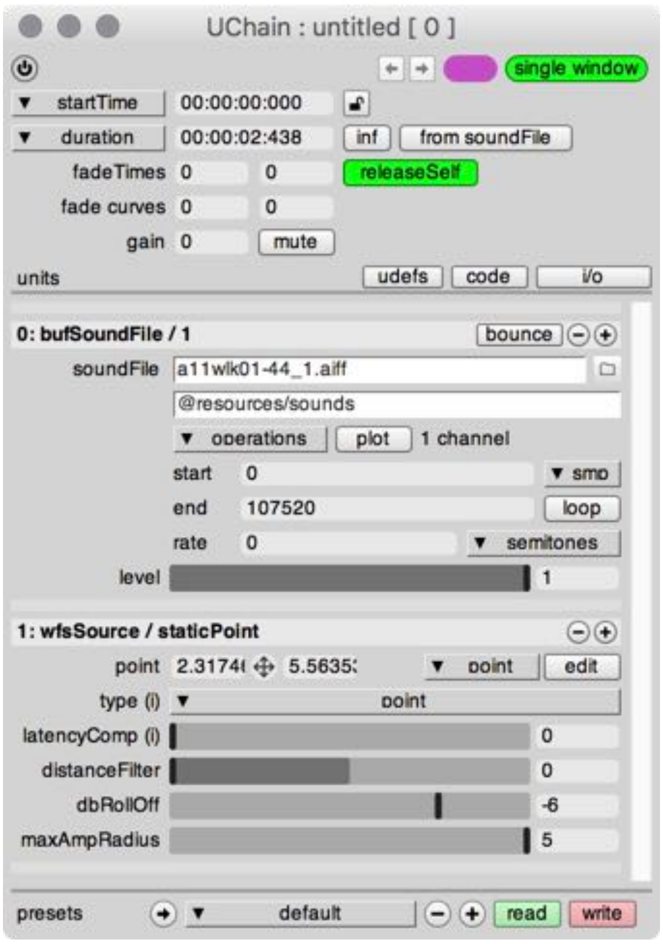
update:
When enabled, this shows the current play position in the Score during playback.

OSC:
When this is enabled, you can control the Score with OSC messages. In the post window (Cmd-\) some information will be shown about the port and type of messages the score responds to.

mouse
Shift-click on Events to select multiple Events.
Click-drag on an empty part in the Score to select multiple Events.
Double-click on an empty part to sets the current play position.
Double-click on the Event to open the Chain window.
Double-click on a Folder event to open it's score.
Cmd-drag: allows drag and drop of events to other score windows.
Alt-drag: copies an Event.

key commands
backspace: delete selected Events
space bar: start/stop/unpause playback
, (comma): start/unpause
. (period): stop
p: pause/unpause
l: loop on/off
d: duplicate selected
i: edit selected
0 (zero): Score position to zero
1-9: Score position to Marker index (or end of score)
+/-: Score position to next/previous marker
arrows up/down: change track of selected Events
arrows left/right: move selected Events or Score position by one snap unit

Chain window


UChain : untitled [0]

startTime 00:00:00:000
duration 00:00:02:438
fadeTimes 0 0
fade curves 0 0
gain 0
units

0: bufSoundFile / 1
soundFile a11wlk01-44_1.aiff
@resources/sounds
1 channel
start 0
end 107520
rate 0
level 1

1: wfsSource / staticPoint
point 2.3174(5.5635:
type (i) point
latencyComp (i) 0
distanceFilter 0
dbRollOff -6
maxAmpRadius 5

presets default

interface elements in the toolbar

<u>power</u>
You can hear the Chain by clicking on the power button of the Chain window on the top left of the toolbar.

<u><-/-> arrow buttons</u>
Undo/redo last change. This applies only to changes that require the window to rebuild; i.e. changing/adding/removing a Unit or UMap

<u>color box</u>
This shows the display color of the event. The default colors are dependent on event type. To change the color, click on this box to open the color editor. If a user-defined color is used, a small cross will show in the upper right corner of the box. Clicking that will remove the user-defined color, and reset to the default.

<u>single window</u>

If single window is selected and you double-click on an Event in the Score window, this will not open a new Chain window, but in an already existing Chain window.
Of single window is deselected, double-clicking on an Event in the Score window will open a new Chain window.

<u>startTime/startBar</u>

Shows the start time of the Event. Modes:
'startTime': shows the startTime in HH:MM:SS:FFF (F = 1/1000 s)
'startBar': shows the startTime in bar/beat format: bar/beat/div (1/1000 beat)

<u>lock</u>

When enabled, the startTime is locked to it's current value (in seconds). This state is not saved with the score.

<u>duration/endTime/endBar</u>

Shows the length or end time of the Event. Modes:
'duration': shows the duration in HH:MM:SS:FFF (F=1/1000 s)
'endTime': shows the end time of the event (startTime + duration)
'endBar': shows the end time in bar/beat format: bar/beat/div (1/1000 beat)

<u>inf</u>

Play the Event infinitely long. With soundfiles playing from bufSoundFile or diskSoundFile, you have to set the loop button in these Units to actually hear the soundfile infinitely long.

<u>from soundFile</u>

Sets the length of the Event by the length of the longest soundfile in the chain.

<u>fadeTimes</u>

Start and end fade time in seconds.

<u>fadeCurves</u>

A curvature setting for each fade time. 0 means a linear fade, negative values speed up the fade at the start, positive values slow it down. In the score editor the shape of the fade curve will be displayed.

<u>releaseSelf</u>

When releaseSelf is enabled, an event will have a fixed duration. It will take care of itself after the score playback engine has started it. This could also be regarded as "one shot" events. When releaseSelf is disabled, the score playback engine will take care of the releasing. This means, that if one would pause the score while this event is playing, it will keep playing until the score is resumed and passes the point where it was supposed to stop. This feature may be less reliable for very short events.

<u>gain</u>

Change the level of the sound.

<u>mute</u>

mute will mute the level of the event. Contrary to the "disable" button in the score editor (loudspeaker icon) the event will still play, only it will not be audible. This means a sound can be muted and unmuted during playback, and the effect will be audible immediately.

udefs
: Click on defs at the bottom of the Toolbar to open the Udefs window. Form there you can drag Udefs to replace or insert in the various Units below the line, and UMapDefs to drag UMaps to parameters of the Units.

code
: Click on code to see the code editors of the Udefs if the Udefs are editable. This calls up the code window (described below)

i/o
: This calls up the i/o window. Here you can change the connections between the defs.

params
: If you switched to the code or the i/o window, you will see a params button that takes you back to the parameters window.

parameters window

Below the line in the parameters window you will find the "units" of which the chain consists. Each Unit is based on a Udef (as found in the Udefs window), and takes care of a specific process in the chain. For example, there are Udefs for Units that can play a soundfile, or generate a sine wave. There are also Udefs for Units that can apply certain effects upon the sound of the previous Unit. And there are dedicated Udefs for Units performing WFS rendering. Each Unit has it's own set of parameters, which are shown and can be edited in the window. The signal path travels from top to bottom; each Unit gets sound and control input from the Unit above it. The Unit that brings the sound to the outputs of the system (such as the WFS panner Units) should always be at the bottom of the page; at the end of the signal path.

buttons

bounce
: This button records the output of the Unit into a new soundfile, and replaces the unit itself by a diskSoundFile unit playing it. It records only the output of the unit, and leaves the rest of the chain unchanged. If you need the soundfile, but don't want to replace the unit, press the 'undo' button at the top of the window after this action. This will restore the unit but not delete the soundfile you just recorded.

minus (-)
: Click on this button to delete the Unit. There always has to be one Unit in the chain for it to be able to play. To be able to hear the sound, there also must be at least one Unit that brings the sound to the outputs of the system (a WFS panner Unit, or for example an "output" Unit)

plus (+)
: Click on this button if you want to duplicate this Unit. A copy of the unit will be placed after it in the chain. The input and output buses of the copied unit (which can be edited in the i/o section) are automatically increased by one step, so that the unit outputs on different channels. This means that you can have multiple units of the same type working in parallel inside the chain. You can use the 'mixer' or 'splay' Udefs to mix their outputs together, or use

multiple output (i.e. wfs_panner) units in the chain. When doing this the Unit view goes into 'mass-edit' mode; you can edit all units of the same type at once. The mass edit mode is explained further elsewhere in this manual. To edit the units separately you can use either the [▲] button that appears instead of the 'bounce' button, or press the ► button that appears next to the 'units' label in the header of the window.

mouse

Drag a Udef from the Udef window on the pale gray rectangles or under a Unit, until you see a blue line, to add a Unit based on it to the chain. (the blue line unfortunately doesn't show on linux / qt versions).

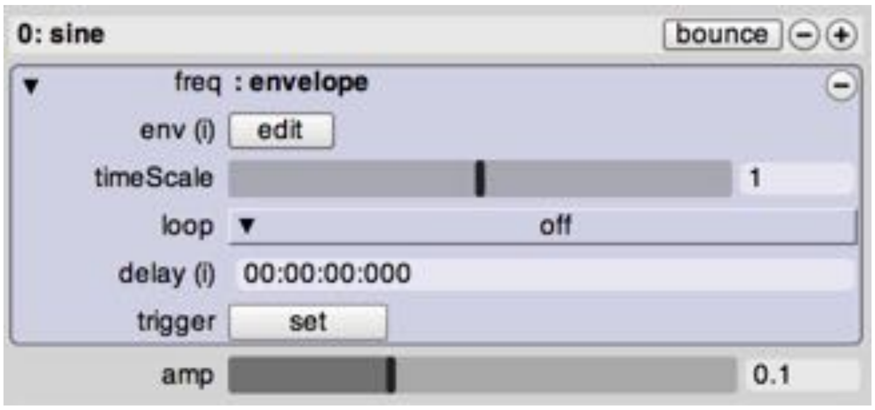
Drag a Udef on top of a Unit (you should see a blue rectangle) to replace it.

cmd-dragging a Unit will move it inside the chain, or copy it to another chain (drag and drop).

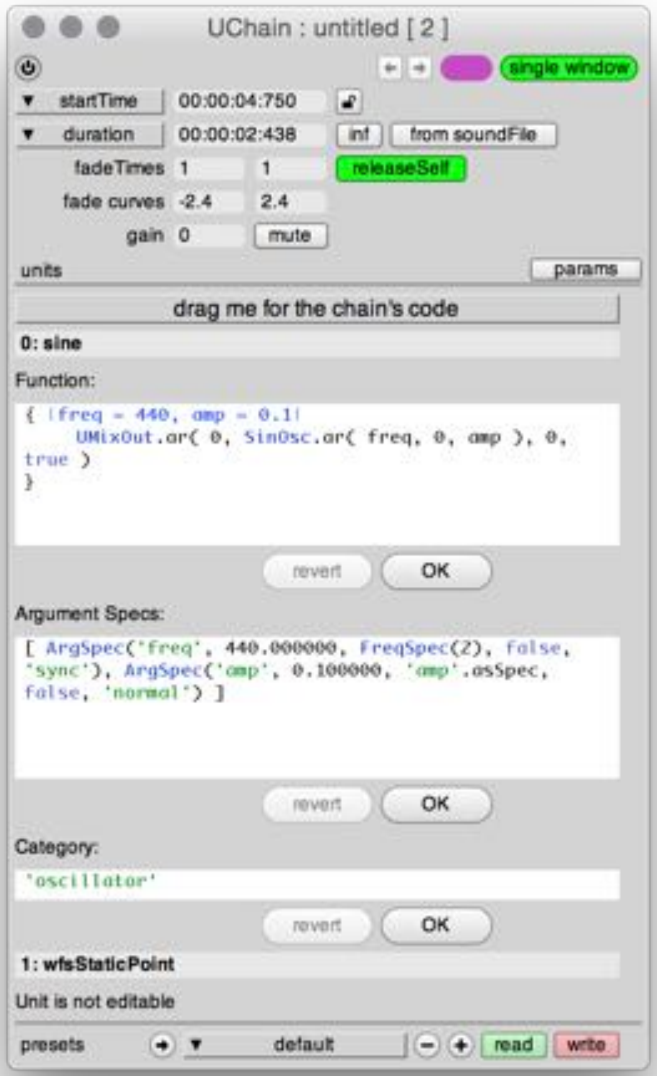
For each Unit you can change the settings of the parameters. Changing the setting will be immediately audible, unless there is a "(i)" behind the parameter name (meaning: 'init'), in which case it will only be effective at the next playback.

UMaps

Drag an UMapDef on top of a parameter name to connect it to a parameter. The parameters that accept the UMapDef will show a blue rectangle during the dragging. The picture below shows a 'sine' Unit with an 'envelope' UMap connected to it's 'freq' parameter. The '-' button will remove the UMap and restore the default value. Using the '▼' button you can show or hide the UMap's settings in the display.


0: sine
freq : envelope
env (i) edit
timeScale 1
loop off
delay (i) 00:00:00:000
trigger set
amp 0.1

code window



```
{ |freq = 440, amp = 0.1|
	UMixOut.ar( 0, SinOsc.ar( freq, 0, amp ), 0,
true )
}
```

```
[ ArgSpec('freq', 440.000000, FreqSpec(2), false,
'sync'), ArgSpec('amp', 0.100000, 'amp'.asSpec,
false, 'normal') ]
```

```
'oscillator'
```

If the Udef is editable, you see the code of the Udef. Changing code here will create what we call a "Local Udef". It will be used for this Unit only, and saved inside the score. Local Udefs keep the name of the original Udef on which the editing started.

For each Udef there are 3 parts in this window:

Function:
This is the code for that defines what the Udef will do.

Argument Specs:
This defines the buttons in the parameters window and how they look.

Category:
This defines the category folder in which the Udef will end up in the Udefs window

Below each of these 3 parts are two buttons: revert and OK.

<u>revert</u>
Undo the changes you made to the Udef. This is only possible before you save the changes.

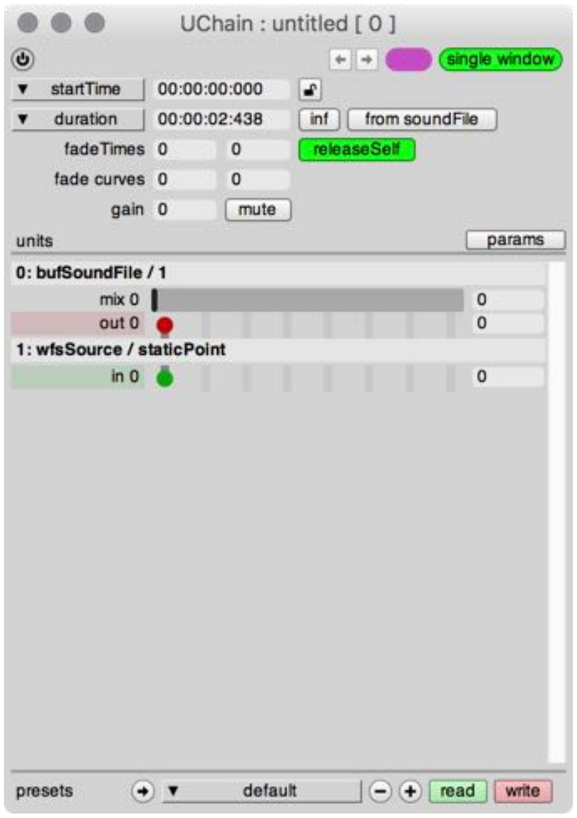
<u>OK</u>
Save the changes you made to the Udef.

If you have edited the Udef and clicked on OK below one of the 3 parts of the code, 2 buttons will appear on the top bar of the Udef: revert and save as Udef.

<u>revert</u>
Undo all changes to the code, and restore the plug-in to the last saved state.

<u>save as Udef</u>
A save dialog opens and you can save the Udef as a normal Udef. It will be used immediately for your Unit, and also show up in the Udefs window. If you save (or later put) it in the same folder as where your score file is located, it will be loaded automatically the next time you open your score. If you put it somewhere else, you will need to load it manually before opening the score. The Udef will get the same name as you give your file.

i/o window



The i/o window shows the audio connections between the units. Each unit (or UMap) that has audio inputs and/or outputs is shown in the list. Each 'in' and 'out' is connected to a certain bus. An input reads from it, and an output replaces the bus contents with the output sound from the unit. The signal flow in the buses is from the top to the bottom (head to tail) of the chain; if a unit outputs to a bus, the units below it can receive it. If a unit below replaces the output with something else, the units that follow after it will get that sound instead.

Some units may have 'mix' slider, which allows sound that was already in the bus to be mixed with the output of the unit. If the mix slider is set to zero, or if there is no mix slider, the 'out' will replace anything that was on the bus before the unit. In some units, the 'in' will block the bus, meaning that the signal that was in the bus before the 'in' is replaced by silence. This is mostly the case by units at the end of the chain, such as the wfs_panners. There is a fixed number of buses within a chain, with a maximum of 32.

The graphic interface tries to show as clearly as possible which signal goes where, by visualizing the buses as thick and thin lines, red dots for outputs and green dots for inputs. By following the thick lines you can see exactly which unit sends and receives from which. In many cases there is only a single bus in use, to and from which all the units in the chain read and write.

<u>in</u>

For each input (there can be multiple inputs on a single unit) a green label is shown, and next to it a slider with a green dot and a number box to set the bus to where the input is connected to.

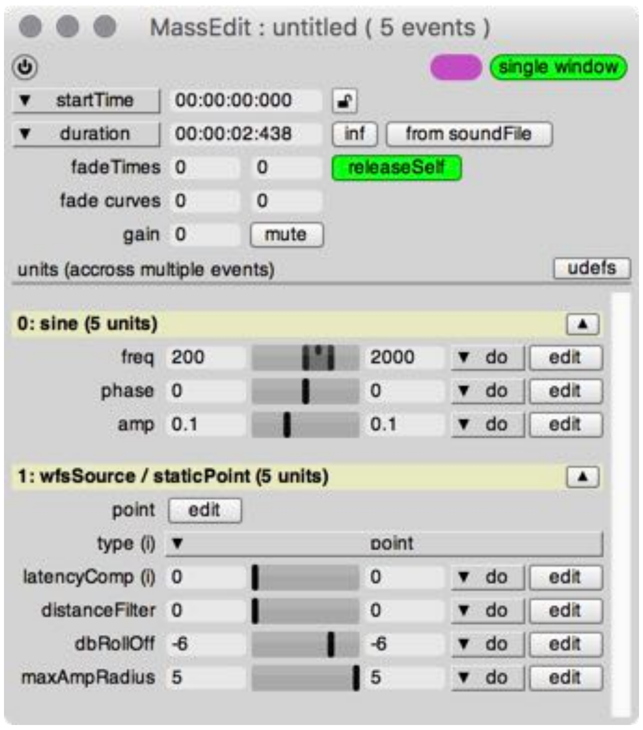
out

For each output (there can be multiple inputs on a single unit) a red label is shown, and next to it a slider with a red dot and a number box to set the bus to where the output is connected to.

mix

For units that support mixed output, a ‘mix’ slider is shown. It always applies to the ‘out’ that is immediately under it. When the slider is set to zero no signal is mixed in, and the unit behaves like a unit without mixed output. If the slider is opened, signal from previous units that output to the same bus is let through. This way multiple units can be easily mixed together in one bus.

Mass edit window



MassEdit : untitled (5 events)

startTime 00:00:00:000
duration 00:00:02:438
fadeTimes 0 0
fade curves 0 0
gain 0
units (accross multiple events)

0: sine (5 units)
freq 200 2000
phase 0 0
amp 0.1 0.1

1: wfsSource / staticPoint (5 units)
type (i) point
latencyComp (i) 0 0
distanceFilter 0 0
dbRollOff -6 -6
maxAmpRadius 5 5

When multiple Events are selected in the Score window and the (i) button is pressed (or the events double-clicked), the Mass edit window appears. In this window the common parameters of all selected events can be edited at once. The Mass editor intelligently scans through all UChains to see which Units are similar, and displays them together as one Unit. The interface elements of the Mass editor are the same as for the normal Chain window, but some of them work differently.

<u>power</u>

This button will play or stop all selected events together at the same time.

<u>color box</u>

This shows the display color of the event. If there are differences between colors of the selected Events, the color box will show the average color. Changing the color by clicking the box will give all selected events the same color

<u>single window</u>

If single window is selected and you double-click on an Event in the Score window, this will not open a new Chain window, but in an already existing Chain window.

Of single window is deselected, double-clicking on an Event in the Score window will open a new Chain window.

<u>startTime/startBar</u>

Shows the start time of the first selected Event. Changing the startTime will move all events together as a group. Modes:

‘startTime’: shows the startTime in HH:MM:SS:FFF (F = 1/1000 s)

‘startBar’: shows the startTime in bar/beat format: bar/beat/div (1/1000 beat)

<u>lock</u>
When enabled, the startTime of the selected events are locked to their current value (in seconds). This state is not saved with the score.

<u>duration/endTime/endBar</u>
Shows the average duration of the selected events. Changing this will add or subtract duration from all selected events. Modes:
'duration': shows the duration in HH:MM:SS:FFF (F=1/1000 s)
'endTime': shows the end time of the first event (startTime + duration)
'endBar': shows the end time in bar/beat format: bar/beat/div (1/1000 beat)

<u>inf</u>
If the majority of the selected Events has infinite duration, this will be highlighted. Setting it manually will make all events use infinite duration (or not).

<u>from soundFile</u>
Sets the length of the Events by the length of the longest soundfile in the chain. This is done for each of the selected Events separately. Events that don't have soundfiles will be ignored

<u>fadeTimes</u>
Start and end fade time in seconds. Changing this will affect all selected events.

<u>releaseSelf</u>
When releaseSelf is enabled, an event will have a fixed duration. It will take care of itself after the score playback engine has started it. This could also be regarded as "one shot" events. When releaseSelf is disabled, the score playback engine will take care of the releasing. This means, that if one would pause the score while this event is playing, it will keep playing until the score is resumed and passes the point where it was supposed to stop. This feature may be less reliable for very short events. Changing this will affect all selected events.

<u>gain</u>
In the MassEditor the average gain setting of the selected Events is shown. Changing it will add or subtract from each UChain individually, keeping the differences between them.

<u>mute</u>
mute will mute the level of the event. Contrary to the "disable" button in the score editor (loudspeaker icon) the event will still play, only it will not be audible. This means a sound can be muted and unmuted during playback, and the effect will be audible immediately. The mute switch is highlighted if more than half of the selected Events is muted. Changing it will mute or unmute all events together.

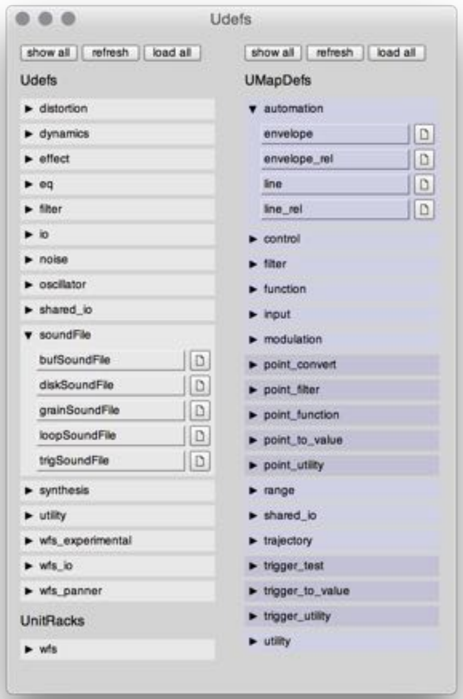
<u>udefs</u>
Click on Udefs at the bottom of the Toolbar to open the Udefs window. Form there you can drag Udefs to replace them in the various Units below the line. Replacing will change the Udef for all selected Chains that contain the previous one. It is not possible to change order, remove or add Units to the selected Chains via the Mass edit window.

Below the line in the parameters window you will find the "units" of that are in the selected UChains. For each Unit type it shows how many times it was found. If only one Unit is found of a certain type, it will show as a normal Unit editor (all parameters can be changed individually). If multiple Units are found of one type, you will see what we call Mass edit Units.

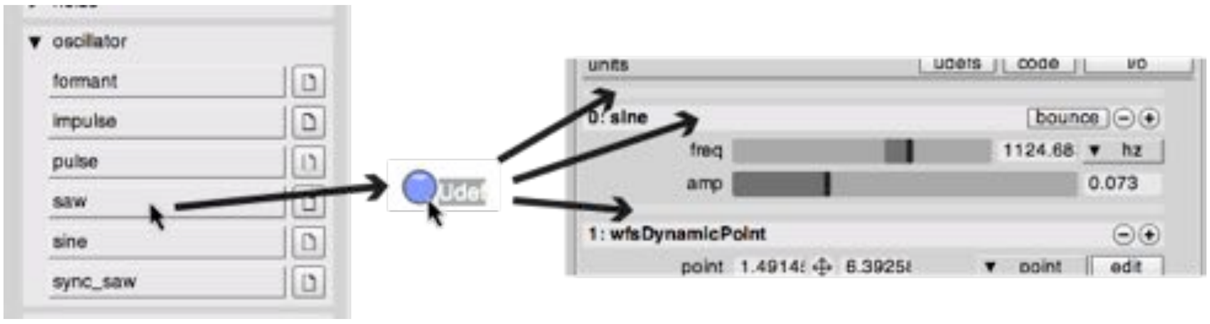
For most parameter types there are mass editing options. The most elaborate one is for single values that show up as sliders in the normal UChain window. In the mass edit window they become range sliders, which show the lowest and the highest setting found among the selected events. Changing these will scale the settings of all selected Units between these minimum and maximum values. There is also the do menu, which shows a number of array operations that can be performed on all parameters together. And there is the edit button, which brings up an editable plot window, where all parameters can be accessed individually or "drawn" together.

For WFS points there will be only an edit button, which brings up a window similar to the Path Editor, where all points can be edited together. For sound files, there will be three buttons. The list button brings up a text window with a list of all the file paths. The copy all button enables the user to copy all the sound files in the selected events to a single new location. It brings up a file save window to select the new location. There you will need to enter a file name, but the files will keep their own name. The folder button will bring up a file open dialog, where the user can select multiple soundfiles. They will be assigned consecutively to the units.

Udefs window

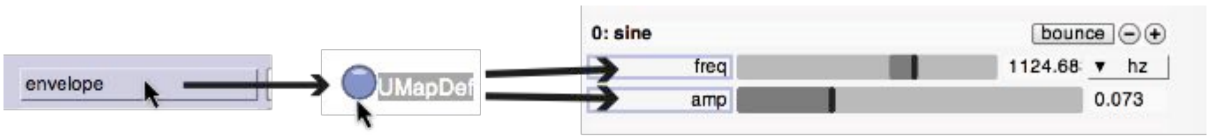


The Udefs window shows all currently loaded Udefs and UMapDefs. They can be dragged and dropped into the Chain or Mass edit window to replace or add Units and UMaps. Udefs can replace and Units in the Chain, by dragging them on the unit name or on the gray areas above and below it.



There are various categories of Udefs shown in the Udefs window. Categories such as ‘oscillator’ and ‘soundFile’ contain Udefs that generate sound, categories such as ‘effect’ and ‘filter’ have Udefs that process sound (typically placed after a sound-generating Unit) and categories such as ‘wfs_panner’ contain Udefs to output sound to the speakers. Every Chain should have at least one of the latter included, normally at the end (bottom) of the Chain, and one of the generative categories at the beginning (top).

The UMapDefs are shown in blue in the Udefs window. UMapDefs are used for modulating parameters (args) of Units. When dragging an UMapDef, the labels of all parameters that are able to use it will show a blue rectangle.



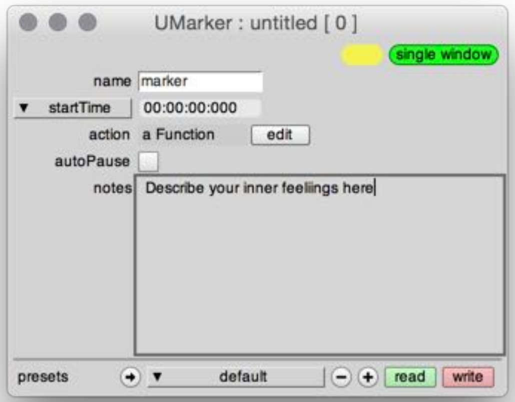
There are various categories of UMapDefs that apply to specific parameter types. For example the UMapDefs in the ‘point_xxx’ categories can only be used on Point parameters (such as the ‘point’ on a ‘wfsDynamicPoint’ Unit).

<u>show all/hide all</u>
Open or close all category ‘folders’ with Udefs or UMapDefs

<u>refresh</u>
Rebuilds the Udefs window.

<u>load all</u>
Loads all Udefs and UnitRacks or UMapDefs from the default directories.

Marker window



The Marker window is the editor for UMarkers. Markers are Events that can perform actions at specific times in a Score. They can also hold text notes, and always have a name.

<u>color box</u>
This shows the display color of the event. The default color for UMarkers is yellow. To change the color, click on this box to open the color editor. If a user-defined color is used, a small cross will show in the upper right corner of the box. Clicking that will remove the user-defined color, and reset to the default.

<u>single window</u>
If single window is selected and you double-click on an Event in the Score window, this will not open a new Chain or Marker window, but use an already existing window.
Of single window is deselected, double-clicking on an Event in the Score window will always open a new window.

<u>name</u>
Specify the name of the marker here. The name is also shown in the Score editor.

<u>startTime/startBar</u>
Shows the start time of the Marker event. Modes:
'startTime': shows the startTime in HH:MM:SS:FFF (F = 1/1000 s)
'startBar': shows the startTime in bar/beat format: bar/beat/div (1/1000 beat)

<u>action</u>
This shows the function to be executed at the start time of the Marker. By default, an empty function is here. The function is written in the SuperCollider language. The arguments passed to the function are 'marker' and 'score'. The marker is the Marker itself, the score is the Score where the marker is in. From there all Events in the Score and also the Score itself can be controlled.

<u>edit button</u>

The edit button opens up a window where you can edit the Marker action. To see some examples of functions that could be in here, select one of the presets at the bottom of the Marker window.

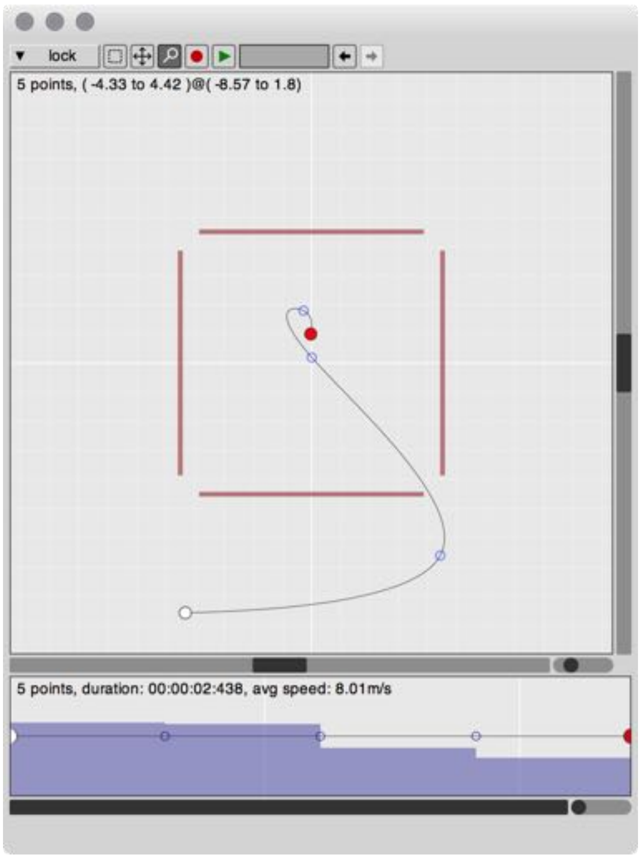
<u>autoPause</u>

If checked the marker will automatically pause the score time line when it passes. In WFSCollider pause doesn't stop the sound, it only stops the timeline. Any still sounding events will keep playing for their normal duration. Events with "releaseSelf" unchecked will keep on playing until the timeline is resumed and their end position is passed. This functionality can be useful for following a score played by an acoustic ensemble.

<u>notes</u>

You can type text into this large text box. This text will be saved with the score. It can be used to add remarks to your score.

Plot window for WFSPaths


lock
5 points, (-4.33 to 4.42)@(-8.57 to 1.8)
5 points, duration: 00:00:02:438, avg speed: 8.01m/s

The plot window shows a projection of the whole spatial trajectory in a path. The blue circles are the stored points of the path, and the gray line is the actual trajectory, including the spline curves. The white and red dots mark the beginning and end of the path. The red/brown lines show the position of the loudspeaker arrays. The white grid in the background shows meters, and the thick white lines cross in the center of the space (at coordinate (0,0)). The x axis goes from left to right (negative to positive), and the y axis from bottom to top (negative to positive), or i.e. from back to front. The path consists of a collection of points and transfer times between them. Also it holds information of the type of interpolation to be used in between the points (b-spline by default).

Tools in the toolbar

<u>Selector menu on the top left of the toolbar</u>

move
  Is equal to a translation of the selected points.

scale
  The selected points are scaled according to the distance of the center.

rotate
  The selected points are rotated around the center (with a fixed radius).

rotateS
  The selected points are rotated around the center and scaled according to the distance from the center.

elastic
  Moves all points (not only the selected ones) in an "elastic" way

twirl
  Same as elastic, but with polar coordinates, creating a "twirl" effect

chain
  Moves all points while keeping equal distance between them.

lock
  Locks all points so that they cannot be altered.

<u>Selection</u>

<u>Rubberband selection</u>

You can select point by clicking on a point. You can add points to a selection by shift-clicking on a point. With the rubberband selection you can select all the points that are inside the rubberband.

With alt-drag you can move the canvas.

<u>Move tool</u>

If you click in the background, you can move the canvas.

If you click on a selected point, what happens depends on the selector menu.

<u>Zoom tool</u>

With the zoom tool you can zoom in. Click drag to select the area. To see the whole movement, double-click on the background.

With alt-drag you can move the canvas.

In all of the above modes it is also possible to drag and drop paths in and from the window, with cmd-drag. Dragging a path from the window to an empty text window will convert it to SuperCollider source code. This code can also be dragged back into the editor window to replace the current path. It is also possible to drag paths between multiple path editors. In every case the path will be copied into the new editor to become a new unique object.

<u>Record tool</u>

With the record tool you can record your drawing in the Path Editor. Speed is recorded as well.

<u>Play tool</u>

Plays the path as a graphic animation/

Time slider
Move manually through the path.

Undo
Undo the last edit.

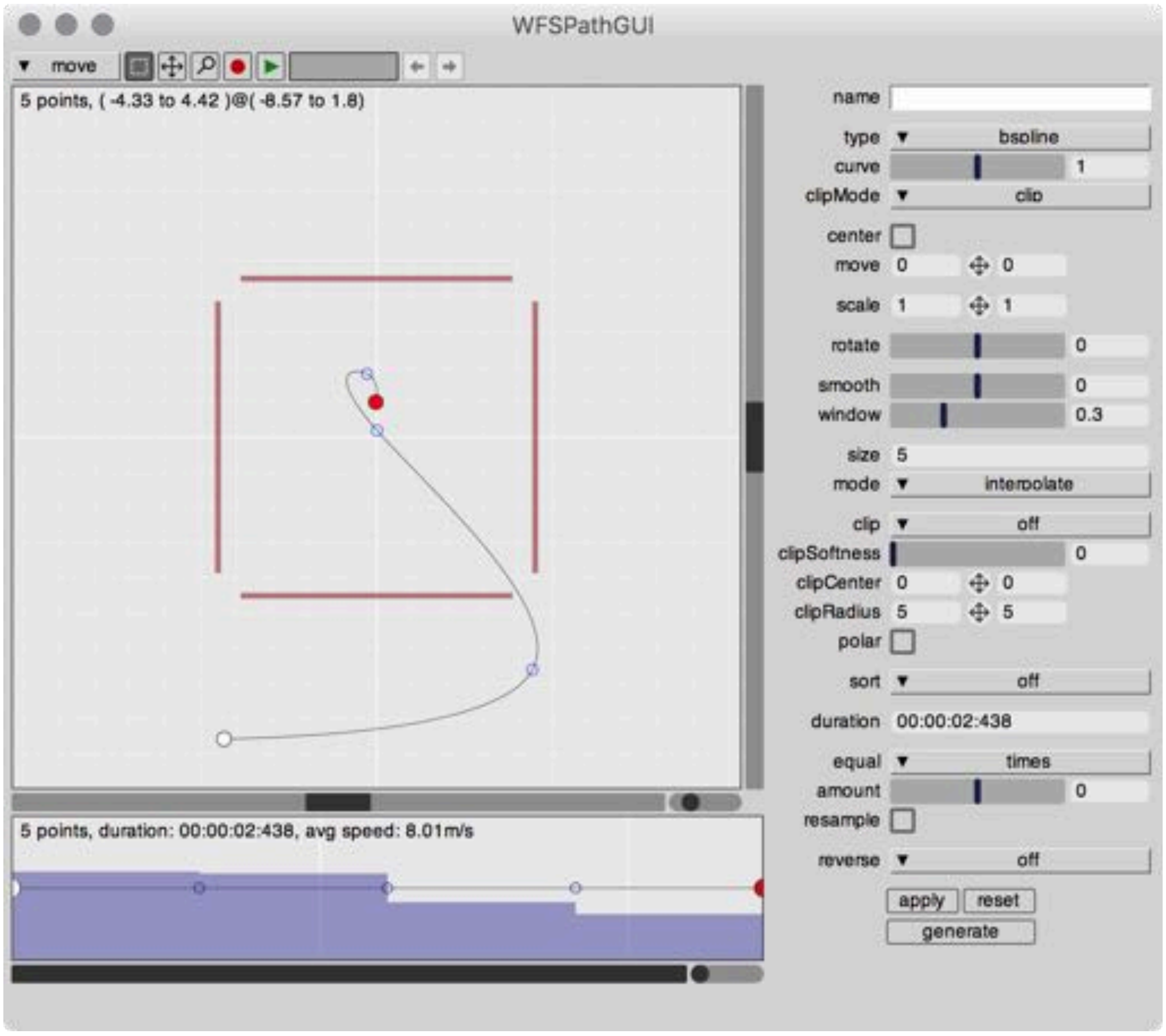
Redo
Redo the last edit you undid.

The Speed and Time display at the bottom of the window

The dots in the middle of this window correspond to the point on the path in the top window. In the window the points are set out on the time-axis. The green point is the first point on the path, the red point is the last point on the path.

If the speeds are high enough, in between two points you can see a blue bar that corresponds to the speed of the movement between two points. The higher the bar, the higher the speed.

Path Editor



5 points, (-4.33 to 4.42)@(-8.57 to 1.8)

5 points, duration: 00:00:02:438, avg speed: 8.01m/s

name
type bspline
curve 1
clipMode clip
center ☐
move 0 0
scale 1 1
rotate 0
smooth 0
window 0.3
size 5
mode interpolate
clip off
clipSoftness 0
clipCenter 0 0
clipRadius 5 5
polar ☐
sort off
duration 00:00:02:438
equal times
amount 0
resample ☐
reverse off

The path editor can be called up from any ‘trajectory’ UMap in the Chain editor, via the [edit] button. It holds a **plot window** (as described above) and a side bar for editing operations.

Tools on the right-hand side of the window

<u>name</u>
  Give the path a name. This name will only be visible in the path editor itself.

Type, curve and clipMode all have an effect on the curve of the path.

<u>type</u>
  bspline
  cubic
  linear
    Changes how the curve will move through the points.

curve

Has an effect if in the type-menu cubic or linear is chosen.

In linear mode the effect is not visible in the editor (the lines will remain straight), but will be audible.

A curve value lower than 1 will result in a lower speed near the points, and a value higher that 1 will result in higher speed near the points.

In bspline mode the curve setting will have no effect.

clipMode

clip
wrap
fold

Has an effect on the curve at the start and endpoints.

With wrap selected the curve acts as if the endpoint would continue at the startpoint.

With fold selected the curve acts as if the endpoint continues by following the curve backward (folding back).

With clip selected, the start and endpoints act as if there is no continuation.

Center and move both influence the position of the path:

center

Enabling this button will move the path or selected points to the center. If only one point is selected it will be moved to position (0,0), which means the 'move' parameters below can be used to enter the absolute position of the point. If multiple points or nothing is selected, the path will retain its shape, but be centered around it's mean position.

move

A translation of the selected points, or all points if none is selected.

By click-dragging on the move arrows in between the points, you can move as well.

scale

The selected points, or all points if none is selected, are scaled according to the distance of the center.

rotate

The selected points, or all points if none is selected, are rotated around the center of the space.

Smooth en window together define the smoothing of the whole curve.

smooth

Makes the corners in the curve smoother. Negative values exaggerate the corners.

window

Defines the length of the path that is taken into consideration when smoothing the curve. A value of 0 means no smoothing is done.

Size and mode together can be used to increase or decrease the number of points in the curve.

<u>size</u>
Defines the number of points in the curve. Changing this value will add or remove points according to the 'mode' setting below.

<u>mode</u>
interpolate
wrap
fold
Interpolate leaves the path as it is, and increases or decreases the number of points between the endpoints.
Wrap adds points to the curve, starting with the start point as the next point, and so on. It is in fact repeating the curve.
Fold continues the curve with the point before the last point, and so on. It is in fact reverse repeating, and after the start point is reached, the direction is changed to the forward direction again.

<u>clip</u>
limits the trajectory to the area specified by <u>clipCenter</u> and <u>clipRadius</u> in various ways
softClip: clip using a soft-clipping algorithm
sineClip: clip using a sine-wave based soft-clipping algorithm
fold: the points are mirrored against the clipping borders
wrap: the points are wrapped (modulo) around the clipping borders
excess: center/residue clipping; the clipping area is cut away in the middle
off (default): no clipping applied

<u>clipSoftness</u>
softening amount for clip (works for all modes)

<u>clipCenter, clipRadius</u>
the center and radius of the clipping area

<u>polar</u>
if checked, puts clip in 'polar' mode; i.e. using an ellipse to clip to rather than a rectangle.

<u>sort</u>
off
distance
angle
scramble
nearest
rel_angle
x
y
mean_xy
Sort the positions in the path according to setting. When 'off', no sorting is applied. 'distance' and 'angle' will sort the points using polar coordinates, 'x', 'y' and 'mean_xy' use cartesian coordinates. The 'nearest' and 'rel_angle' (relative angle) use a point-by-point sorting algorithm. 'scramble' will do the opposite of sorting; randomize the order of the points.

duration

Defines the duration of the path. Changing the duration is also reflected in the Chain window.

Equal, amount and resample can be used to smooth the time or speed differences between the different points on the path. The time and speed differences can also be seen in the time and speed window at the bottom of the Path Editor.

equal

times

speeds

Choose if you want to equalize time differences or speed differences.

amount

Drag it to the right: if set to 1, all times, or all speeds will be equal (depending on the setting of the equal menu). If you set it to -1 differences will be exaggerated (if there were differences to begin with...).

resample

If you have a path that has differing speeds, set equal to 'speeds', and amount to 1. You will see that resample will divide all points evenly over the path.

reverse

Reverses the order of the points in the path, or certain aspects of it

all: reverses positions and times
x: reverses only the x axis positions
y: reverses only the y axis positions
angles: reverses the angles (from center), keeps the distances and times
distances: reverses the distances (from center), keeps the angles and times
positions: reverses the positions, keeps the times
times: reverses the (delta) times, keeps the positions

apply

Apply all changes you have made in this window to the data of the path, and reset all controllers.

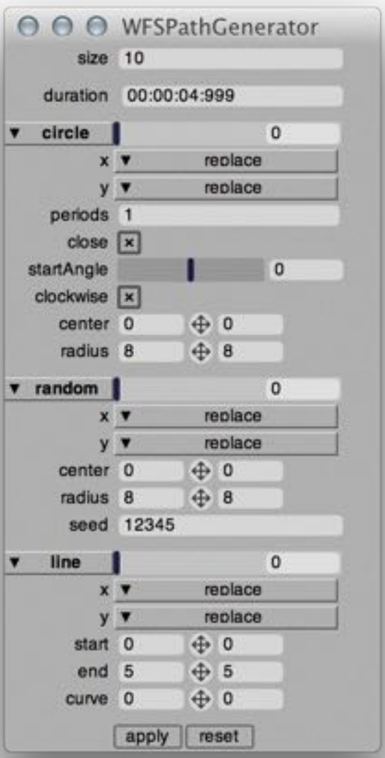
reset

Reset the path to the original data, and reset all controllers.

generate

Opens the Path Generator.

Path generator



WFSPathGenerator

size 10

duration 00:00:04:999

▼ circle 0

x ▼ replace

y ▼ replace

periods 1

close ☒

startAngle 0

clockwise ☒

center 0 0

radius 8 8

▼ random 0

x ▼ replace

y ▼ replace

center 0 0

radius 8 8

seed 12345

▼ line 0

x ▼ replace

y ▼ replace

start 0 0

end 5 5

curve 0 0

apply reset

<u>size</u>

Size reflects the size in the Path Editor. To start a completely fresh path with 2 points, set size to 2 and click on apply (at the bottom). If you do not click apply, the data of the path you started with will be recalled if you add points.

<u>duration</u>

Duration reflects the duration in the Path Editor.

There are 3 sections underneath this. Each of the 3 sections has the same possibilities. They provide different types of trajectory generation and adaptation.

<u>menu</u>

Sets the type of generator to one of the following types:

align
brown
circle
line
lineTime
lissajous
orbit

polygon
randTime
random
sine
spiral
spline

Each generator type is described in detail in the section below.

<u>amount slider</u>

With the amount slider you can set how much of the chosen process in the menu will be applied to the path. Beware: the slider has a default value of 0. If you do not change this value, nothing will happen if you change the settings.

<u>x menu</u>

This menu is available in all place processes, but not in time processes.

bypass
replace
+
-
*
min
max

Each of these choices defines the way the process will be applied to the x-axis.
bypass: noting will happen.
replace: replace the data of the path with the values of this process.
+: add the values of this process to the data of the path.
-: subtract the values of this process from the data of the path.
*: multiply the values of this process with the data of the path.
min: (is reversed) sets the maximum value of the data of the path.
max: (reversed) sets the minimum value of the data of the path.

<u>y menu</u>

This menu is available in all place processes, but not in time processes.

bypass
replace
+
-
*
min
max

Each of these choices defines the way the process will be applied to the y-axis.
bypass: nothing will happen.
replace: replace the data of the path with the values of this process.
+: add the values of this process to the data of the path.
-: subtract the values of this process from the data of the path.
*: multiply the values of this process with the data of the path.
min: (is reversed) sets the maximum value of the data of the path.
max: (reversed) sets the minimum value of the data of the path.

<u>time</u>

This menu is available in the time process, but not in the place processes.
bypass
replace
+
-
*
min
max

Each of these choices defines the way the process will be applied to the y-axis.
bypass: nothing will happen.
replace: replace the data of the path with the values of this process.
+: add the values of this process to the data of the path.
-: subtract the values of this process from the data of the path.
*: multiply the values of this process with the data of the path.
min: (is reversed) sets the maximum value of the data of the path.
max: (reversed) sets the minimum value of the data of the path.

The generator processes in the Path Generator

<u>align</u>

This is a position generator that alters the positions in the current trajectory.
It aligns all positions to a single line or circle by projecting them on it, or it equals the spacing between positons over an axis.

<u>direction</u>

In ‘line’, ‘spacing’ and ‘spacing_nearest’ mode, the direction slider sets the angle of the line. 0 means the x axis, 1 the y axis.

<u>mode</u>

line: project the positions on a straight line
circle: project the positions on a circle
spacing: equally distribute the positions along an axis (x or y or in between according to ‘direction’ slider)
spacing_nearest: same as spacing, but first makes sure the order of the positions remains the same as in the original

<u>brown</u>

This is a position generator.
It creates a brownian random path based on polar coordinates. Each point is calculated by adding a random angle and distance to the previous point.

<u>center</u>

The center that is used as a reference for the scaling.

<u>radius</u>

The amount of scaling.

<u>angleStep</u>

The maximum angle step per point

<u>step</u>

The maximum distance step per point

<u>seed</u>

Seed for the random generator. Change this to get another random distribution.

circle

This is a position generator.

If you want a circle, do not forget to set clipMode in the Path Editor to wrap, so the start and end point behave as if they are connected.

periods

1 period is 1 whole circle.

close

If you want to end the path at the point it started, check this box. If you forgot the direction of the closed path, press play in the Path Editor.

startAngle

The angle of the start point in degrees (-180 to 180)

clockwise

The direction of the path.

center

The center that is used as a reference for the scaling (= the center of the circle).

radius

The amount of scaling in the x an y direction (x and y radius...) of the circle.

line

This is a position generator.

start

Start point of the line.

end

End point of the line.

curve

The first value sets the curve to convex or concave, the second value defines the sharpness of the curve.

lineTime

This is a time generator, influencing only the times between the points. The total duration of the path will remain the same with any setting of this generator.

start

The relative duration of the first segment

end

The relative duration of the last segment

curve

This value defines a curvature in the interpolation between the first and last segment. A curve setting of ‘0’ means a linear interpolation, a positive curve makes the curve lean towards the start duration.

lissajous

This is a position generator.

This generator creates Lissajous curves (http://en.wikipedia.org/wiki/Lissajous_curve). This is done by combining different sinusoidal functions to the x and y axes.

periods

1 period means 1 cycle during the whole path. The period can be set for the x and y axes separately

close

If you want to end the path at the point it started, check this box.

startAngle

The phase of the sine wave, set for x and y axes separately in degrees (-180 to 180)

center

The center that is used as a reference for the scaling (= the center of the circle).

radius

The amount of scaling in the x an y direction (x and y radius...).

orbit

This is a position generator.

It creates an orbit around the center point, similar to the ones found on objects in outer space. In default settings the orbit generates a circle, but it becomes different with the ‘eccent’ and ‘eccentAngle’ settings.

periods

1 period means 1 cycle during the whole trajectory

close

If you want to end the trajectory at the point it started, check this box.

startAngle

The angle of the start point in degrees (-180 to 180)

clockwise

The direction of the trajectory.

center

The center point for the trajectory.

radius

The amount of scaling in the x an y direction (x and y radius...).

eccent

The amoint of orbital eccentricity

eccentAngle

The angle of the orbital eccentricity (-180 to 180 degrees)

polygon
This is a position generator.
It creates a polygon shape with a specified number of sides, regardless of the amount of positions in your trajectory.
sides
number of sides (2 or more). Doesn’t need to be a whole number.
orientation
If orientation is 0 (default), the upper side of the shape will be a corner. If -1 the shape will be rotated to the left so that the upper side of the shape aligns with the first side of the polygon. At +1 the same rotation is applied to the right.
offset
Sets there the starting point of the trajectory on the shape (0-1)
close
If you want to end the path at the point it started, check this box.
periods
1 period is 1 whole cycle over the shape
clockwise
The direction of the trajectory.
center
The center that is used as a reference for the scaling (= the center of the shape).
radius
The amount of scaling in the x an y direction (x and y radius...) of the shape.

randTime
This is a time generator.
min
The minimum time.
max
The maximum time.
seed
Seed for the random generator. Change this to get another random distribution.

random
This is a position generator.
center
The center that is used as a reference for the scaling.
radius
The amount of scaling.
seed
Seed for the random generator. Change this to get another random distribution.

sine
This is a position generator.
periods
Number of periods of the sine wave (between 0 and 2 pi).
phase
Starting phase of the sine wave.
start
Location of the start point.
end
Location of the end point.
amp
Amplitude of the sine.

spiral
This is a position generator. A spiral is a circular path that can start and end with different radius and center. The settings are similar to that of the circle generator

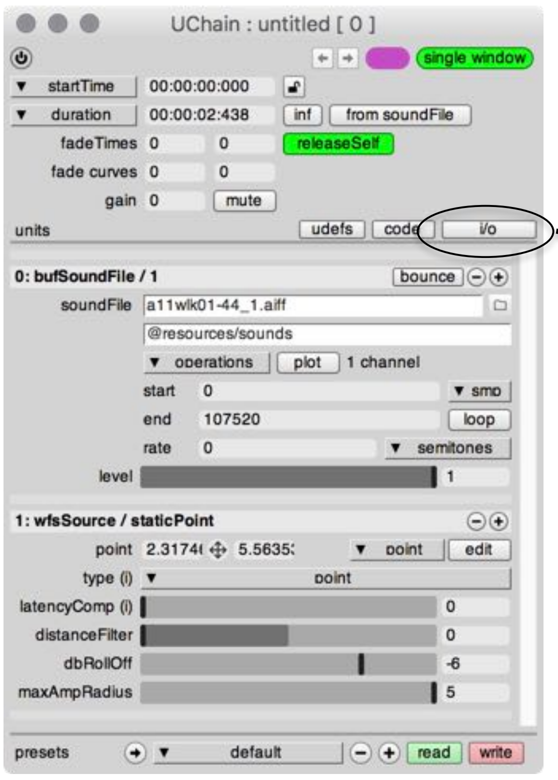
periods
1 period is 1 whole circle.
close
If you want to end the path at the point it started, check this box.
startAngle
The angle of the start point in degrees (-180 to 180)
clockwise
The direction of the path.
startCenter
The center coordinate (x,y) of the circle at the beginning of the path
startRadius
The radius of the circle at the beginning of the path. Different x and y values can create an elliptical circle.
endCenter
The center coordinate (x,y) of the circle at the end of the path
endRadius
The radius of the circle at the end of the path

spline
This is a position generator.
start
Set the start point.
end
Set the end point.
c1
The curve controller of the start point. (Try start (-5, -5), end (5,0) and c1 (0, -5), and next c1 (-5, 0), to get a feeling for what this means).
c2
The curve controller of the end point.

More about Units and UChains

Signal flow

The signal flow inside a UChain is organized in buses. Each bus has its own number (0-31) and is assumed to be empty at the beginning of the chain. In most common cases only one bus is used throughout the chain. For example, the default UChain (which is created when a new event is made in the Score Editor) contains a 'bufSoundFile' Unit and a 'wfsSource' Unit. The 'bufSoundFile' generates output to bus 0, and the 'wfsSource' reads from that bus. The signal flows downwards from the top of the UChain to the bottom. If the 'bufSoundFile' would not be placed *before* the 'wfsSource' Unit there would be no sound, because the 'wfsSource' in that case would read from the bus before the 'bufSoundFile' has written to it. This is the way things connect in a chain.

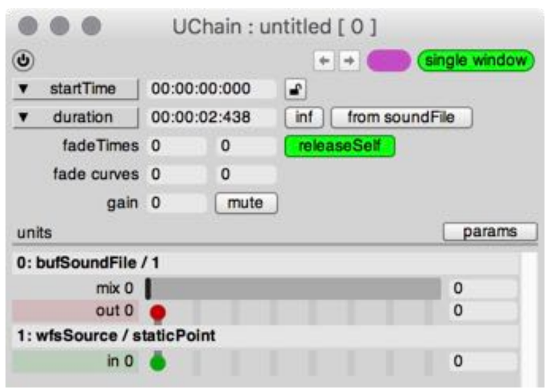

UChain : untitled [0]

startTime 00:00:00:000
duration 00:00:02:438
fadeTimes 0 0
fade curves 0 0
gain 0

0: bufSoundFile / 1
soundFile a11wlk01-44_1.aiff
@resources/sounds
1 channel
start 0
end 107520
rate 0
level 1

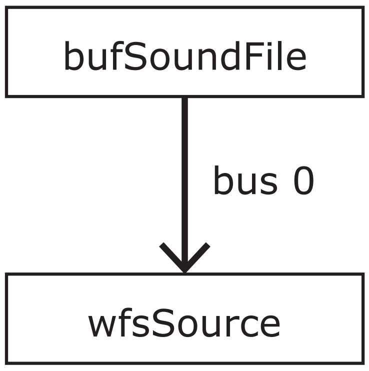
1: wfsSource / staticPoint
point 2.3174(5.5635:
type (i) point
latencyComp (i) 0
distanceFilter 0
dbRollOff -6
maxAmpRadius 5

presets default


UChain : untitled [0]

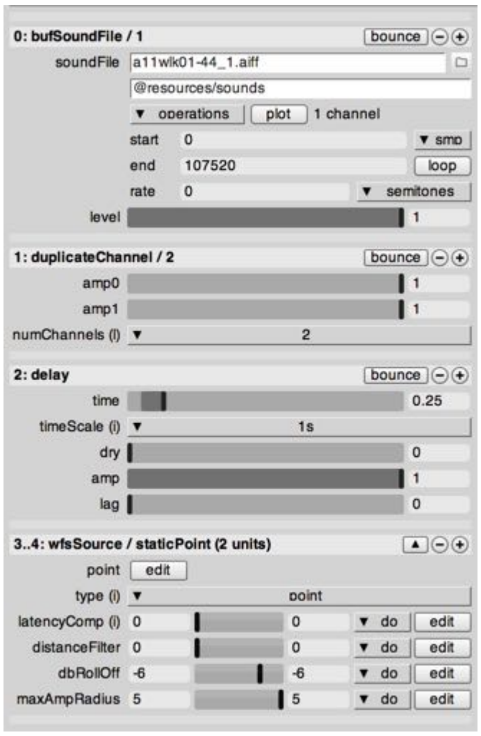
startTime 00:00:00:000
duration 00:00:02:438
fadeTimes 0 0
fade curves 0 0
gain 0

0: bufSoundFile / 1
mix 0 0
out 0 0
1: wfsSource / staticPoint
in 0 0



Things can however get more complex within a UChain. A maximum of 32 audio buses can be uses within a single UChain. This enables the use of multichannel audio files, but also situations where processed versions of a signal live alongside the original version, or where a single signal is distributed over multiple wfs sources. There are a number of Udefs that can help in this kind of situations. For example 'duplicateChannel', 'mixer' and 'splay' which can respectively add copies of signals to other buses, mix signals from multiple buses together and distribute an arbitrary number of buses over another number of buses. Also the '+' and '-' buttons enable easy multichannel expansion of units in the chain; with the '+' button a copy of the Unit is added and automatically set to output on the next bus number. The 'i/o' button sends the user to the bus number allocation panel, where the inputs and outputs of all units can be assigned to bus numbers. This may involve some clever planning, but is highly flexible. It all works as long as the output and input bus numbers correspond and the order of the units is correct. Every time a unit outputs to a bus, the previous contents of that bus are overwritten. Some units have a ‘mix’ slider with which the previous content of the bus can be mixed with the new content. Below is an example of a chain that creates a source with a soundfile and another source with a 0.25s delayed version of the soundfile.

UChain editor



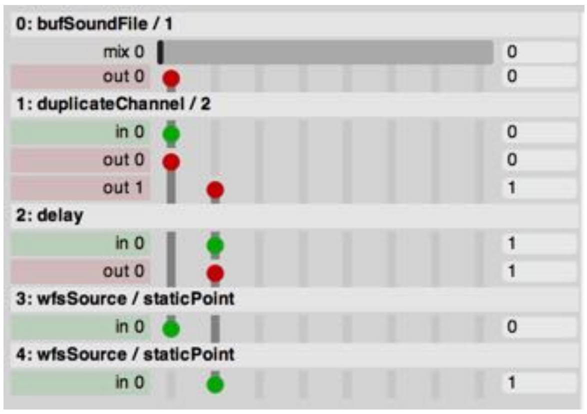
0: bufSoundFile / 1
soundFile a11wlk01-44_1.aiff
@resources/sounds
1 channel
start 0
end 107520
rate 0 semitones
level 1

1: duplicateChannel / 2
amp0 1
amp1 1
numChannels (i) 2

2: delay
time 0.25
timeScale (i) 1s
dry 0
amp 1
lag 0

3..4: wfsSource / staticPoint (2 units)
type (i) point
latencyComp (i) 0 0
distanceFilter 0 0
dbRollOff -6 -6
maxAmpRadius 5 5

UChain i/o editor



0: bufSoundFile / 1
mix 0 0
out 0 0

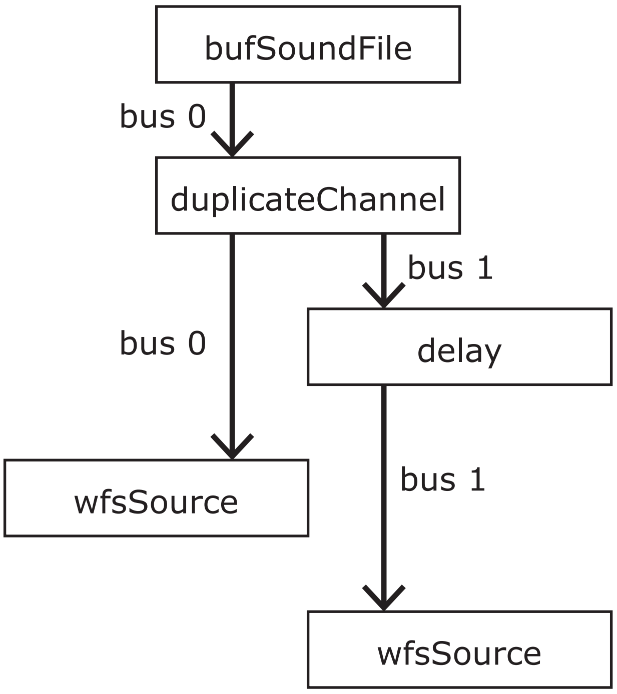
1: duplicateChannel / 2
in 0 0
out 0 0
out 1 1

2: delay
in 0 1
out 0 1

3: wfsSource / staticPoint
in 0 0

4: wfsSource / staticPoint
in 0 1

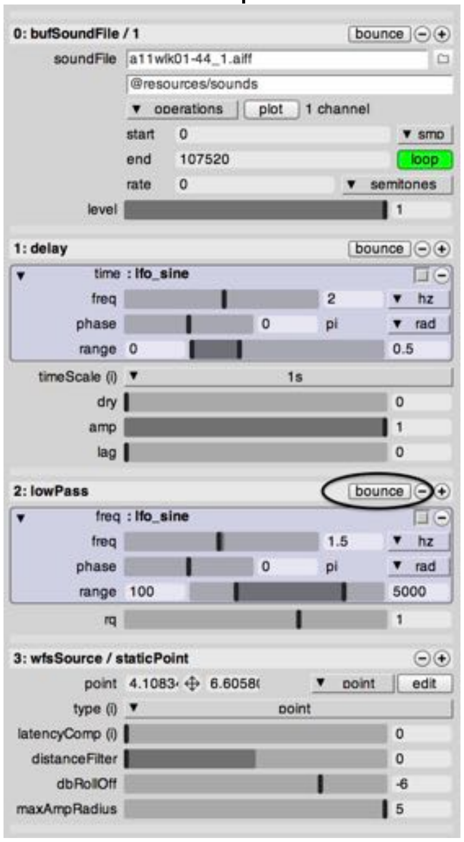
Signal flow



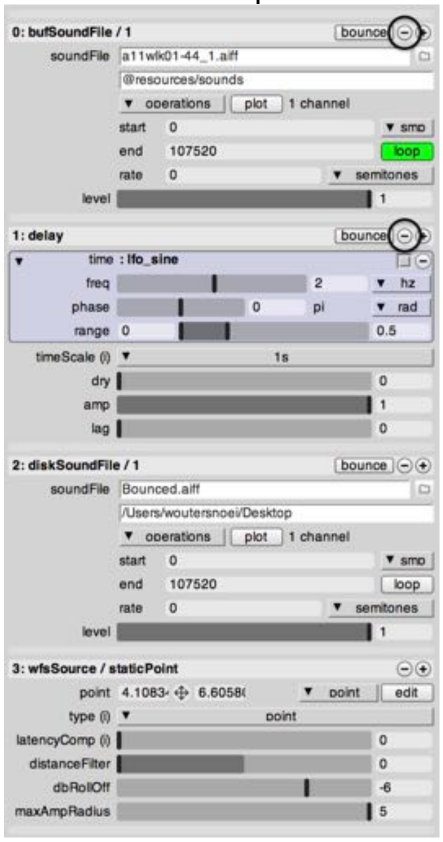
Bouncing

The ‘bounce’ button on most units can be used to mix part of the chain into a single audio file. This can be used for example to save cpu power, or if the chain gets needlessly complex. When the bounce is used WFSCollider use that unit and all units before it in the chain to create a mixdown. A file saving dialog shows up to allow the user to give a name and place for the soundfile (which should typically go in the same directory as your score file). The mixdown will happen in non-realtime and have as many channels as the bounced unit. After the mixdown is complete the unit itself will be replaced by a ‘diskSoundFile’ unit with the bounced file. In most cases it would make sense to remove all preceding units after this operation is completed. Below is an example of a bounce of a UChain with some filtering and delay modulation:

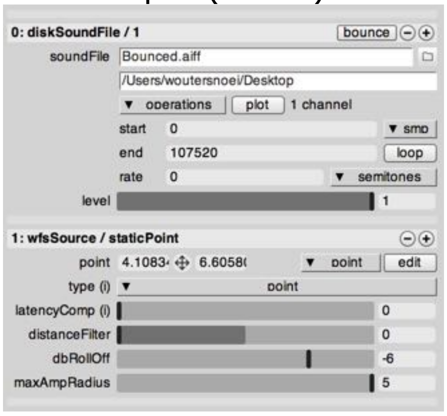
step 1


0: bufSoundFile / 1
soundFile a11wlk01-44_1.aiff
@resources/sounds
operations plot 1 channel
start 0 smp
end 107520 loop
rate 0 semitones
level 1
1: delay
time : lfo_sine
freq 2 hz
phase 0 pi rad
range 0 0.5
timeScale (i) 1s
dry 0
amp 1
lag 0
2: lowPass
freq : lfo_sine
freq 1.5 hz
phase 0 pi rad
range 100 5000
rq 1
3: wfsSource / staticPoint
point 4.1083 6.6058 point edit
type (i) point
latencyComp (i) 0
distanceFilter 0
dbRollOff -6
maxAmpRadius 5

step 2

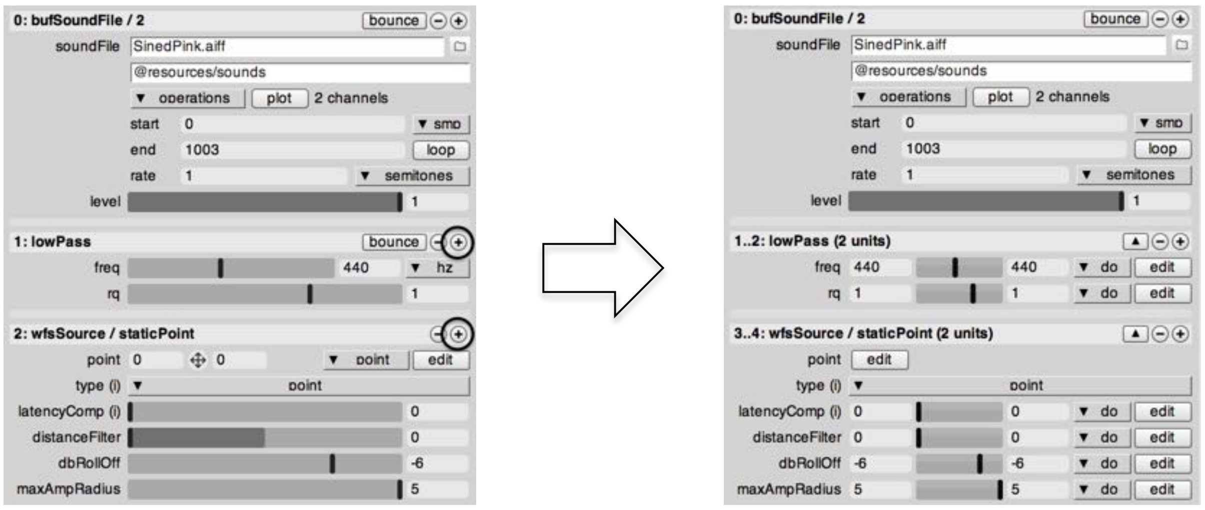

0: bufSoundFile / 1
soundFile a11wlk01-44_1.aiff
@resources/sounds
operations plot 1 channel
start 0 smp
end 107520 loop
rate 0 semitones
level 1
1: delay
time : lfo_sine
freq 2 hz
phase 0 pi rad
range 0 0.5
timeScale (i) 1s
dry 0
amp 1
lag 0
2: diskSoundFile / 1
soundFile Bounced.aiff
/Users/woutersnoei/Desktop
operations plot 1 channel
start 0 smp
end 107520 loop
rate 0 semitones
level 1
3: wfsSource / staticPoint
point 4.1083 6.6058 point edit
type (i) point
latencyComp (i) 0
distanceFilter 0
dbRollOff -6
maxAmpRadius 5

step 3 (result)


0: diskSoundFile / 1
soundFile Bounced.aiff
/Users/woutersnoei/Desktop
operations plot 1 channel
start 0 smp
end 107520 loop
rate 0 semitones
level 1
1: wfsSource / staticPoint
point 4.1083 6.6058 point edit
type (i) point
latencyComp (i) 0
distanceFilter 0
dbRollOff -6
maxAmpRadius 5

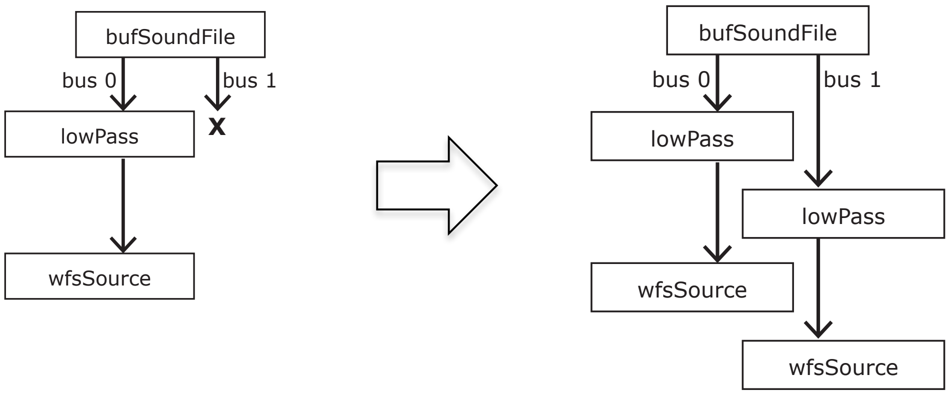
Multi-channel operation

In most cases a UChain will contain only one audio channel, leading to a single wfs_panner source. It is possible though to use multiple channels inside one chain. This can be useful for example to process stereo or multi-channel audio files, or to program inter-relations between multiple sources. There can be up to 32 audio channels in a single UChain. The example below shows how to create a two-channel UChain starting with a loaded stereo audio file (named SinedPink.aiff). The 'bufSoundFile' Unit (and all other soundfile related units) will automatically become multi-channel if a multi-channel soundfile is loaded. This is shown in the graphic interface, which now says "2 channels". The rest of the chain however is still single-channel; it will need to be made multichannel, and the user can fully decide what happens with each of the channels. In the example below we start out with a single-channel UChain, consisting of a soundfile, a low pass filter and a dynamic point source. They are all single-channel units, but as soon as the stereo soundfile is loaded the first unit becomes two-channel. In this case we want both channels to get their own low pass filter and point source. To accomplish this we press the (+) button on both the lowPass and wfsSource units. This creates a copy of each of the units, which are automatically assigned to the neighboring channel.


0: bufSoundFile / 2
soundFile SinedPink.aiff
@resources/sounds
operations | plot | 2 channels
start 0 | smo
end 1003 | loop
rate 1 | semitones
level 1
1: lowPass
freq 440 hz
rq 1
2: wfsSource / staticPoint
point 0 0 | point | edit
type (i) point
latencyComp (i) 0
distanceFilter 0
dbRollOff -6
maxAmpRadius 5

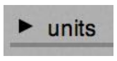
0: bufSoundFile / 2
soundFile SinedPink.aiff
@resources/sounds
operations | plot | 2 channels
start 0 | smo
end 1003 | loop
rate 1 | semitones
level 1
1..2: lowPass (2 units)
freq 440 440 | do | edit
rq 1 1 | do | edit
3..4: wfsSource / staticPoint (2 units)
point edit
type (i) point
latencyComp (i) 0 0 | do | edit
distanceFilter 0 0 | do | edit
dbRollOff -6 -6 | do | edit
maxAmpRadius 5 5 | do | edit

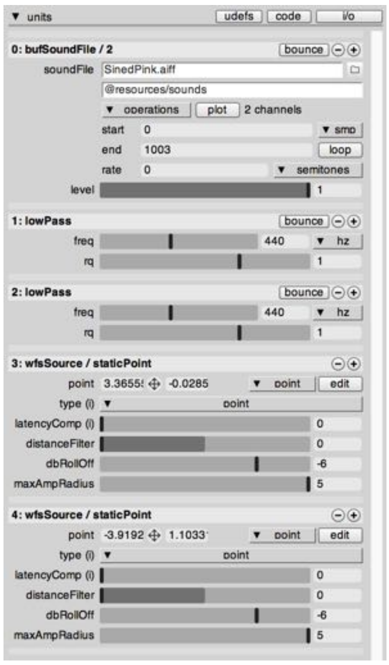
In the interface the lowPass and wfsDynamicPoint units now both state "(2 units)" in their headers, and the sliders and controls are now mass-edit controls (controlling the range of values rather then the individual ones. The signal changes as follows:



This means there is now a total of 5 (!) units in the chain. The individual units can be shown and edited in two ways:

- Press the [>] button next to “units” at the top of the UChain window. This will “unfold” the chain and show all units individually. You can now for example change one of the low-pass filters into a high-pass by dragging a highPass Udef from the Udefs window onto it. And you can edit the settings of each of the units individually
- Another way is to press a [^] button in one of the multi-units headers. This will open up a new window with the individual units in it.




units | udefs | code | i/o

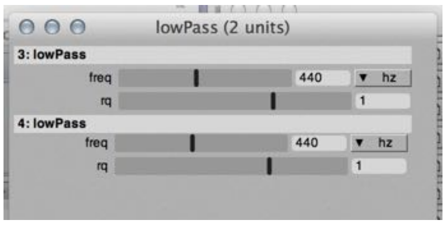
0: bufSoundFile / 2 | bounce
soundFile SinedPink.aiff
@resources/sounds
operations | plot | 2 channels
start 0 | smp
end 1003 | loop
rate 0 | semitones
level 1

1: lowPass | bounce
freq 440 | hz
rq 1

2: lowPass | bounce
freq 440 | hz
rq 1

3: wfsSource / staticPoint
point 3.3655 -0.0285 | point | edit
type (i) point
latencyComp (i) 0
distanceFilter 0
dbRollOff -6
maxAmpRadius 5

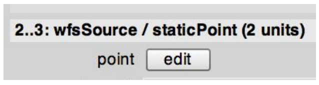
4: wfsSource / staticPoint
point -3.9192 1.1033 | point | edit
type (i) point
latencyComp (i) 0
distanceFilter 0
dbRollOff -6
maxAmpRadius 5

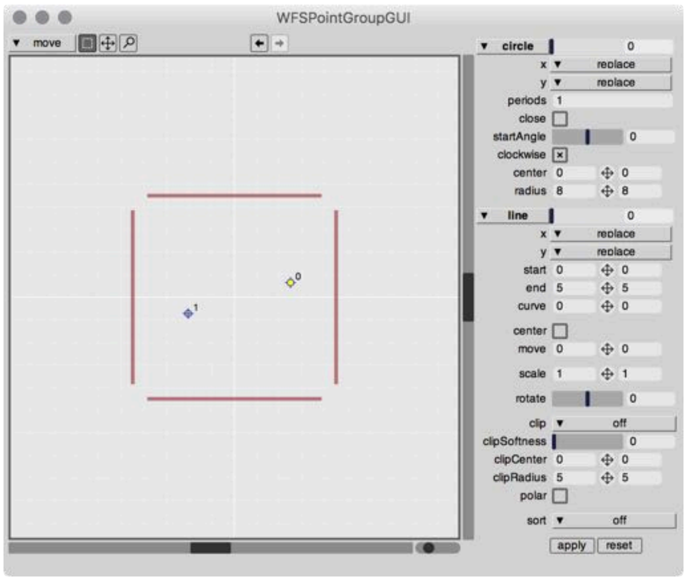

lowPass (2 units)

3: lowPass
freq 440 | hz
rq 1

4: lowPass
freq 440 | hz
rq 1

If you want to group-edit the positions of the sources you can also use the “edit” button in which appears at the ‘point’ parameter of point sources. This opens up the WFSPointGroup editor, where the points can be manipulated individually or as a group.


2..3: wfsSource / staticPoint (2 units)
point | edit


WFSPointGroupGUI

move

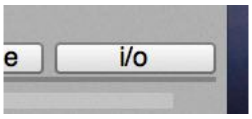
circle 0
x replace
y replace
periods 1
close
startAngle 0
clockwise ☒
center 0 0
radius 8 8

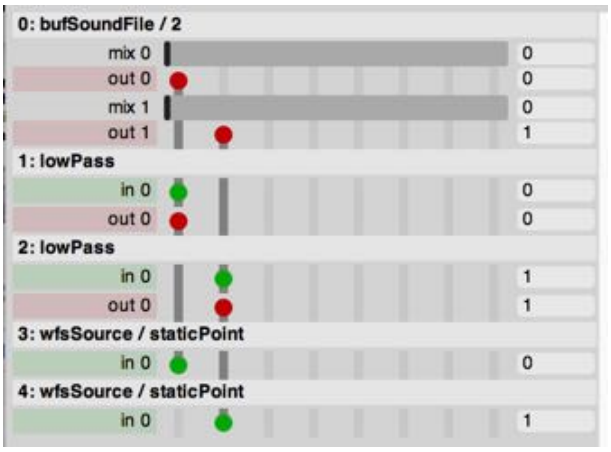
line 0
x replace
y replace
start 0 0
end 5 5
curve 0 0

center
move 0 0
scale 1 1
rotate 0
clip off
clipSoftness 0
clipCenter 0 0
clipRadius 5 5
polar
sort off

apply | reset

It is also possible to see and edit the signal flow of a (multi-channel) UChain. This is done via the i/o button. This brings up an editor in which the input and output buses (starting at bus number 0) can be assigned for each individual unit. In our example this window will look as follows:





0: bufSoundFile / 2
mix 0 0
out 0 0
mix 1 0
out 1 1
1: lowPass
in 0 0
out 0 0
2: lowPass
in 0 1
out 0 1
3: wfsSource / staticPoint
in 0 0
4: wfsSource / staticPoint
in 0 1

There are also a number of Udefs that can perform specific operations in multi-channel signal flows. These are:

- ‘duplicateChannel’ : can duplicate a single channel to multiple channels, with amplitude settings for each of them
- ‘mixer’: can mix multiple channels together into a single one
- ‘splay’: can mix or distribute multiple channels over another number of multiple channels, and manipulate the order etc.

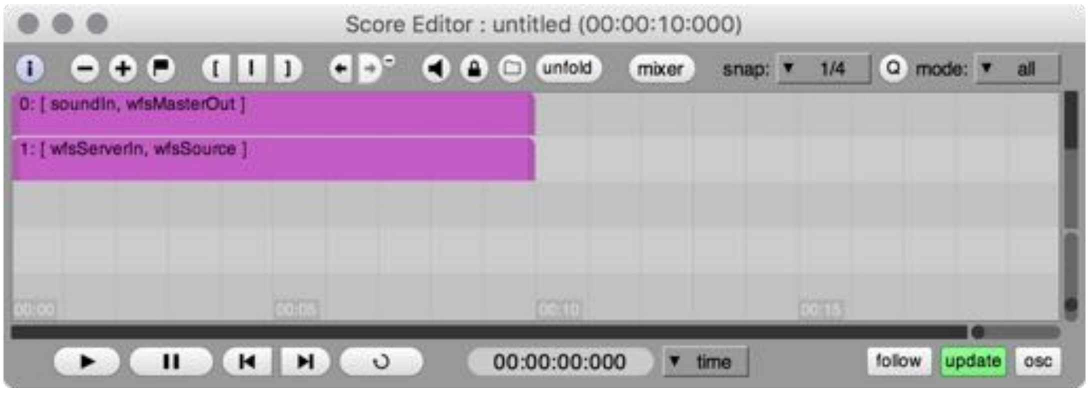
Also, a number of Udefs have a ‘numChannels’ parameter, by which the user can decide how many channels are inputted and/or outputted from a single Unit. Examples of this are ‘grainSoundFile’ and ‘gain’, and the multi-channel manipulation Udefs discussed above.

When using UMaps in multi-channel situations it can sometimes be wise to use shared values or points. For more information about that please refer to the next chapter; More about UMaps.

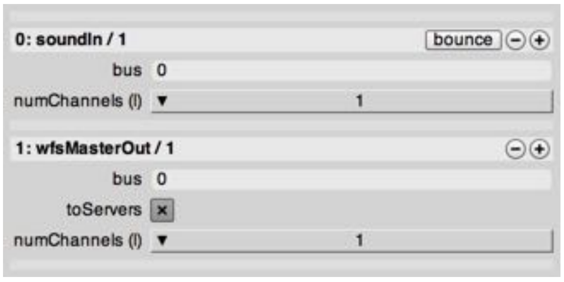
Live input

On single-computer installs of WFSCollider, getting live sound input into a UChain is quite easy. Just insert a ‘soundIn’ unit and choose the correct input bus (starting at 0). However, this method will not work on larger multi-computer setups such as the *Game Of Life WFS system (GOL)*. This section is about accomplishing live input on that particular system.

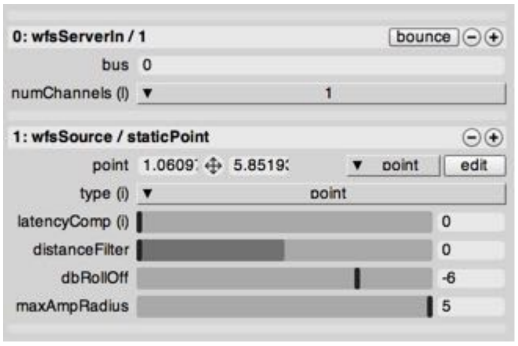
On the GOL system the sound inputs need to be distributed over the multiple rendering servers, via the ‘master’ computer. The master computer has its own sound card, with multiple analog inputs. From there a digital connection can route a maximum of 8 channels to the rendering servers (and from there to the speakers). Routing a signal from the master inputs to the server outputs requires the use of **two UChains** playing simultaneously. The first UChain will route the signal from the master computer inputs to the server inputs. The second UChain will get the input on the servers and do further processing and wfs rendering. In the score editor it would look something like this:


Score Editor : untitled (00:00:10:000)
unfold
mixer
snap: 1/4
mode: all
0: [soundIn, wfsMasterOut]
1: [wfsServerIn, wfsSource]
00:00:00:000
time
follow
update
osc

The first UChain needs to include at least a ‘soundIn’ and a ‘wfsMasterOut’ unit (from the ‘wfs_io’ category). These units force the UChain to be played on the master computer and take care of the first part of the routing. In between the ‘soundIn’ and ‘wfsMasterOut’ there can be other units for processing the incoming signal. On the ‘wfsMasterOut’ unit make sure the ‘toServers’ box is checked (if unchecked the signal will be sent to the master computer’s analog outputs instead). The second chain should start with a ‘wfsServerIn’ unit, to get the sound into the chain. In the example of a single live input channel sent to a static point without further processing on the GOL system, the settings of the two chains should look like the pictures below. The bus number (0-7) of ‘wfsMasterOut’ on the first chain should correspond with that of ‘wfsServerIn’ on the second chain. By using the ‘numChannels’ settings on the first chain it is possible to route multiple channels at once to the servers, which will be sent to subsequent bus numbers. Please note that there will always be some latency between input and output.


0: soundIn / 1
bus 0
numChannels (l) 1

1: wfsMasterOut / 1
bus 0
toServers ☒
numChannels (l) 1

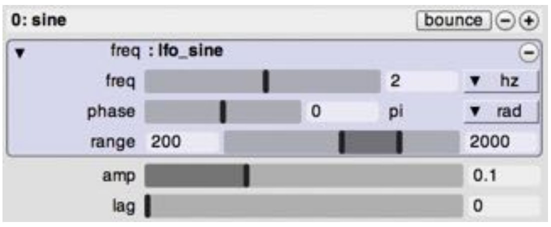

0: wfsServerIn / 1
bus 0
numChannels (l) 1

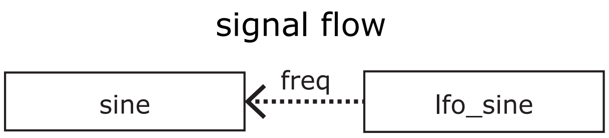
1: wfsSource / staticPoint
point 1.0609: 5.8519: point
type (i) point
latencyComp (i) 0
distanceFilter 0
dbRollOff -6
maxAmpRadius 5

More about UMaps

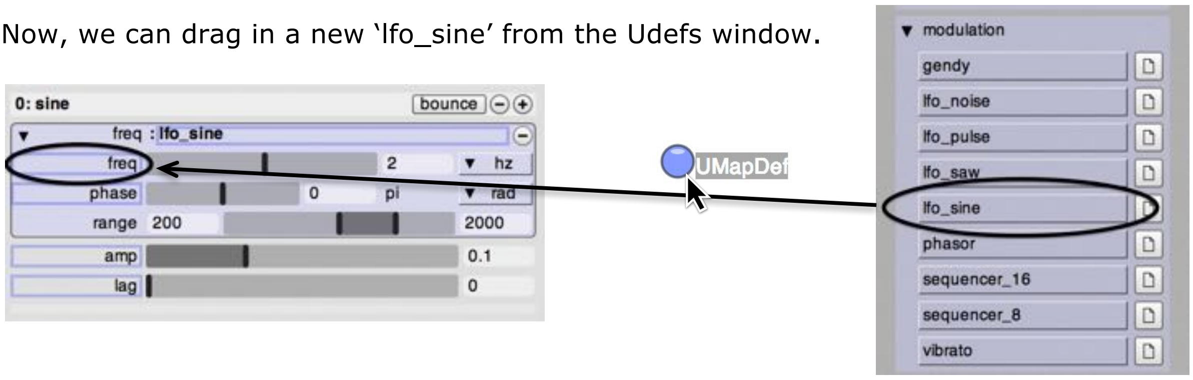
Nested UMaps

UMaps can be used to modulate, automate and control parameters of units. They can also modulate, automate and control parameters of ... UMaps. This is what we call nested UMaps. Each of the parameters of an UMap can itself be controlled by another UMap. This creates a possibly infinite chain of modulators and makes many kinds of advanced control signals possible. To do this simply drag a new UMapDef to the parameter you want to control. The following example shows a sine oscillator of which the frequency is controlled by an lfo sine.

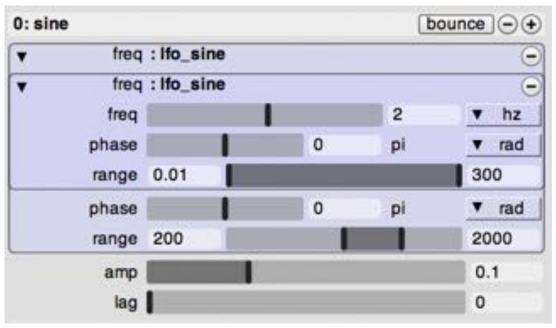


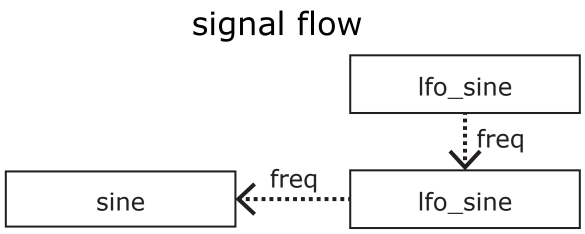


Now, we can drag in a new 'lfo_sine' from the Udefs window.

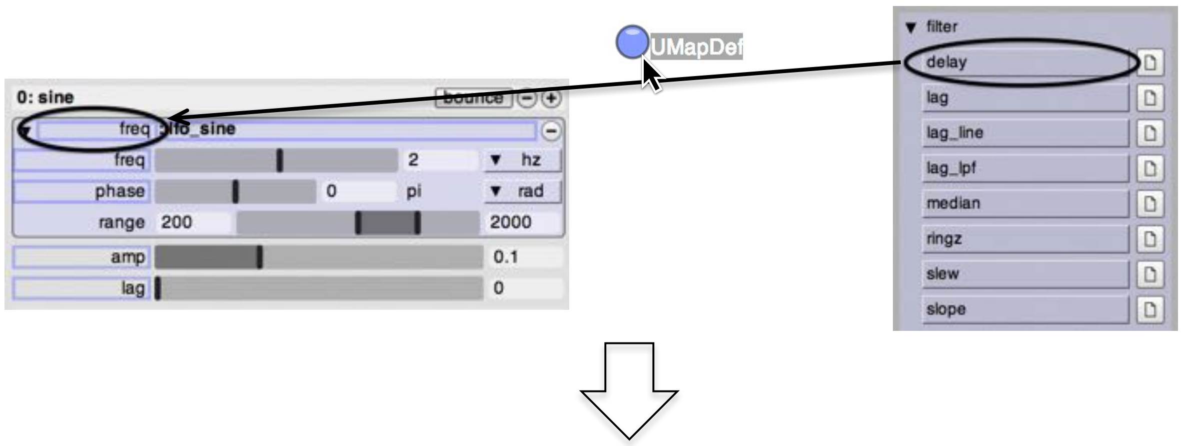


This creates a nested UMap controlling the frequency of the first 'lfo_sine', resulting in this case in a three-operator FM synthesis module (for those who know what that means).

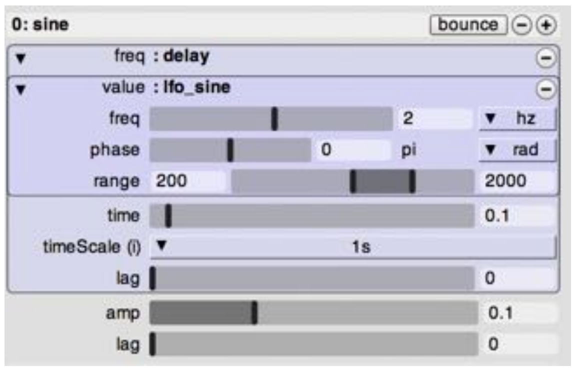




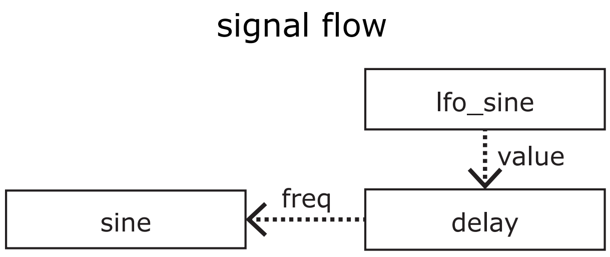
Some UMaps are constructed in such a way that they can also be *inserted* in the UMap signal path, rather than being appended as in the example above. These are typically the 'filter' UMaps, but also some of the 'utility' UMaps. Also the 'point_...' categories include many UMaps that can be used to insert. For UMapDefs that qualify for this an extra blue box shows up when dragging them from the Udefs window. The box is around the name of the parameter of an already existing UMap. If for example we would want to delay the output of an UMap, we could drag in a ‘delay’ UMap, to be placed in between the ‘lfo_sine’ and the unit.



0: sine
freq : lfo_sine
freq 2 hz
phase 0 pi rad
range 200 2000
amp 0.1
lag 0



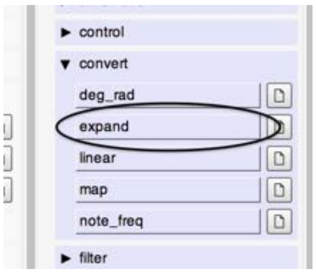
0: sine
freq : delay
value : lfo_sine
freq 2 hz
phase 0 pi rad
range 200 2000
time 0.1
timeScale (i) 1s
lag 0
amp 0.1
lag 0

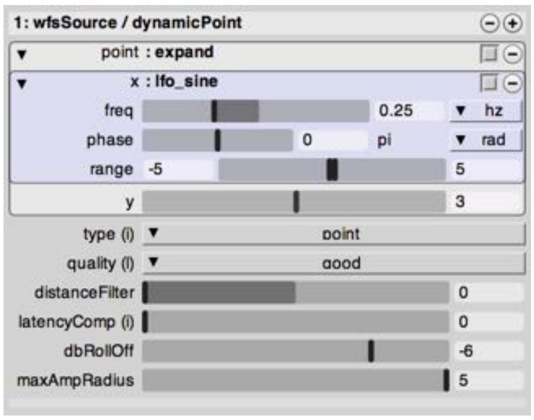


Pressing the (-) button on the ‘delay’ UMap in this case will remove the delay, but keep the ‘lfo_sine’.

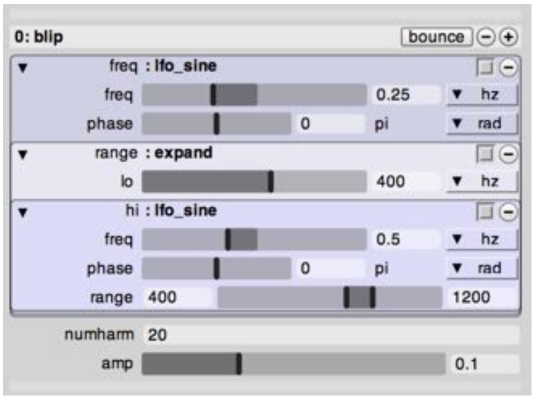
Expand

Some units have parameters with multiple values. For example points and ranges (two values), or eq settings (multiple values). If you want to control those values individually with UMaps, an ‘expand’ UMap (category: ‘convert’) needs to be inserted. ‘expand’ converts a number of supported multi-value parameters to named single values. Points will become ‘x’ and ‘y’, ranges ‘lo’ and ‘hi’. Eq parameters will be expanded to a named set of values. Each of these values can receive a single UMap. A few examples:

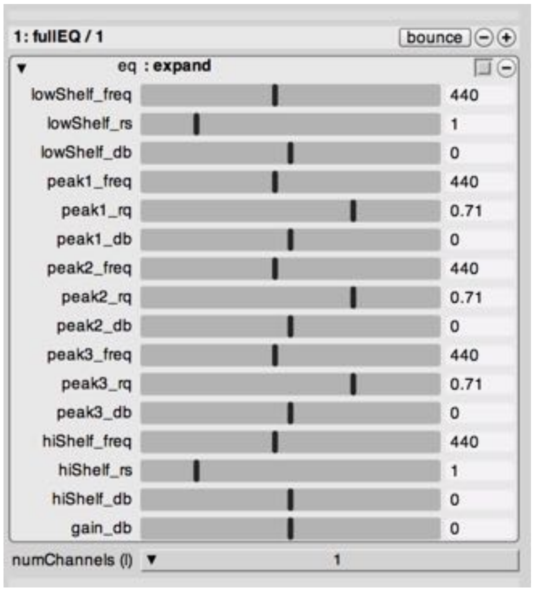




1: wfsSource / dynamicPoint
point : expand
x : lfo_sine
freq 0.25 hz
phase 0 pi rad
range -5 5
y 3
type (i) point
quality (i) good
distanceFilter 0
latencyComp (i) 0
dbRollOff -6
maxAmpRadius 5



0: blip
freq : lfo_sine
freq 0.25 hz
phase 0 pi rad
range : expand
lo 400 hz
hi : lfo_sine
freq 0.5 hz
phase 0 pi rad
range 400 1200
numharm 20
amp 0.1



1: fullEQ / 1
eq : expand

lowShelf_freq	440
lowShelf_rs	1
lowShelf_db	0
peak1_freq	440
peak1_rq	0.71
peak1_db	0
peak2_freq	440
peak2_rq	0.71
peak2_db	0
peak3_freq	440
peak3_rq	0.71
peak3_db	0
hiShelf_freq	440
hiShelf_rs	1
hiShelf_db	0
gain_db	0

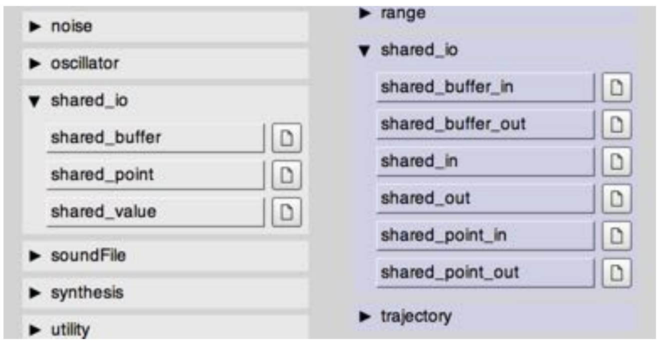
numChannels (i) 1

Specific ways of converting multi-value parameters to single or differently formatted values are also available, such as the ‘polar’ for points (category: ‘point_utility’) and various range options (category: ‘range’).

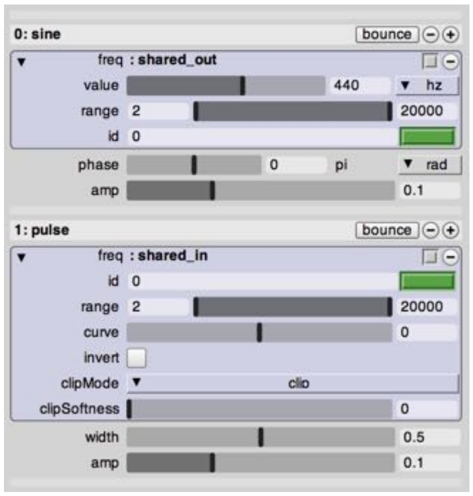
Shared values

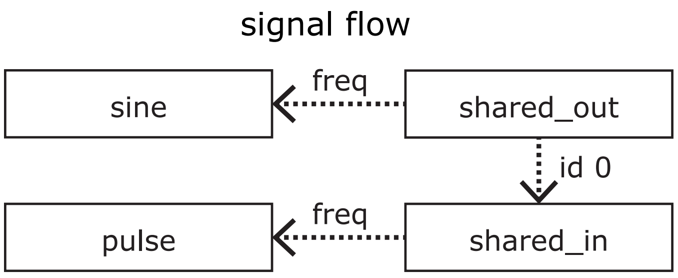
From version 2.2 and onwards WFSCollider supports a concept known as *shared values*. Shared values enable UMaps to share their output among multiple units or parameters.

A shared value can be created by enclosing your UMap into a ‘shared_out’ UMap (‘shared_io’ UMapDef category), or by creating a ‘shared_value’ unit (‘shared_io’ UDef category) earlier in the chain. The value is shared within the chain, and recognized by an ‘id’ number (0-99). A shared value can be retrieved by the ‘shared_in’ UMap, which typically needs to be further down the chain then the original ‘shared_out’ or ‘shared_value’. ‘shared_in’ offers extensive scaling options to make the value match the desired range.



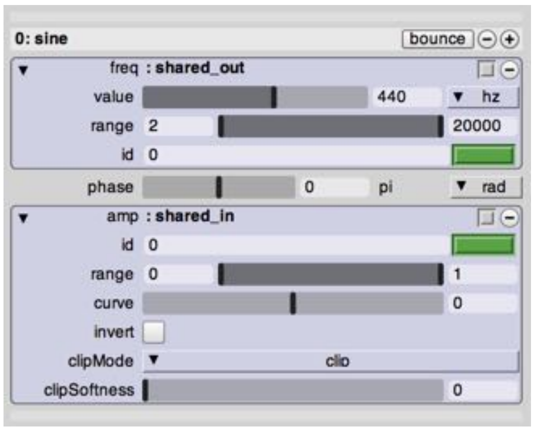
The signal flow of shared values can best be explained along a few examples. If we would have two oscillators, let’s say a ‘sine’ and a ‘pulse’, within one chain, and we would want let them have the same frequency, it could be by inserting a ‘shared_out’ on the frequency of the ‘sine’, and a ‘shared_in’ on that of the ‘pulse’. Now, when playing this chain the ‘value’ of the ‘shared_out’ will be the frequency of both units.

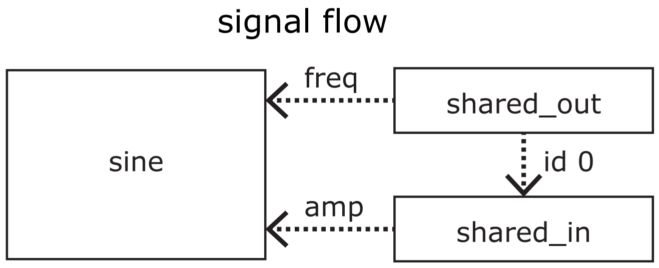

0: sine bounce
freq : shared_out
value 440 hz
range 2 20000
id 0
phase 0 pi rad
amp 0.1
1: pulse bounce
freq : shared_in
id 0
range 2 20000
curve 0
invert
clipMode clip
clipSoftness 0
width 0.5
amp 0.1



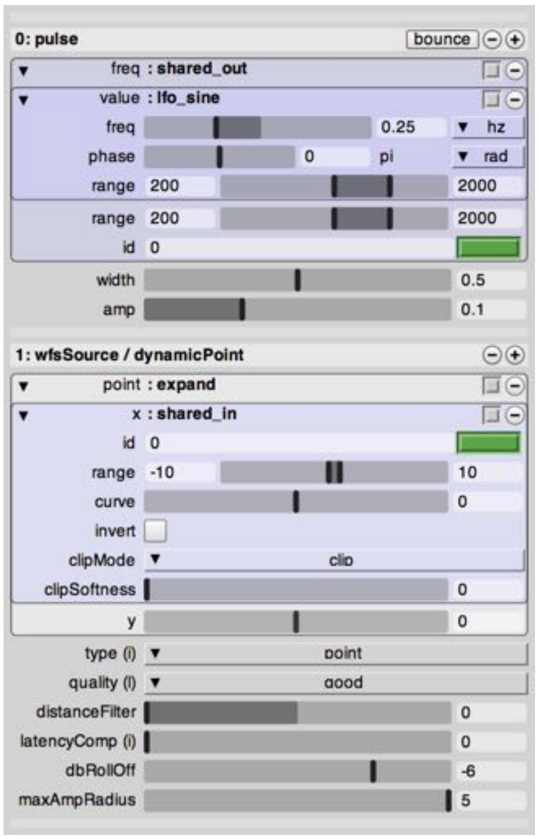
The range sliders on both shared_out and shared_in can be used to map the value to your liking. For example, if I would set the range of ‘shared_out’ to 2-10000, and that of ‘shared_in’ to 4-20000, the frequency of the pulse would effectively become 1 octave higher than that of the sine (note that the parameter ranges of all ‘freq’ parameters are on an exponential scale). The ‘clipMode’ parameter can be used to decide what to do with out-of-range values, and ‘curve’ and ‘invert’ can be used to modify the mapping relationship even further.

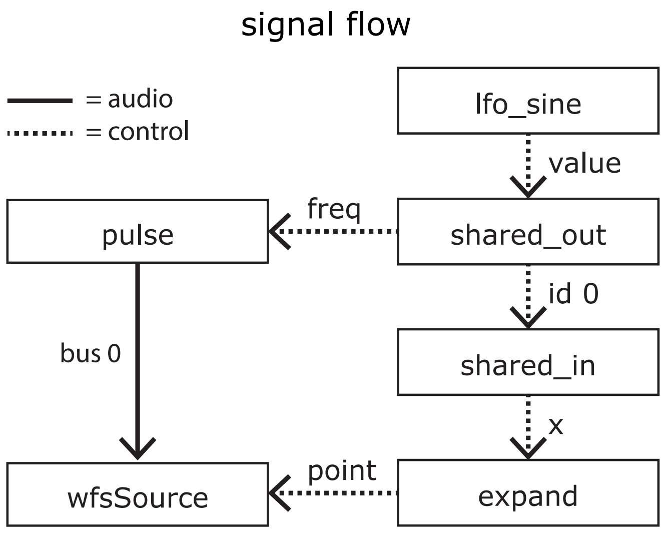
Another thing that could be done is coupling the ‘shared_in’ to another parameter with a different range. This may happen even within a single unit, as long as the ‘shared_out’ comes before the ‘shared_in’. In that case the set range of the ‘shared_in’ is mapped to that of the ‘shared_out’. If we would do this for example with the ‘amp’ of a ‘sine’ unit, it creates a direct relationship between the frequency and the amplitude of the sine. With the default settings this would result in a sine wave that gets louder when it’s frequency rises; where a 2Hz freq corresponds with a 0.0 amp, and a 20kHz freq with a 1.0 amp.


0: sine
freq : shared_out
value 440 hz
range 2 20000
id 0
phase 0 pi rad
amp : shared_in
id 0
range 0 1
curve 0
invert
clipMode clip
clipSoftness 0

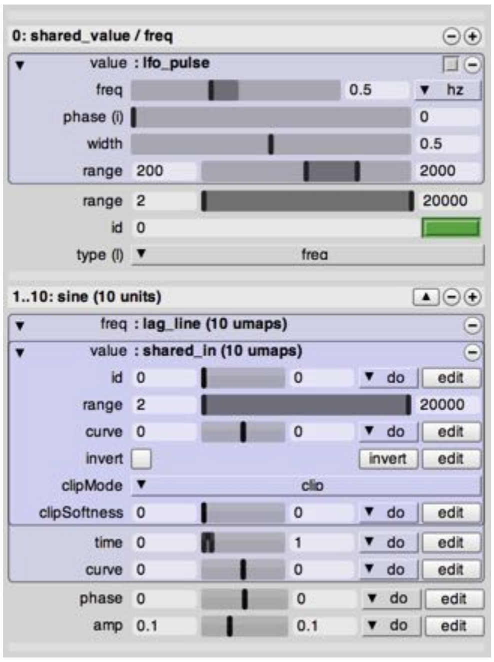


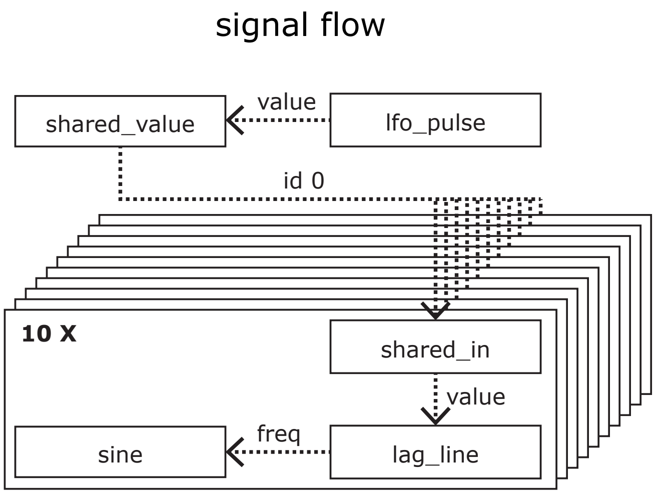
So a shared value can be converted from any range to any other range. It can also be used as an input of UMaps, and it’s value itself can also be an UMap. The following example shows a pulse oscillator on a point source, of which the frequency is modulated by a lfo_sine which at the same time controls the ‘x’ position of the source.


0: pulse
freq : shared_out
value : lfo_sine
freq 0.25 hz
phase 0 pi rad
range 200 2000
range 200 2000
id 0
width 0.5
amp 0.1
1: wfsSource / dynamicPoint
point : expand
x : shared_in
id 0
range -10 10
curve 0
invert
clipMode clip
clipSoftness 0
y 0
type (i) point
quality (i) good
distanceFilter 0
latencyComp (i) 0
dbRollOff -6
maxAmpRadius 5



The second way of creating a shared value is via the 'shared_value' Udef. This allows a dedicated unit in the UChain which only purpose is to create a shared value. The unit has no audio inputs or outputs. A typical use for this unit is when the shared value is used by many units further in the chain, where it could be convenient to have the source value somewhere easy to access. A 'shared_value' Udef can switch between various types of values, each matching a commonly used value range. This is merely for convenience, as the 'shared_in' UMaps further in the chain will always map the value to the desired range for the parameter they are attached to. An example of a use case of 'shared_value' would be the following, where one frequency value is used to drive 10 sine oscillators, each of them with a different lag time (smoothing time for value change).

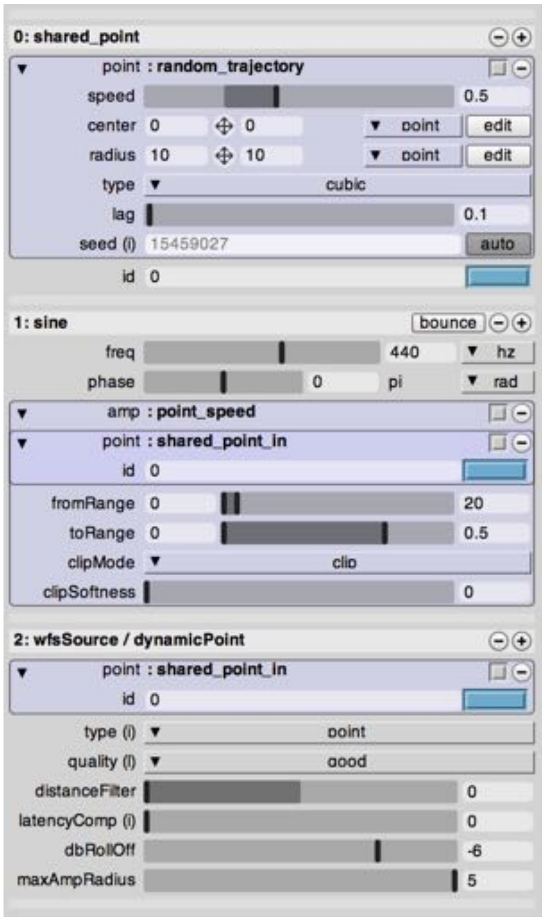

```
0: shared_value / freq
  value : lfo_pulse
    freq          0.5     hz
    phase (i)     0
    width         0.5
    range   200           2000
  range   2               20000
  id      0
  type (i)        freq
1..10: sine (10 units)
  freq : lag_line (10 umaps)
    value : shared_in (10 umaps)
      id            0     0       do  edit
      range         2     20000
      curve         0     0       do  edit
      invert                  invert  edit
      clipMode            clip
      clipSoftness  0     0       do  edit
    time            0     1       do  edit
    curve           0     0       do  edit
  phase             0     0       do  edit
  amp               0.1   0.1     do  edit
```



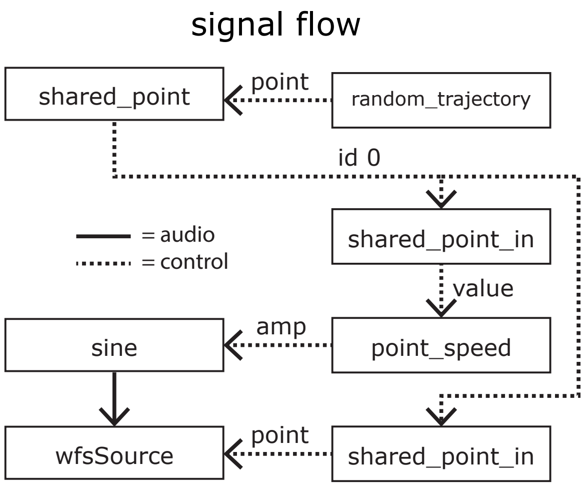
Shared points

A similar system exists for *shared points*. They have their own 'id' system (also 0-99), which doesn't overlap with the shared value id's. Shared points can be created with the 'shared_point_out' UMap ('shared_io' UMapDef category) or a 'shared_point' unit ('shared_io' Udef category), and can be retrieved with the 'shared_point_in' UMap.

Shared points work the same way as shared values. A point can be shared over multiple units, and there is a special category of UMapDefs for converting these points to single values. The example below creates a random trajectory for a wfs point source, and at the same time measures the speed of the movement to modulate the amplitude of the source signal. In this case the sound gets louder when it is moving faster.



```
0: shared_point
▼       point : random_trajectory
        speed                          0.5
        center  0   ✥ 0        ▼ point   edit
        radius  10  ✥ 10       ▼ point   edit
        type    ▼        cubic
        lag                            0.1
        seed (i) 15459027              auto
        id      0
1: sine                                bounce
        freq                  440      ▼ hz
        phase            0    pi       ▼ rad
▼       amp : point_speed
▼       point : shared_point_in
        id      0
        fromRange  0                   20
        toRange    0                   0.5
        clipMode   ▼      clip
        clipSoftness                   0
2: wfsSource / dynamicPoint
▼       point : shared_point_in
        id      0
        type (i)     ▼    point
        quality (i)  ▼    good
        distanceFilter                 0
        latencyComp (i)                0
        dbRollOff                      -6
        maxAmpRadius                   5
```



Shared points can be used for many ways of coupling the position of sources to sound parameter changes. They can also be used to manipulate positions of multiple sources within a chain, via various UMaps from the ‘point_filter’ and ‘point_utility’ categories.

Appendix A:
Udef and UMapDef reference

Udefs

Udefs are definition files for Units. They can be found in the Udefs window, and dragged into the Chain window from there.

distortion

bitCrusher

Introduces quantization noise to a signal by downsampling and bit-rate rounding
rate: fraction of the original sampling rate. If fs == 44100; rate == 0.25 means fs becomes 11025
bits: virtual number of bits (1-24), can be fractional

clip

The clip Udef can clip the incoming signal in various ways, including a softness area where the signal is distorted to create less harsh clipping curves.
scale: a scaling ratio applied to the input signal
offset: a DC offset added to the scaled input signal
clip: clip amplitude level (0-1)
clipMode: the type of clipping applied (0-3)
- 0: softClip; normal clipping with softclip area
- 1: sineClip; normal clipping with a sinusoidial/s-curve softclip area
- 2: fold; the signal is mirrored between the clipping borders
- 3: wrap; the signal is wrapped around (modulo / %) the clipping borders
- 4: excess; inverse/center cipping - only the signal outside the clipping area is kept

clipSoftness: amount of softening (0-1). This can be applied to any of the modes. Mode 0 (softClip) applies softening through non-linear distortion. Modes 1-4 apply softening through a sine function
makeUp: amount of automatic make-up gain added to compensate for the lost signal amplitude (0-1)
cutoff: frequency of a 2-pole high-cut filter applied after the clipping stage

tanhDistort

Distorts the sound via a hyperbolic tangent function.
inGain: input gain (dB).
outGain: output gain (dB).

dynamics

limiter

A lookahead transparent limiter.
limit: in dB.
lookAhead: lookahead time in seconds. The signal will be delayed by this amount of time, to give the limiter the chance to eliminate peaks from it. Lower lookahead times will result in faster response, but also more side-effects. lookAhead is an 'init' parameter (recognized by the "(i)" in the UChain editor), which means changed values only become effective when the event is started the next time.

noiseGate

A noise gate.
thresh: threshold in dB
att: attack time
hold: hold time
rel: release time
hysteresis: hysteresis applied to threshold (0-1). A hysteresis value 1 (maximum) will make the gate never close agian after opening.

peakCompressor

A compressor that uses the peak values in the amplitude as reference.
thresh: in dB.
ratio: ratio of the reduced level to the original level.
knee: the size of the knee in dB over and under the threshold. 0 is equal to no knee.
att: attack. The time it takes to follow an attack.
rel: release. The time it takes to follow a decay.
makeUp: make-up gain.

rmsCompressor

An rms-compressor. The mean value is taken over a range of 40 samples.
thresh: in dB.
ratio: ratio of the reduced level to the original level.
knee: the size of the knee in dB over and under the threshold. 0 is equal to no knee.
att: attack. The time it takes to follow an attack.
rel: release. The time it takes to follow a decay.
makeUp: make-up gain.

effect

convolution

A realtime convolution udef, that uses partioned convolution, useful for creating reverbs. This udef requires soundfiles in a special format, usually with a .partconv extension. These files can be created via the gui interface in the 'operations' menu; convert ir file (converts an existing impulse response audio file) and generate danstowell (creates a new random impulse response based on an algorithm by Dan Stowell. Convolution currently supports only mono impulse responses. For multichannel use you need to create multiple convolution units in your chain.
file: a PartConvBuffer object; points to a .partconv file on disk.
wet: wet (processed sound) level
dry: dry level

delay

time: the length of the delay.
timeScale: the number of seconds by which 'time' is multiplied. This value can not be changed during playback.
dry: level of the dry signal.
amp: level of the delayed signal.
lag: smoothing time applied to change of delay time, use to prevent clicks during change of delay time.

freeverb

a simple reverb.
mix: wet/dry mix (0-1)
room: room size (0-1)
damp: damp (0-1) (0 means no damping)

freqShift

freqShift implements single sideband amplitude modulation, also known as frequency shifting, but not to be confused with pitch shifting. Frequency shifting moves all the components of a signal by a fixed amount but does not preserve the original harmonic relationships.
freq: the amount of shifting (Hz)
phase: phase of the freq shift (radians)

magFreeze

An FFT-based (spectral) Udef that is able to freeze the magnitude of an incoming audio signal. The freeze parameter (named 'hold') can be modulated. There is also the option to convolve the signal with noise, effectively randomizing the phase information.
fftSize: (512, 1024, 2048 or 4096) the window size of the FFT. Smaller windows result in faster response but more side effects
hold: (0-1/false-true) freeze magnitudes when true/1
amp: amplitude of output signal
noise: the amount of convolved noise signal
seed: random seed of the noise

pitchShift

A time domain granular pitch shifter. Grains have a triangular amplitude envelope and an overlap of 4:1.
semitones: the number of semitones to shift the pitch with.
pitchDisp: the maximum random deviation of the pitch.
timeDisp: a random offset from 0 to the number of seconds set with timeDisp that is added to the delay of the grain.

ringMod

A ringmodulator that modulates the soundfile with a sine oscillator, a pink noise generator and/or a second sound file.
sine: level of the sine oscillator.
freq: frequency of the sine oscillator.
noise: level of the pink noise generator.
input: level of the input on the 1st audio bus.
mix: balance between the original and ringmodulated sound.

tremolo

Applies a tremolo to the amplitude of the input sound.
speed: number of tremolo's per second.
amount: sets the minimum and maximum of the tremolo effect.
smooth: smoothen the modulator signal; 0 means square wave, 1 means sine wave, and everything in between.

eq

fullEQ

This is a filter with a lowshelf filter, 3 parametric filters and a hihgshelf filter.

Clicking the [edit] button in UChainGUI opens the EQ Edit window. You can set the filter by dragging in the window, setting the values in combination with the popup menu or choosing from the presets. The eq algorithm (EQdef) is the same as that of the global Unit lib EQ (UGlobalEQ), and it shares presets with it.

eq: an UEQ object, holding all settings for the eq.

numChannels: number of channels (*).

(*) only the following numbers of channels are allowed:
1,2,3,4,5,6,7,8,10,12,16,24,32

simpleEQ

This is a filter with a lowshelf filter, 1 parametric filter and a hihgshelf filter.

Clicking the [edit] button in UChainGUI opens the EQ Edit window. You can set the filter by dragging in the window, setting the values in combination with the popup menu or choosing from the presets.

lowMidHi: an UEQ object, holding all settings for the eq.

tiltEQ

Combined High Shelf and Low Shelf filter. Default settings mimic the Tonelux TILT eq response.

tilt: amount of tilt (in dBs). Positive numbers increase high frequencies.

center: center frequency (default 650)

rs: the reciprocal of S. Shell boost/cut slope for both shelving filters (default 2)

filter

allPass
An all-pass filter based on a cubic interpolated delay line.
freq: frequency.
decayTime: decay time in seconds
dry: level of dry (unprocessed) signal

bandPass
A band-pass filter
freq: center frequency.
bw: the bandwidth in octaves between -3 dB frequencies.

bandStop
A band-stop filter
freq: center frequency.
bw: the bandwidth in octaves between -3 dB frequencies.

combFilter
A comb filter.
freq: frequency.
decayTime: decay time in seconds
dry: level of dry (unprocessed) signal

cutFilter
This is a combination of a low cut and a high cut filter. Both filters are cascaded Butterworth filters, and order is settable between 2nd and 8th.
freq: cutoff frequency for a low cut and a high cut filter.
order: sets the order (steepness) of both filters. The orders are:
0: off
1: 2nd order (12dB/octave)
2 (default): 4th order (24dB/octave)
3: 6th order (36dB/octave)
4: 8th order (48dB/octave)
lag: smoothing time for the freq parameter.
mode: operating mode;
low_high (default): combination of low and high cut filters
low: only low cut
high: only high cut

formlet
An FOF-like formant filter.
freq: frequency.
attackTime: attack time in seconds
decayTime: decay time in seconds
dry: level of unprocessed sound

furseDistanceFilter
** if a furseDistanceFilter is used in combination with a wfsDynamicPoint or wfsStaticPoint in a chain, the next time the score is opened it will be removed and replaced by a 'distanceFilter' setting in on the panner unit **
The furseDistanceFilter is a filter that applies distance filtering according to a formula by Richard W. Furse (http://www.muse.demon.co.uk/vspace/model.html):
cutoff = 100000 / distance
where cutoff is Hz and distance is in metres.
point: a Point from which the distance to the center of the room (0,0) is determined
amount: strength of the effect. A cubed multiplier for the distance. 0 means no filtering, 1 means normal, 2 means the distance in the formula is multiplied by 8 (2**3).

highPass
A resonant high-pass filter
freq: cutoff frequency.
rq: the reciprocal of Q. bandwidth / cutoffFreq

highShelf
High Shelf filter
freq: cutoff frequency.
rs: the reciprocal of S. Shell boost/cut slope
db: boost/cut the center frequency (in dBs).

klank
A bank of resonator filters.
range: the minimum and maximum frequency
exponential: distribution of frequencies: 0: linear, 1: exponential
variation: amount of random variation. 0: no variation, 0.5: linear distribution, 1: full random.
amp: range of random amplitudes
ringTime: range of random ring times. Can not be modulated.
seed: random seed
n: number of oscillators

leakDC
Removes DC offset from signal.
cutFreq: frequencies below the cutFreq (Hz) will be reduced or removed. A lower cutFreq will make the DC offset removal less effective, a higher cutFreq will result in better offset removal but also in audible loss of bass frequencies in the sound

lowPass
A resonant low-pass filter
freq: cutoff frequency.
rq: the reciprocal of Q. bandwidth / cutoffFreq

lowShelf
Low Shelf filter
freq: cutoff frequency.
rs: the reciprocal of S. Shell boost/cut slope
db: boost/cut the center frequency (in dBs).

median
A median filter, effectively removing spikes from incoming values by calculating the median value of a number of samples (length).
length: length of the median window
amount: mix between original signal and filtered signal (0-1)
residue: (boolean) if true the unit will output the difference between the original input signal and the output signal.

moogVCF
A Robert Moog style low-pass filter, modulated with an LFO
freq: cutoff frequencies between which the modulation takes place.
modSpeed: speed of modulation (cycles per second).
modShape: shape of LFO; 0: square wave, 1: sine wave.
res: resonance of the filter (0-1)

peak
Parametric equalizer
freq: cutoff frequency.
rq: the reciprocal of Q. bandwidth / cutoffFreq
db: boost/cut the center frequency (in dBs).

io

output
Sends sounds from the previous units in the UChain to hardware output busses.
WFSCollider users: This sends only to the outputs of the master server. To send to individual speakers on the WFS system, use wfsIndex instead. To send output "the official way" only to the outputs of the master audio interface, use the wfsMasterOut Udef instead (and set "toServers" to false if you really want the analog outputs of the master interface)
bus: the channels are sent out starting with this bus.
numChannels: number of channels (*).

outputWithSubwoofer
Sends sounds from the previous units in the UChain to hardware output busses.
Send also the another output for subwoofer mix.
Note to WFSCollider users: This sends only to the outputs of the master server. To send to individual speakers on the WFS system, use wfsIndex instead. To send output "the official way" only to the outputs of the master audio interface, use the wfsMasterOut Udef instead (and set "toServers" to false if you really want the analog outputs of the master interface)
bus: the channels are sent out starting with this bus.
numChannels: number of channels (*).

roundPanOutput
Sends sounds from the previous units in the UChain to hardware output busses, using a panning algorithm for clockwise circular speaker setups.
bus: the channels are sent out starting with this bus.
point: the position of the source (0@0: center, positive x is left, positive y is front)
dbRollOff: number of dB's amplitude rolloff per distance doubling
speakerRadius: the distance of the virtual microphones from the center of the room, in meters. When 0, no inter-speaker delays are applied (only a distance-dependant doppler delay). Default value is 0.19, which is okay for normal speaker setups. Optimum value may differ per setup.
orientation: the speaker index of the front. 0.5 (default) means that the front position is in between the first and second speaker (normal for stereo and quad setups). This value can also be modulated to rotate the whole spatial image.
numChannels: number of speakers (*)

soundIn
Read sound from a hardware audio input.
WFSCollider users: This unit will only get sound input on the master server. To send the sound to the rendering servers, use a \wfsMasterOut unit, and a new UChain starting with a \wfsServerIn.
bus: the bus that you want to use to read audio from. Starts counting from 0; 0 means the first input of your audio device.
numChannels: number of channels (*).

(*) only the following numbers of channels are allowed:
1,2,3,4,5,6,7,8,10,12,16,24,32

stereoOutput
Sends sounds from the previous units in the UChain to hardware output busses, using a panning algorithm for stereo panning
bus: the channels are sent out starting with this bus.
point: the position of the source (0@0: center, positive x is left, positive y is front - although the difference between front and back is not audible in this algorithm)
dbRollOff: number of dB's amplitude rolloff per distance doubling
speakerRadius: the distance of the virtual microphones from the center of the room (i.e. half the distance between the speakers), in meters. When 0, no inter-speaker

delays are applied (only a distance-dependant doppler delay). Default value is 0.3, which is optimal for normal speaker setups. For headphone, use 0.1.
lag: a lag time applied on all parameters

noise

brownNoise
A brownian noise generator. Generates noise whose spectrum falls off in power by 6 dB per octave.
http://en.wikipedia.org/wiki/Brownian_noise
amp: amplitude (0-1) of the noise
seed: random seed (positive whole number). The same seed will always result in exactly the same noise on any computer. If you want decorrelated noise from multiple sources, change the seed on each source to a different number.

crackle
A noise generator based on a chaotic function.
chaosParam: value between 1 and 2; higher values produce more crackling
amp: amplitude (0-1) of the noise
seed: random seed (positive whole number). The same seed will always result in exactly the same noise on any computer. If you want decorrelated noise from multiple sources, change the seed on each source to a different number.

dust
A random impulse generator. Generates impulses of random amplitude (positive and negative) with random time intervals in between.
density: the average amount of impulses per second
amp: maximum amplitude (0-1) of the pulses
seed: random seed (positive whole number). The same seed will always result in exactly the signal on any computer. If you want decorrelated noise from multiple sources, change the seed on each source to a different number.

grayNoise
Generates noise which results from flipping random bits in a word. This type of noise has a high RMS level relative to its peak to peak level. The spectrum is emphasized towards lower frequencies.
amp: amplitude (0-1) of the noise
seed: random seed (positive whole number). The same seed will always result in exactly the same noise on any computer. If you want decorrelated noise from multiple sources, change the seed on each source to a different number.

lfNoise
A sample-and-hold noise generator. Generates random values at frequency rate.
freq: frequency (2-20000Hz)
amp: amplitude (0-1) of the noise
type: the type of noise:
0: step or sample-and-hold noise; hard jumps at each value change
1: linear interpolated noise
2: cubic interpolated noise
3: clip noise
- settings in between these types are also possible, effectively crossfading between them.
seed: random seed (positive whole number). The same seed will always result in exactly the same noise on any computer. If you want decorrelated noise from multiple sources, change the seed on each source to a different number.

pinkNoise

A pink noise generator. Generates noise whose spectrum falls off in power by 3 dB per octave. This gives equal power over the span of each octave. This version gives 8 octaves of pink noise.
http://en.wikipedia.org/wiki/Pink_noise
amp: amplitude (0-1) of the noise
seed: random seed (positive whole number). The same seed will always result in exactly the same noise on any computer. If you want decorrelated noise from multiple sources, change the seed on each source to a different number.

whiteNoise

A white noise generator. Generates noise whose spectrum has equal power at all frequencies.
http://en.wikipedia.org/wiki/White_noise
amp: amplitude (0-1) of the noise
seed: random seed (positive whole number). The same seed will always result in exactly the same noise on any computer. If you want decorrelated noise from multiple sources, change the seed on each source to a different number.

oscillator

blip
Band Limited ImPulse generator. All harmonics have equal amplitude. This is the equivalent of 'buzz' in MusicN languages.
freq: frequency (Hz)
numharm: number of harmonics (1-...)
amp: amplitude (0-1)

formant
A formant generator. Generates a set of harmonics around a formant frequency at a given fundamental frequency.
freq: frequency (Hz)
fromant: frequency of the formant (Hz)
bw: bandwidth of the formant as multiplier of the freq
amp: amplitude (0-1)

impulse
A non-bandlimited impulse generator. Outputs single sample impulses.
freq: frequency (Hz)
scale: a multiplier for the frequency (if scale is 0, only a single impulse is produced)
numSamples: width of the impulse in number of samples (1-100)
amp: amplitude (0-1)

pulse
A band-limited pulse wave generator.
http://en.wikipedia.org/wiki/Sawtooth_wave
freq: frequency (Hz)
width: relative width (0-1) of the pulse. 0.5 means a square wave.
amp: amplitude (0-1)

saw
A band-limited sawtooth wave generator.
http://en.wikipedia.org/wiki/Sawtooth_wave
freq: frequency (Hz)
amp: amplitude (0-1)

sine
A sine wave oscillator.
http://en.wikipedia.org/wiki/Sine_wave
freq: frequency (Hz)
phase: phase (-pi - pi)
amp: amplitude (0-1)

sync_saw
A synced saw wave generator.
http://en.wikipedia.org/wiki/Oscillator_sync
syncFreq: fundamental frequency (Hz)
sawFreq: frequency of synced saw - prefferably > syncFreq (Hz)
amp: amplitude (0-1)

shared_io

shared_buffer

** this Udef should be used in conjunction with 'shared_buffer_in' UMaps **

The 'shared_buffer' Udef creates a unit that is able to share a (soundfile) buffer or UMap with other units further in the chain. The 'shared_buffer' unit needs to before the units that use it in the order of the chain. To retreive the value in an other unit use the 'shared_buffer_in' UMapDef, and make sure the 'id' setting is the same on both. The buffer will always be mono (for multichannel audiofiles only the first channel will be loaded).

soundFile: a soundfile object of which the buffer will be shared. Note: the 'rate' and 'loop' settings of the soundfile are ignored, and can be set separately in the corresponding shared_buffer_in UMaps.

id: the id (0-99) by which the buffer can be retreived by a 'shared_buffer_in' UMap. In the GUI this will show up as a colored box, which can be dragged and dropped onto the receiving unit.

shared_point

** this Udef should be used in conjunction with 'shared_point_in' UMap **

The shared_point Udef creates an unit that is able to share point information with other units used further in the chain. This is useful when multiple units or unit parameters need to use the same point information, or derrive things from it. The shared_point would always need to come first in the order of the chain; it sends the point data to a bus. To retreive the data in another unit use the 'shared_in_point' UMapDef, and make sure the 'id' setting is the same on both. This will only work _within_ a single chain.

point: the point to be shared (can be an UMap)

id: the id (0-99) by which the point can be retreived by 'shared_point_in'

shared_value

** this Udef should be used in conjunction with 'shared_in' UMaps **

The 'shared_value' Udef creates a unit that is able to share a value or UMap with other units used further in the chain. The 'shared_value' unit would always need to be first in the order of the chain; it sends the value to a bus to the other units. To retreive the value in an other unit use the 'shared_in' UMapDef, and make sure the 'id' setting is the same on both. The value can be set in 6 different formats: value (0-1), freq (2-20000), amp (0-1, curved for amplitude use), trigger (a trigger signal), boolean (true/false) and range (0-1). The range type produces two shared values from the low and high value of the range. The 'shared_in' UMapDef will automatically map the value to the range of the parameter it is connected to, which means it is possible to use for example the 'freq' value of this unit to influence the 'amp' value of another unit. The shared_value itself does not output any sound, it only sends the value. All this will only work _within_ a single chain.

value: the value to share (can be UMap, not available for 'range' type)

range: the range of the value to be shared (not available for 'trigger' and 'boolean' types, in case of 'range' type this becomes the actual output values)

id: the id (0-99) by which the value can be retreived by a 'shared_in' UMap

type: the type of parameter (\value, \freq, \amp, \trigger, \boolean, \range)

lo_id / hi_id: id's of the lo and hi value in case of type 'range'

soundFile

bufSoundFile

A soundfile player. This plays soundfiles of any duration, with any (*) number of channels, by loading the whole file into the memory. Use this typically for shorter (less than apx 2 minutes) audiofiles. For longer files it is advised to use \diskSoundFile instead

soundFile: a BufSndFile object, a region of an existig soundfile url on your hard drive. The object knows the duration and file path of the soundfile, and can be set to clip off the start and end of the file.
- start: region startoffset (samples or seconds)
- end: end of the region (samples or seconds)
- loop: loop mode on or off (can be changed during playback)
- rate: playback rate (semitones or ratio)

level: playback level (amplitude of the soundfile)

(*) only the following numbers of channels are allowed:
1,2,3,4,5,6,7,8,10,12,16,24,32

diskSoundFile

A soundfile player. This plays soundfiles of any duration, with any (*) number of channels, by loading the whole file into the memory. Use this typically for longer (over apx 2 minutes) audiofiles. For shorter files you can also use \bufSoundFile.

soundFile: a DiskSndFile object, a region of an existig soundfile url on your hard drive. The object knows the duration and file path of the soundfile, and can be set to clip off the start and end of the file.
- start: region startoffset (samples or seconds)
- end: end of the region (samples or seconds) (**)
- loop: loop mode on or off (***)
- rate: playback rate (semitones or ratio)

level: playback level (amplitude of the soundfile)

(*) only the following numbers of channels are allowed:
1,2,3,4,5,6,7,8,10,12,16,24,32

grainSoundFile

A granular soundfile player. This will play tiny fragments of the soundfile with optional random dispersion and distribution.

soundFile: a MonoBufSndFile object, a region of an existig soundfile url on your hard drive. The object knows the duration and file path of the soundfile, and can be set to clip off the start and end of the file.
- start: region startoffset (samples or seconds)
- end: end of the region (samples or seconds)
- loop: loop mode on or off (can be changed during playback)
- rate: playback rate (semitones or ratio)

If the file has multiple channels, only the first one will be used
pos: range of position in the file between 0 (start) and 1 (end)
density: the average number of grains per second
timeVar: time variation factor (0-4). 0 means no variation, 4 means times between grains can be within (1/4) and 4 times the original duration
rateVar: ratio variation in number of octaves
overlap: average number of overlapping grains. The duration of the grains is change to match this.
pan: the pan position range. This is only active if numChannels > 1
seed: random seed
level: playback level (amplitude of the soundfile)
numChannels: number of output channels

loopSoundFile

loops a soundfile buffer, with settable crossfade time and optional random variation of loop start and duration.
loopStart: (range) minimum and maximum start position of the loop (0-1).
loopDur: (range) minimum and maximum duration of the loop (0-1).
The duration of the whole soundfile is mapped from 0 to 1 in loopStart and loopDur. The actual start position and duration are randomly chosen every time the loop starts again.
crossfade: the crossfade time in seconds. The loop engine has 4 voices, which means that if the loop duration in seconds is less than 1/4 of the crossfade time there may be audible clicks.
seed: random seed. The same value for seed gives the exact same signal on every machine, every time the unit is played.
soundFile: a BufSndFile object, a region of an existig soundfile url on your hard drive. The object knows the duration and file path of the soundfile, and can be set to clip off the start and end of the file.
- start: region startoffset (samples or seconds)
- end: end of the region (samples or seconds)
- loop: loop mode on or off (can be changed during playback)
- rate: playback rate (semitones or ratio)
rateScale: a scale value for the playback rate (can be modulated)
level: playback level (amplitude of the soundfile)

only the following numbers of channels are allowed:
1,2,3,4,5,6,7,8,10,12,16,24,32

trigSoundFile

plays a soundfile buffer upon a trigger (can be UMap), with settable crossfade time and start position.
trigger: the trigger that makes the playback jump to the start position
crossfade: crossfade time applied when trigger is initiated (max. voices: 4)
pos: position in the soundfile (0-1)
reverse: if true, the soundfile is played back in reverse. Position is also reversed in this case
soundFile: a BufSndFile object, a region of an existig soundfile url on your hard drive. The object knows the duration and file path of the soundfile, and can be set to clip off the start and end of the file.
- start: region startoffset (samples or seconds)
- end: end of the region (samples or seconds)
- loop: loop mode on or off (can be changed during playback, crossfade time does not apply here)
- rate: playback rate (semitones or ratio)
rateScale: a scale value for the playback rate (can be modulated)
level: playback level (amplitude of the soundfile)

only the following numbers of channels are allowed:
1,2,3,4,5,6,7,8,10,12,16,24,32

synthesis

babblingbrook_jmc
The famous James Mc.Cartney babbling brook. based on code posted to sc-users 2007-04-07 by james mccartney
amp: amplitude (0-1)
seed: random seed (positive whole number). The same seed will always result in exactly the same signal on any computer.

blipTest
A test signal generator, including a "blip" oscillator and pink noise generator, modulated with a pulse lfo. This generator is equal to the "blip" audioType in WFSCollider version 1.
rate: frequency (Hz) of pulse lfo
freq: frequency (Hz) of blip oscillator
noiseLevel: amplitude (0-1) of pink noise generator
blipLevel: amplitude (0-1) of blip oscillator

cymbalic_mcld
A cymbal sound by Dan Stowell. Based on the example at http://www.mcld.co.uk/cymbalsynthesis/ published 2008 by Dan Stowell
decay: decay time (s) of the cymbal
ratio: scale factor for the frequencies (0.25-4)
amp: amplitude (0-1)
seed: random seed (positive whole number). The same seed will always result in exactly the same signal on any computer.

default
The SuperCollider default sound.
freq: frequency (Hz) of the bell
amp: amplitude (0-1)
seed: random seed (positive whole number). The same seed will always result in exactly the same signal on any computer.

gendy
A GENDY generator (by Nick Collins). This is implementation of the dynamic stochastic synthesis generator conceived by Iannis Xenakis and described in Formalized Music (1992, Stuyvesant, NY: Pendragon Press) chapter 9 (pp 246-254) and chapters 13 and 14 (pp 289-322).
ampdist: Choice of probability distribution for the next perturbation of the amplitude of a control point. The distributions are (adapted from the GENDYN program in Formalized Music):
0: LINEAR.
1: CAUCHY.
2: LOGIST.
3: HYPERBCOS.
4: ARCSINE.
5: EXPON.
The "SINUS" distribution, which needs time-varying input adparam and ddparam, is left out as it would not work in this udef.

durdist: Choice of distribution for the perturbation of the current inter control point duration.
adparam: A parameter for the shape of the amplitude probability distribution
ddparam: A parameter for the shape of the duration probability distribution
freq: Minimum and maximum allowed frequencies of oscillation for the Gendy1 oscillator, so gives the smallest and largest period the duration is allowed to take on.
ampscale: Multiplier for the distribution's delta value for amplitude. An ampscale of 1.0 allows the full range of -1 to 1 for a change of amplitude.

durscale: Multiplier for the distribution's delta value for duration. An ampscale of 1.0 allows the full range of -1 to 1 for a change of duration.
initCPs: Initialise the number of control points in the memory. Xenakis specifies 12. There would be this number of control points per cycle of the oscillator, though the oscillator's period will constantly change due to the duration distribution.
knum: Current number of utilised control points (can be modulated).
amp: the amplitude of the signal (0-1).
seed: random seed (positive whole number). The same seed will always result in exactly the same signal on any computer. If you want different signals for different UChains that play at the same moment, use different seeds.

grainSine
A granular synthesis generator. This will play tiny segments ("grains") of sine waves with optional random dispersion and distribution.
range: the minimum and maximum frequency in between which each grain is randomly chosen
pos: range of position in the file between 0 (start) and 1 (end)
density: the average number of grains per second
timeVar: time variation factor (0-4). 0 means no variation, 4 means times between grains can be within (1/4) and 4 times the original duration
overlap: average number of overlapping grains. The duration of the grains is change to match this.
pan: the pan position range. This is only active if numChannels > 1
amp: the overall amplitude of the generator
seed: random seed
numChannels: number of output channels

klang
A bank of sine oscillators.
range: the minimum and maximum frequency
exponential: distribution of frequencies: 0: linear, 1: exponential
variation: amount of random variation. 0: no variation, 0.5: linear distribution, 1: full random.
amp: range of random amplitudes
phase: range of random phases
seed: random seed
n: number of oscillators

sos_bell
A simple bell sound by Dan Stowell. based on a sound-on-sound 'synth secrets' tutorial. Adapted to sound indefinitely.
freq: frequency (Hz) of the bell
decay: decay time (s) of the bell.
amp: amplitude (0-1)
seed: random seed (positive whole number). The same seed will always result in exactly the same signal on any computer.

utility

crossfade

Equal power crossfade between two sources.
crossfade: amount of left/right
numChannels: number of channels

duplicateChannel

duplicates a single channel to multiple outputs.
numChannels: number of output channels (*).

(*) only the following numbers of channels are allowed:
1,2,3,4,5,6,7,8,10,12,16,24,32

envelope_amp

An amplitude envelope.
env: an Env. It should have levels between 0 and 1, which will be mapped with according to the \amp ControlSpec.

gain

A simple gain stage. Can set a gain in dB and inverse phase.
gain: added gain (dB)
inverse: inverse phase (false/true)
numChannels: number of channels

mixer

mixes multiple channels together to a single one by summing
amp0 .. ampN: amplitude of each input signal
numChannels: number of channels (*).

(*) only the following numbers of channels are allowed:
1,2,3,4,5,6,7,8,10,12,16,24,32

multiply

A simple amplitude multiplication Udef.
mul: amplitude multiplication value (can be UMap)

splay

divides an arbitrary number of inputs over an arbitrary number of outputs via equal power intensity panning. Can also be used to manipulate fixed arrays of channels (i.e. rotate, leak etc.).
spread: the amount of spreading; 0 means all inputs go to first output, 1 (default) means all outputs will be used
width: the width sets how many outputs are used for each input. A cosine window is applied here.
center: rotates the outputs; 0 means the first input goes to the first output, 1 or -1 means the first input goes to the middle output.
numChannels: number of input channels (*).
numOutputs: number of output channels (1-8)

wfs_experimental

wfsDynamicDirectional

This udef creates a dynamic point source on the WFS system, with a directional radiation pattern. The position of the source can be changed in realtime (dynamic).

This feature of WFSCollider is currently under development, please test and report your findings

point: This point represents the absolute location of the point source.

direction: the direction of the radiation pattern

radiation: defines the radiation pattern in 4 values; [o,d,q,n]

- o: omnidirectional component (0-1)
- d: dipole component (0-1)
- q: quadrupole component (0-1)
- n: pole multiplier (1-8)

the graphic interface shows a directional amplitude plot of the currently set values

quality: (i) (\normal or \better) setting for the interpolation quality of the delays.

normal = linear interpolation

better = cubic interpolation

Cubic interpolation uses about 50% more cpu.

latencyComp: (i) cancels the delay caused bu the distance of the source. 0 = no compensation, 1 = full compensation. This can reduce the doppler shift for moving sources, and can ensure synchronicity between this source and other sources.

dbRollOff: decrease in level due to a doubling of the distance. In nature this is -6 dB in free field conditions, and might be -3 dB in extremely reverberant rooms.

maxAmpRadius: defines the radius of the circular area in the centre where the level will always be 0 dB. The dbRollOff starts where this area ends. So the larger the area, the louder the sounds outside of it.

wfs_io

wfsMasterOut

Output audio from the master audio interface of the system (if there is one).

Using this Udef in a UChain will cause the whole UChain to be played on the master computer instead of the servers.

On the Game of Life WFS system it can be used either to send audio digitally from the master computer to the servers (for which there are 8 buses available – i.e. one ADAT cable), or to the analog outputs of the master audio interface (for example to send audio signals in sync with the WFS system to other speakers or headphones).

bus: number of a hardware output (starting at 0)

toServers: if true (default), the output is to the digital inputs of the servers. If not selected, it is sent to analog outputs.

numChannels: number of channels (1-8)

wfsServerIn

Input audio sent digitally from the master computer on the server.

Using this Udef in your UChain will make the UChain play only on the servers. It is meant to use together with another UChain that holds a wfsMasterOut. Together, these can make a bridge between the master computer and the servers, typically needed for feeding live audio into the system and distributing it over the servers. On the Game of Life WFS system there are 8 buses available for this.

bus: number of the hardware input bus (0-7).

numChannels: number of channels (1-8)

wfs_panner

wfsIndex

This udef can send sound to an individual speaker on the WFS system. The index of the speaker can be changed in realtime (dynamic) when attaching an UMap, and will be static when no UMap is attached.

index: the number of the speaker, starting at 0.

gain: the output level (dB).

wfsSource

This udef creates a virtual source on the WFS system. It can be a point source or a plane wave source. The position of the source can static or changing over time (dynamic). The unit changes internally depending on the type of the source. If the point is set normally (default) the unit is in 'static' mode, meaning that the point can not be moved during playback. Changes to the point will not be audible until the unit (and chain) is stopped and started again. When applying an UMap on the point (location) of the source, the source becomes 'dynamic', which will add an extra 'quality' parameter, and make the source able to move over time. The reason for the difference between static and dynamic is in optimization; a 'static' source takes considerably less CPU power than a 'dynamic' source. If you want to move the source in realtime by hand, try using (for example) a 'point_lag' UMap on the point argument. This will change the umap into 'dynamic' mode, allowing it to move during playback.

point: This point represents the absolute location of the point source, or the center position of a plane wave (the point on the line nearest to the center of the room). When point becomes a UMap, the unit goes into 'dynamic' mode, except when the UMap outputs single values (for example: 'random_point').

type: (i) \point or \plane

quality: (i) (\normal or \better) setting for the interpolation quality of the delays (only in 'dynamic' mode)
normal = linear interpolation
better = cubic interpolation
Cubic interpolation uses about 50% more cpu.

latencyComp: (i) cancels the delay caused bu the distance of the source. 0 = no compensation, 1 = full compensation. This can reduce the doppler shift for moving sources, and can ensure synchronicity between this source and other sources.

dbRollOff: decrease in level due to a doubling of the distance. In nature this is -6 dB in free field conditions, and might be -3 dB in extremely reverberant rooms.

maxAmpRadius: defines the radius of the circular area in the centre where the level will always be 0 dB (maximum). The dbRollOff starts where this area ends. So the larger the area, the louder the sounds outside of it.

UnitRacks

UnitRacks are presets made of several Udefs.

wfs

wfsPathPlayerTrack

A wfsDynamicPoint with a ‘trajectory’ UMap applied.

wfsSimpleReverb

A combination of wfsPoint, 4 times simpleReverb and wfsStaticPlane, and wfsDynamicPoint. Place a bufSoundFile or diskSoundFile as first Unit in the Chain window and then add wfsSimpleReverb.

This UnitRack is connected in such a way that the controls of the simpleReverb are connected to wfsPoint. I this way the placement is taken into consideration by the reverbs. Each reverb is sent to a wfsStaticPlane in such a way that the plane waves together form 4 'walls' of reverberation.

The direct out (or dry) signal is made by wfsDynamicPoint. In wfsDynamicPoint pointFromBus is selected and it gets its coordinates from wfsPoint, like the reverberators.

I this way you can add reverberation to your sounds in a way that makes sense with WFS.

UMapDefs

UMapDefs are definition files for UMaps, which can be used to modulate or control parameters of Units. They can be found in the Udefs window, and dragged into the Chain window from there, on top of a parameter name.

automation

envelope

An envelope generator.
env: an Env or EnvM object, containing time and level values.
timeScale: a scale for the time/duration of the Env (can be modulated)
loop: loopmode (0: off, 1: loop, 2: alternate)
delay: delay time before starting the env
trigger: a trigger that restarts the env

envelope_rel

An envelope generator of which the duration is automatically scaled to that of the event.
env: an Env or EnvM object, containing time and level values.
timeScale: if timeScale is 1, the duration of the envelope will become exactly that of the event, lower and higher timeScales result in a shorter and longer envelope durations.
loop: loopmode (0: off, 1: loop, 2: alternate)
delay: delay time before starting the env (not affected by timeScale)
trigger: a trigger that restarts the env

line

An line / ramp generator.
a: start value
b: end value
curve: a curve value for the line. A 0 (zero) curve creates a linear line, a positive curve value makes the line tend towards the start value, and v.v.
duration: duration of the line in seconds (can be modulated)
loop: loopmode (0: off, 1: loop, 2: alternate)
delay: delay time before the line starts
trigger: a trigger that restarts the line

line_rel

An line generator of which the duration is automatically scaled to that of the event.
a: start value
b: end value
curve: a curve value for the line. A 0 (zero) curve creates a linear line, a positive curve value makes the line tend towards the low value, and v.v.
timeScale: if timeScale is 1, the duration of the line will become exactly that of the event, lower and higher timeScales result in a shorter and longer envelope durations.
loop: loopmode (0: off, 1: loop, 2: alternate)
delay: delay time before the line starts
trigger: a trigger that restarts the line

control

envir_get

Creates an UMap that gets the value of an environment variable in SuperCollider. The value will only be used if it exists, and if it is valid input.

value: the actual output value (this will change when the variable changes, but can also be changed manually)

active: if true, the UMap will listen to the specified variable and update it regularly. A message with the key and value range is posted in the post window when active becomes true. Hitting cmd-. will make the UMap stop updating it's value, but whenever the Unit is started, it will re-activate.

key: the name of the variable (as String). If the key is not specified (default), it will automatically be set to the name of the parameter to which the UMap is connected when 'active' is set to true.

speed: the speed by which the value is checked and updated, in times per second

mapped: if true, the value will be mapped from 0-1 to the output range of the UMap, if false (default) the value will be used directly.

global_control

Creates an UMap that listens to a control in UGlobalControl

value: the actual output value (this will change when the variable changes, but can also be changed manually)

active: if true, the UMap will listen to the specified global control and update it whenever it changes. If a non-existing key is used, it will be added to UGlobalControl automatically when 'active' == true.

controlKey: the name of the globalControl (a Symbol)

osc_set

Creates an UMap that listens to OSC messages. Works similar to all 'midi_xxx' UMapDefs.

value: the actual output value (this will change when a the OSC message is received, but can also be changed manually)

active: if true, the UMap will listen to the specified OSC message. Hitting cmd-. will make the UMap stop listening, but whenever the Unit is started, it will re-activate. The OSC port is always that of the SuperCollider or WFSCollider application, normally 57120 but that may change sometimes. The current port number is posted together with the message name and range in the post window when active is set to true.

message: the OSC message (String) to which the UMap is listening. This will be set to the name of the parameter the UMap is connected to if not specified (default). The actual message the UMap listens to is always preceeded by a "/" char.

mapped: if true, the received values will be mapped from 0-1 to the range of the output value. If false (default) no mapping is applied and values will be set directly.

convert

deg_rad
Creates an UMap for converting radians to degrees, to be used on any UMap with an angle setting in radians (-pi to pi)
deg: amount of degrees (-180-180)

expand
Creates an UMap that expands certain multi-value args into single values. This UMap patches it stored values and UMaps directly to the correct inputs. Use it for example to control the x and y values of a Point separately, or to make EQ's change over time.
Argnames depend on the type of Spec. The following Specs have dedicated argnames:
RangeSpec:
lo: low / first value
hi: high / second value
PointSpec:
x: x value (default range -200m to 200m)
y: y value (default range -200m to 200m)
EQSpec:
<band_name>_<param_name>: an arg with the correct spec for the correct parameter
... etc
ArraySpec:
value<n>: the value at index <n>
all arg ranges are mapped to that of the parameter to which the UMap is connected.
This UMapDef replaces the former 'lo_hi' and 'x_y' UMapDefs, which are now automatically re-routed to 'expand'.

linear
gives an arg that doesn't have a linear ArgSpec a linear warp in the same range. For example; \freq args usually have non-linear warps (\exp in this case), as well as \amp args.
value: the value to be mapped (0-1)

map
Maps a value between 0 and 1 to a specific range, with user-settable warp (linear, exponential etc.). The warp setting overrides that of the input spec. It can also be used to covert time or integers to a specific range for modulation.
value: the value to be mapped (0-1)
min: minimum map output (value == 0)
max: maxmum map output (value == 1)
warp: warp type
-1: auto (default - warp type of input Spec)
0: \step
1: \lin
2: \exp
3: \cos
4: \sin
5: \curve
6: \amp
curve: the curve amount when warp is set to \curve.

note_freq
Creates an UMap for converting diatonic mini-note values to a frequency, in semitones where 69 == 440Hz.
note: a midi note value
round: rounding factor (0 - 12)
detune: detune amount in cents (-50 - 50)

filter

delay

This UMapDef creates an UMap for delaying value changes. The delay time itself can also be modulated.
value: the value (or UMap) to be delayed
time: the length of the delay.
timeScale: the number of seconds by which 'time' is multiplied. This value can not be changed during playback.
lag: smoothing time applied to change of delay time, use to prevent sudden jumps during change of delay time.

lag

An UMap that slows down value changes over time. It is implemented as a one-pole low-pass filter, with different settings for up- and downward movement.
value: the value upon which the lag is applied (this can be another UMap as well)
timeUp: the time it takes to move upwards to within 0.01% towards a new value
timeDown: the time it takes to move downwards to within 0.01% towards a new value
order: (1-4) order of the lag, a higher order produces a smoother lag curve. The UMap cascades four lag filters, and the order parameter sets how many of them are actually used.

lag_line

An UMap that slows down value changes over time. It does this by creating a line between the old and new value
value: the value upon which the lag is applied (this can be another UMap as well)
time: the time it takes to move to the new value
curve: a curve value for the line. 0 means linear, a negative curve makes the value move faster in the beginning, a positive curve makes it move faster at the end.

lag_lpf

An UMap that slows down value changes over time. It is implemented as a two-pole low-pass filter. Please note that changing the 'time' variable during playback may cause the filter to become temporarily unstable.
value: the value upon which the lag is applied (this can be another UMap as well)
time: the time it takes to move to within 0.01% towards a new value

median

An UMap that applies a median filter, effectively removing spikes from incoming values by calculating the median value of a number of samples (length).
value: the value upon which the median is applied (this can be another UMap as well)
length: length of the median window
amount: mix between original signal and filtered signal (0-1)

peak

Creates an UMap for following the highest or lowest peak from the incoming value (or UMap)
value: the input value (or UMap)
decay: (0-1) a decay parameter for the (low)peak detection. 0 means no decay, 1 means infinite decay
direction: (-1 - 1) if 1 (default) the UMap will follow output the peak / maximum value, if -1 the UMap will follow the bottom / minimum value. Values in between will interpolate between these two.

ringz

Creates an UMap that routes an input value through a ringing filter with attack and decay and adds the result to the original value. This causes oscillations to happen at value changes.

value: the value upon which the lag is applied (this can be another UMap as well)
freq: the frequency of the filter. Rapid changes of this value may cause the filter to be temporarily unstable
attackTime: the attack time of the
decayTime: the decay time of the ringing filter
amp: amplitude of the ringing filter

slew
An UMap that limits the speed of value changes
value: the value upon which the lag is applied (this can be another UMap as well)
timeUp: the time it takes to move upwards across the whole range
timeDown: the time it takes to move downwards across the whole range

slope
An UMap that differentiates its input, returning the amount of change per control cycle.
value: the value upon which the lag is applied (this can be another UMap as well)
abs: if true the UMap returns the absolute slope, if false it returns negative and positive slope
range: the range to which the output values are mapped

function

function
Creates an UMap that evaluates a SuperCollider function every time the UChain is started
function: the function. The result of the function should be a value between 0 and 1
value: the result of the function (for display only)

random_point
Creates an UMap that generates a new random point value each time it is started. This UMap can also be used on 'init' mode parameters.
center: (Point) the center of the rectangle within which a random point can be generated.
radius: (Point) x and y radius of the rectangle within which the random point is generated.
value: the output point (can only be changed by the UMap itself)

random_time
Creates an UMap that generates a new random time value each time it is started. This UMap can also be used on 'init' mode parameters. It can only be used for time (SMPTESpec) parameters.
min: minimum time
max: maximum time
value: the output value (can only be changed by the UMap itself)

random_value
Creates an UMap that generates a new random value each time it is started. Typical use for this would be at the 'seed' parameter of any random-generator unit. This UMap can also be used on 'init' mode parameters.
range: the range in between which the value will be created
value: the output value (can only be changed by the UMap itself)

input

amp_follow

Creates an UMap that listens to an audio bus from a previous unit in the chain, and follow it's amplitude.
fromRange: the amplitude range to be used as output
toRange: the range to which the output is scaled
attackTime: the attack time of the amplitude follower
releaseTime: the release time of the amplitude follower
invert: if true, the fromRange is inverted (i.e. low values become high and v.v.)
curve: a curvature value for toRange. If curve == 0, the range is linear.
clipMode: can be one of the following modes:
0 - 'clip' (default): clip values outside fromRange
1 - 'fold': fold values outside fromRange
2 - 'wrap': wrap around values outside fromRange
3 - 'none': no clipping applied (values can go outside toRange)
clipSoftness: softens the edges of 'clip' and 'fold' modes.

map_audio

Creates an UMap that listens to an audio bus from a previous unit in the chain and converts it to a control signal.
fromRange: the input range to listen to (-1 to 1)
toRange: the range to which the output is scaled
curve: a curvature value for toRange. If curve == 0, the range is linear.
invert: if true, the fromRange is inverted (i.e. low values become high and v.v.)
lag: lagging filter time for value changes
clipMode: can be one of the following modes:
0 - 'clip' (default): clip values outside fromRange
1 - 'fold': fold values outside fromRange
2 - 'wrap': wrap around values outside fromRange
3 - 'none': no clipping applied (values can go outside toRange)
clipSoftness: softens the edges of 'clip' and 'fold' modes (0-1).

onsets

Creates an UMap that listens to an audio bus from a previous unit in the chain and detects the beginning of notes/drumbeats/etc..
threshold: the detection threshold, typically between 0 and 1
odftype: chooses which onset detection function is used. In many cases the default will be fine. The following choices are available;
\power
generally OK, good for percussive input, and also very efficient
\magsum
generally OK, good for percussive input, and also very efficient
\complex
performs generally very well, but more CPU-intensive
\rcomplex
performs generally very well, and slightly more efficient than \complex
\phase
generally good, especially for tonal input, medium efficiency
\wphase
generally very good, especially for tonal input, medium efficiency
\mkl
generally very good, medium efficiency, pretty different from the other methods
relaxtime: specifies the time (in seconds) for the normalisation to "forget" about a recent onset. If you find too much re-triggering (e.g. as a note dies away unevenly) then you might wish to increase this value.
floor: floor is a lower limit, connected to the idea of how quiet the sound is expected to get without becoming indistinguishable from noise. For some cleanly-recorded classical music with wide dynamic variations, I found it helpful to go down as far as 0.000001.

mingap: specifies a minimum gap (in FFT frames) between onset detections, a brute-force way to prevent too many doubled detections.
medianspan: specifies the size (in FFT frames) of the median window used for smoothing the detection function before triggering.
range: the UMap's output range. Triggers will cause jumps from the minimum to the maximum of this range.
fftsize: the size of the fft buffer used for the analisys. A smaller buffer will result in quicker but less accurate detection.

<u>pitch_follow</u>
Creates an UMap that listens to an audio bus from a previous unit in the chain, and follow it's pitch.
ampTreshold: an amplitude threshold above which pitch detection is performed
median: number of frames of built-in median filter
pitchRange: the range of pitches to detect (will never go out of range)
toRange: the range to which the pitchRange is scaled
invert: if true, the toRange is inverted (i.e. low values become high and v.v.)
curve: a curvature value for toRange. If curve == 0, the range is linear.
hasPitchID: shared value id for a signal indicating the 'clarity' of the pitch. This value can be obtained with a 'shared_in' UMap further in the chain, using the same id. The signal varies from 0 to 1, indicating the purity of the tone, where 0 is no tone and 1 is maximum tone.

<u>rec_soundBuf</u>
This UMapDef can be used to replace a regular BufSndFile (soundFile) on \bufSoundFile, \grainSoundFile, \loopSoundFile and \trigSoundFile units. It creates a single-channel (mono) buffer that can record audio from previous units in the chain, in realtime. The buffer is passed on to the unit for immediate playback. Note that the buffer is always empty at the start, and will be deleted after stopping the chain.
buffer: the buffer; In the editor the duration of the buffer can be set in seconds.
recLevel: level of recording input (0 means silence)
preLevel: level of the existing material in the buffer, to mix with the input
run: turns the recording on and off
loop: if enabled, the recording will loop (i.e. start at the beginning of the buffer when the end is reached), if off the recording will end when the end of the buffer is reached.
trigger: a trigger to start or reset the recording at the beginning of the buffer
playbackRate: the rate for playback (passed through to the playback unit)
playbackLoop: enables/disables looped playback (passed through to the playback unit)

midi

midi_bend

Creates an UMap that listens to MIDI pitchbend messages. Works similar to all 'midi_xxx' UMapDefs.
value: the actual output value (this will change when a midi message is received, but can also be changed manually)
active: if true, the UMap will listen to the specified MIDI message. Hitting cmd-. will make the UMap stop listening, but whenever the Unit is started, it will re-activate. The MIDI client will be inited when the first midi_xxx UMap is made 'active'.
channel: the MIDI channel to listen to. 0-15 or nil (any channel).
learn: if learn is true, the 'channel' value will be changed according to any incoming MIDI pitchbend message.
fromRange: the used range of the midi message
toRange: the output range of the UMap
invert: if true, the input range is inverted (i.e. low values become high and v.v.)
curve: a curvature value for the mapped range. If curve == 0, the range is linear.
clipMode: can be one of the following modes (Symbol):
'ignore': ignore values outside fromRange
'clip' (default): clip values outside fromRange
'fold': fold values outside fromRange
'wrap': wrap around values outside fromRange
'none': no clipping applied (values can go outside toRange)
clipSoftness: softens the edges of 'clip' and 'fold' modes.

midi_cc

Creates an UMap that listens to MIDI control messages. Works similar to all 'midi_xxx' UMapDefs.
value: the actual output value (this will change when a midi message is received, but can also be changed manually)
active: if true, the UMap will listen to the specified MIDI message. Hitting cmd-. will make the UMap stop listening, but whenever the Unit is started, it will re-activate. The MIDI client will be inited when the first midi_xxx UMap is made 'active'.
cc: the control number (0-127) to listen to
channel: the MIDI channel to listen to. 0-15 or nil (any channel).
learn: if learn is true, the 'channel' and 'cc' values will be changed according to any incoming MIDI control message, as long as 'learn' is true.
fromRange: the used range of the midi message
toRange: the output range of the UMap
invert: if true, the input range is inverted (i.e. low values become high and v.v.)
curve: a curvature value for the mapped range. If curve == 0, the range is linear.
clipMode: can be one of the following modes (Symbol):
'ignore': ignore values outside fromRange
'clip' (default): clip values outside fromRange
'fold': fold values outside fromRange
'wrap': wrap around values outside fromRange
'none': no clipping applied (values can go outside toRange)
clipSoftness: softens the edges of 'clip' and 'fold' modes.

midi_noteOn

Creates an UMap that listens to MIDI note on messages. Works similar to all 'midi_xxx' UMapDefs.
value: the actual output value (this will change when a midi message is received, but can also be changed manually)
active: if true, the UMap will listen to the specified MIDI message. Hitting cmd-. will make the UMap stop listening, but whenever the Unit is started, it will re-activate. The MIDI client will be inited when the first midi_xxx UMap is made 'active'.
channel: the MIDI channel to listen to. 0-15 or nil (any channel).
learn: if learn is true, the 'channel' value will be changed according to any incoming MIDI note on message, as long as 'learn' is true.

fromRange: the used range of the midi message
toRange: the output range of the UMap
invert: if true, the input range is inverted (i.e. low values become high and v.v.)
curve: a curvature value for the mapped range. If curve == 0, the range is linear.
clipMode: can be one of the following modes (Symbol):
'ignore': ignore values outside fromRange
'clip' (default): clip values outside fromRange
'fold': fold values outside fromRange
'wrap': wrap around values outside fromRange
'none': no clipping applied (values can go outside toRange)
clipSoftness: softens the edges of 'clip' and 'fold' modes.
The 'value' arg range is mapped to that of the parameter to which the UMap is connected.

<u>midi_noteOn_freq</u>

Creates an UMap that listens to MIDI note on messages. It implements diatonic scaling meant for use with \freq parameters. Works similar to all 'midi_xxx' UMapDefs.
value: the actual output value (this will change when a midi message is received, but can also be changed manually)
active: if true, the UMap will listen to the specified MIDI message. Hitting cmd-. will make the UMap stop listening, but whenever the Unit is started, it will re-activate. The MIDI client will be inited when the first midi_xxx UMap is made 'active'.
channel: the MIDI channel to listen to. 0-15 or nil (any channel).
learn: if learn is true, the 'channel' value will be changed according to any incoming MIDI note on message, as long as 'learn' is true.
range: the used range of input note numbers. Notes outside this range will be ignored.
a: the frequency of "A3" (note number 69)
transpose: amount of transposition in semitones
scale: scaling of semitones around (transposed) note number 69
invert: if true, the range is inverted around note number 69

<u>midi_note_gate</u>

Creates an UMap that listens to MIDI velocity in note messages of a specific note. It sets the value to the velocity of a note on message, and to 0 for note-off messages. Works similar to all 'midi_xxx' UMapDefs.
value: the actual output value (this will change when a midi message is received, but can also be changed manually)
active: if true, the UMap will listen to the specified MIDI message. Hitting cmd-. will make the UMap stop listening, but whenever the Unit is started, it will re-activate. The MIDI client will be inited when the first midi_xxx UMap is made 'active'.
nn: the note number to listen to
channel: the MIDI channel to listen to. 0-15 or nil (any channel).
learn: if learn is true, the 'channel' and 'nn' values will be changed according to any incoming MIDI note on message, as long as 'learn' is true.
fromRange: the used range of the midi message
toRange: the output range of the UMap
invert: if true, the input range is inverted (i.e. low values become high and v.v.)
curve: a curvature value for the mapped range. If curve == 0, the range is linear.
clipMode: can be one of the following modes (Symbol):
'ignore': ignore values outside fromRange
'clip' (default): clip values outside fromRange
'fold': fold values outside fromRange
'wrap': wrap around values outside fromRange
'none': no clipping applied (values can go outside toRange)
clipSoftness: softens the edges of 'clip' and 'fold' modes.

<u>midi_touch</u>

Creates an UMap that listens to MIDI aftertouch (channel pressure) messages. Works similar to all 'midi_xxx' UMapDefs.

value: the actual output value (this will change when a midi message is received, but can also be changed manually)
active: if true, the UMap will listen to the specified MIDI message. Hitting cmd-. will make the UMap stop listening, but whenever the Unit is started, it will re-activate. The MIDI client will be inited when the first midi_xxx UMap is made 'active'.
channel: the MIDI channel to listen to. 0-15 or nil (any channel).
learn: if learn is true, the 'channel' value will be changed according to any incoming MIDI aftertouch message.
fromRange: the used range of the midi message
toRange: the output range of the UMap
invert: if true, the input range is inverted (i.e. low values become high and v.v.)
curve: a curvature value for the mapped range. If curve == 0, the range is linear.
clipMode: can be one of the following modes (Symbol):
'ignore': ignore values outside fromRange
'clip' (default): clip values outside fromRange
'fold': fold values outside fromRange
'wrap': wrap around values outside fromRange
'none': no clipping applied (values can go outside toRange)
clipSoftness: softens the edges of 'clip' and 'fold' modes.

modulation

gendy

A GENDY generator (by Nick Collins). This is implementation of the dynamic stochastic synthesis generator conceived by Iannis Xenakis and described in Formalized Music (1992, Stuyvesant, NY: Pendragon Press) chapter 9 (pp 246-254) and chapters 13 and 14 (pp 289-322).

ampdist: Choice of probability distribution for the next perturbation of the amplitude of a control point. The distributions are (adapted from the GENDYN program in Formalized Music):
0: LINEAR.
1: CAUCHY.
2: LOGIST.
3: HYPERBCOS.
4: ARCSINE.
5: EXPON.

The "SINUS" distribution, which needs time-varying input adparam and ddparam, is left out as it would not work in this udef.

durdist: Choice of distribution for the perturbation of the current inter control point duration.

adparam: A parameter for the shape of the amplitude probability distribution

ddparam: A parameter for the shape of the duration probability distribution

freq: Minimum and maximum allowed frequencies of oscillation for the Gendy1 oscillator, so gives the smallest and largest period the duration is allowed to take on.

ampscale: Multiplier for the distribution's delta value for amplitude. An ampscale of 1.0 allows the full range of -1 to 1 for a change of amplitude.

durscale: Multiplier for the distribution's delta value for duration. An ampscale of 1.0 allows the full range of -1 to 1 for a change of duration.

initCPs: Initialise the number of control points in the memory. Xenakis specifies 12. There would be this number of control points per cycle of the oscillator, though the oscillator's period will constantly change due to the duration distribution.

knum: Current number of utilised control points (can be modulated).

seed: random seed (positive whole number). The same seed will always result in exactly the same signal on any computer. If you want different signals for different UChains that play at the same moment, use different seeds.

range: the output value range

lfo_noise

Creates an UMap with a low frequency noise oscillator.

freq: the frequency of the noise generator

type: the type of noise:
0: step or sample-and-hold noise; hard jumps at each value change
1: linear interpolated noise
2: cubic interpolated noise

seed: random seed (positive whole number). The same seed will always result in exactly the signal on any computer.

range: the output range

lfo_pulse

Creates an UMap with a low frequency pulse wave oscillator.

freq: the frequency of the pulse wave

phase: the initial phase (0-1) of the pulse wave

width: the width of the pulse. 0.5 (default) creates a square wave

range: the output range

lfo_saw

Creates an UMap with a low frequency saw/triangle wave oscillator.

freq: the frequency of the pulse wave

phase: the initial phase (0-1) of the pulse wave

width: the width of the sawtooth;
0 creates a saw wave: |\|\
0.5 creates a traingular wave: /\/\
1 creates an reversed saw wave: /|/|
range: the output range

lfo_sine
Creates an UMap with a low frequency sine wave oscillator.
freq: the frequency of the sine wave
phase: (-pi - pi) the start phase of the sine wave (can be modulated)
range: the output range

lfo_step_noise
Creates an UMap for generating step noise. Random values are chosen on time intervals.
time: the time between the steps in seconds, minimum and maximum
type: the type of transitions; step or line:
0: step or sample-and-hold noise; hard jumps at each value change
1: linear interpolation
seed: random seed (positive whole number). The same seed will always result in exactly the signal on any computer.
range: the output range

phasor
Creates an UMap with a looping linear ramp, that can be reset via a trigger.
speed: the speed of the ramp
up: if true (default) the ramp moves upwards, if false the ramp moves downwards
startPos: the start value of the ramp
range: the output range
trigger: if set, the ramp will jump to 'startPos'

sequencer_16
Creates an UMap that can sequence a number (maximum 16) of values (or UMaps) over time. If you need to sequence <= values, it is more efficient to use \sequencer_8.
speed: number of steps per second
reverse: if true, move backwards through the steps
range: the range (0-15) of used steps.
smooth: smoothening parameter (0: hard steps, 1: linear interpolation)
value0 - value15: the values of the steps (can be UMaps)

sequencer_8
Creates an UMap that can sequence a number (maximum 8) of values (or UMaps) over time. If you need to sequence > 8 values, use \sequencer_16 instead.
speed: number of steps per second
reverse: if true, move backwards through the steps
range: the range (0-7) of used steps.
smooth: smoothening parameter (0: hard steps, 1: linear interpolation)
value0 - value7: the values of the steps (can be UMaps)

vibrato
Creates an UMap for vibrato modulation. The typical use would be on a 'freq' parameter, but it will work for any type.
value: the fundamental value around which the vibrato takes place
freq: the frequency (rate) of the vibrato
depth: size of vibrato deviation around the fundamental, as a proportion of the fundamental. 0.02 = 2% of the fundamental.
delay: delay before vibrato is established in seconds (a singer tends to attack a note and then stabilise with vibrato, for instance)

onset: transition time in seconds from no vibrato to full vibrato after the initial delay time
rateVar: noise on the freq, expressed as a proportion of the rate; can change once per cycle of vibrato
depthVar: noise on the depth of modulation, expressed as a proportion of the depth; can change once per cycle of vibrato. The noise affects independently the up and the down part of vibrato shape within a cycle
lag: a lag time for the fundamental value
seed: random seed (positive whole number). The same seed will always result in exactly the same random variation on any computer.

operator

add
: Creates an UMap adding (+) two values together.
value: the value to add to
add: the amount to add
factor: multiplier for add (negative value results in subtraction0

divide
: Creates an UMap for division (/) of two values.
value: the value
divide: the divider
negative: divide by negative value
factor: exponent of 10 to multiply the divider by

max
: This UMap outputs the highest (maximum) out of two input values
a: the first value
b: the second value

min
: This UMap outputs the lowest (minumum) out of two input values
a: the first value
b: the second value

multiply
: Creates an UMap for multiplication (*) of two values.
value: the value
multiply: the multiplier
negative: multiply with negative value
factor: exponent of 10 to multiply the multiplier by

round
: Creates an UMap for rounding off (quantizing) values. The output value will be the nearest multiple of the 'round' value.
value: the value to be rounded
round: the value to round to

point_filter

delay_point

Creates an UMap for applying a delay to a point input. When the point is moving the delayed point will always follow behind it.
point: the point to lag
time: the length of the delay.
timeScale: the number of seconds by which 'time' is multiplied. This value can not be changed during playback.
lag: smoothing time applied to change of delay time, use to prevent sudden jumps during change of delay time.

follow_point

Creates an UMap for applying speed and rotation limits to a point input.
point: the point to follow
maxSpeed: maximum speed in m/s
maxRotation: maximum rotation in deg/s

lag_point

Creates an UMap for applying a lagging filter to point input. It's main purpose is to reduce unwanted doppler shift effects.
point: the point to lag
time: the time it will take for the point to travel to a new location
linear: if true the lag will be a linear (ramp), if false (default) the lag is applied using a 3th order filter. The latter will produce less unnatural doppler shift changes.

slew_point

Creates an UMap for applying a speed limit to point input.
point: the point to lag
maxSpeed: maximum speed in m/s

point_to_value

point_angle

Creates an UMap for converting the angle between two points (or point UMaps) to a value range.
point1: the first point
point2: the second point
centerAngle: the reference angle (subtracted from the absolute angle)
fromRange: the angle range from which the output value will be calculated, divided by pi
toRange: the value range to which the angle range will be mapped
invert: (boolean) if true the angle range will be inverted
clipMode: when the angle is outside the fromRange, the clipMode decides what happens. The angle is "unwrapped" internally, so if the point would for example spin multiple times it will go outside the clipping area.
0 - 'clip': clip inside the value range
1 - 'fold': fold/mirror inside the value range
2 - 'wrap' (default): wrap around the value range
3 - 'none': no clipping applied (values can go outside the value range)
clipSoftness: softens the edges of 'clip' and 'fold' modes (0-1).

point_axis

Creates an UMap for converting a point (or point UMap) to a single value, by projecting it on a line. The line can be the x axis (default) or y axis, but via the 'rotate' argument it can also be somewhere in between.
point: the point
which: the axis to project on

rotate: amount of rotation (to deviate from the axis)
fromRange: the range from which the output value will be calculated
toRange: the value range to which the range will be mapped
clipMode: when the value is outside the fromRange, the clipMode decides what happens
0 - 'clip' (default): clip inside the value range
1 - 'fold': fold/mirror inside the value range
2 - 'wrap': wrap around the value range
3 - 'none': no clipping applied (values can go outside the value range)
clipSoftness: softens the edges of 'clip' and 'fold' modes (0-1).

point_direction
Creates an UMap for getting the direction angle of a moving point.
point: the point
centerAngle: the reference angle (subtracted from the absolute angle)
fromRange: the angle range from which the output value will be calculated, divided by pi
toRange: the value range to which the angle range will be mapped
invert: (boolean) if true the angle range will be inverted
clipMode: when the angle is outside the fromRange, the clipMode decides what happens. The angle is "unwrapped" internally, so if the point would for example spin multiple times it will go outside the clipping area.
0 - 'clip': clip inside the value range
1 - 'fold': fold/mirror inside the value range
2 - 'wrap' (default): wrap around the value range
3 - 'none': no clipping applied (values can go outside the value range)
clipSoftness: softens the edges of 'clip' and 'fold' modes (0-1).
lag: linear lag time for output

point_distance
Creates an UMap for converting the distance between two points (or point UMaps) to a value range.
point1: the first point
point2: the second point
fromRange: the distance range from which the output value will be calculated
toRange: the value range to which the distance range will be mapped
invert: (boolean) if true the distance range will be inverted
clipMode: when the distance is outside the fromRange, the clipMode decides what happens
0 - 'clip' (default): clip inside the value range
1 - 'fold': fold/mirror inside the value range
2 - 'wrap': wrap around the value range
3 - 'none': no clipping applied (values can go outside the value range)
clipSoftness: softens the edges of 'clip' and 'fold' modes (0-1).

point_in_rect
Creates an UMap for testing if a point is inside or outside a rectangle.
point: the point
center: the center of the rect area
radius: the radius of the rect area
inValue: the value to return when the point is inside the rect
outValue: the value to return when the point is outside the rect

point_speed
Creates an UMap for measuring the speed of a point (or point UMap) and convert it to a value.
point: the point
fromRange: the speed range from which the output value will be calculated
toRange: the value range to which the speed range will be mapped
clipMode: when the speed is outside the fromRange, the clipMode decides what happens

0 - 'clip' (default): clip inside the value range
1 - 'fold': fold/mirror inside the value range
2 - 'wrap': wrap around the value range
3 - 'none': no clipping applied (values can go outside the value range)
clipSoftness: softens the edges of 'clip' and 'fold' modes (0-1).

point_utility

clip_point

Creates an UMap for applying clipping, wrapping, folding (mirroring) and more on points
point: the point to clip
center: the center of the clipping area
radius: the radius of the clipping area
clipMode: can be one of the following modes:
0 - 'clip' (default): clip coordinates outside clipping area
1 - 'fold': fold/mirror coordinates outside clipping area
2 - 'wrap': wrap around coordinates outside clipping area
3 - 'none': no clipping applied (values can go outside clipping area)
clipSoftness: softens the edges of 'clip' and 'fold' modes (0-1).

crossfade_point

Creates an UMap for crossfading between two points via linear interpolation. The points can also be UMaps.
a: the first point
b: the second point
crossfade: (0-1) the crossfading position (a to b)

index_trajectory

Creates an UMap for indexing through 2D spatial trajectories. Via the 'index' value, which can be another UMap, all positions on the course of the path can be reached. The index scales between 0 and 1, the start and end positions of the trajectory. The time information stored in the trajectory object is not used; the distance between the breakpoints is used equally. This umap is intended for use on units with PointSpec or WFSPointSpec based args.
index: a value between 0 and 1, referring to the start and end points of the path. If the path is set to 'loop', the path is extended by adding the start point as last point.
trajectory: a WFSPathBuffer object. This object creates and points to a buffer with the data of the trajectory. This buffer is either filled with data sent directly, or data read from a .wfspath file (generated via the [write data] button in the GUI). Data from file is usually more reliable than sending via network. Note that the 'delay' and 'rate' settings have no effect on the behaviour of this UMap.
addPoint: a point (or point UMap) can be added to the trajectory position, effectively moving the trajectory as a whole.

mirror_point

Creates an UMap intended that mirrors a point across a line. This can be used for example for creating early reflection positions.
point: the point to rotate (can be an UMap)
plane: the line/plane across which the point is mirrored.
absolute: if true, the point will only be mirrored when it is in front of the line, and thus the mirrored point will always stay behind the line.

polar

Creates an UMap intended for use on modulatable point parameters. It converts the point into a polar, with linear controls for rho and theta, which on their turn can be used to assign other UMaps to.
rho: the rho value (distance from 0@0)
theta: the theta value (angle from 0@0)

rho_theta

Creates an UMap intended for use on modulatable point parameters. It converts the point into a polar, with linear controls for rho and theta, which on their turn can be used to assign other UMaps to.

rho: the rho value (distance from 0@0)

theta: the theta value (angle from 0@0)

rotate_point

Creates an UMap intended that rotates and scales an input point.

point: the point to rotate (can be an UMap)

rotate: the amount of rotation in degrees (-pi - pi)

scale: a scale amount (0.25 - 4)

sample_and_hold_point

This UMap implements sample-and-hold process for points. Every time a trigger is received the output point becomes the input point of that moment.

point: the point or point UMap to sample

trigger: the trigger

time: a linear lag time to move to the new point

scale_point

Creates an UMap for scaling and moving a point. The point can also be UMaps.

point: the first point

scale: a multiplier for x/y

move: a point to add to the point

select_8_point

Creates an UMap that can select one from 8 points.

index: the index of the point (0-7)

interpolation: interpolation type (0:step, 1:linear) // cubic: todo

point0 - point7: the values of the steps (can be UMaps)

range

center_focus

Creates an UMap intended for use on modulatable range parameters. It converts the range into three linear controls: center, focus and knee. If the center value is at minimum or maximum, the low and high values of the range will be the same. The focus parameter controls the behaviour of the range in between. If focus == 1 (maximum), the range values will be equal to center. If focus == 0, the range will be [min,max] when the center value is halfway in between. The knee parameter controls the smoothness of the transition via sine mapping.

```
center -> range
with focus == 0;
0      -> [ 0,    0    ]
0.25   -> [ 0,    0.5  ]
0.5    -> [ 0,    1    ]
0.75   -> [ 0.5,  1    ]
1      -> [ 1,    1    ]
with focus == 0.5;
0      -> [ 0,    0    ]
0.25   -> [ 0,    1/3  ]
0.5    -> [ 1/3,  2/3  ]
0.75   -> [ 2/3,  1    ]
1      -> [ 1,    1    ]
with focus == 1; ( same as \center_range with range == 0)
0      -> [ 0,    0    ]
0.25   -> [ 0.25, 0.25 ]
0.5    -> [ 0.5,  0.5  ]
0.75   -> [ 0.75, 0.75 ]
1      -> [ 1,    1    ]
```

center: position above or below the center (-1 - 1)
focus: the width of the range (0-1)
knee: transition smoothness at point where lo or hi value starts changing

center_range

Creates an UMap intended for use on modulatable range parameters. It converts the range into two linear controls: center and range, which on their turn can be used to assign other UMaps to.
center: the center of the range
range: the width of the range (0-1)

value_to_range

Creates an UMap intended for use on modulatable range parameters. The UMap derrives a value range by taking the peak and the lowest value from the input value over time.
value: the input value (or UMap)
decay: (0-1) a decay parameter for the (low)peak detection. 0 means no decay, 1 means infinite decay

shared_io

shared_buffer_in

** this UMapDef should be used in conjunction with a 'shared_buffer' unit **

The shared_buffer_in UMapDef can receive a buffer pointer from a 'shared_buffer' unit earlier in the chain. This way a single buffer can be played by multiple buffer-playing units (i.e. bufSoundFile, grainSoundFile etc.).

id: the id (0-99) by which the point can be retreived from 'shared_buffer'
rate: the playback rate
loop: loop mode on/off for playback

shared_buffer_out

** this UMapDef should be used in conjunction with a 'shared_buffer_in' unit **

The 'shared_buffer_out' UMapDef creates a unit that is able to share a (soundfile) buffer or UMap with other units further in the chain. The 'shared_buffer_out' unit needs to before the units that use it in the order of the chain. To retreive the value in an other unit use the 'shared_buffer_in' UMapDef, and make sure the 'id' setting is the same on both. The buffer will always be mono (for multichannel audiofiles only the first channel will be loaded).

id: the id (0-99) by which the buffer can be retreived by a 'shared_buffer_in' UMap. In the GUI this will show up as a colored box, which can be dragged and dropped onto the receiving unit.

shared_in

** this UMapDef should be used in conjunction with 'shared_out' **

The shared_in UMapDef can receive a value from UMaps used earlier in the chain. This is useful when multiple units or unit parameters need to use the same information, or derrive things from it. The shared_in would always need to come after a 'shared_out' in the order of the chain; it recieves the data from a bus. To send the data use the 'shared_out' UMapDef in an other unit or parameter earlier in the chain, and make sure the 'id' setting is the same on both. There can be multiple 'shared_in' UMaps with the same 'id' in one chain, which would all receive data from the same 'shared_out'. 'shared_in' automatically maps the incoming value to the range of the parameter it is connected to. I.e. if the 'shared_out' containse a frequency value between 2 and 20000, and the 'shared_in' is connected to an amplitude parameter, the value will be scaled from 2-20000 to 0-1, according to the specs. This scaling can be influenced both with the 'range' parameter on 'shared_out' and the 'range', 'curve', 'invert' and 'clipMode' settings on 'shared_in'. All this will only work _within_ a single chain.

id: the id (0-99) by which the value can be retreived from 'shared_out_point'
range: the range to map the received value to
curve: a curve setting into which the value can be wrapped
invert: (boolean) if true the range is inverted; high becomes low and v.v.
clipMode: if the 'range' on the corresponding 'shared_out' UMap is not full, the sent value may go out of bounds. The clipMode on 'shared_in' defines how to deal witch such values:
0: clip; clip everything between the min and max of the 'range'
1: fold; fold (mirror) values outside the range so that they fall into it
2: wrap; wrap around at the borders of the range
3: none; no clipping; values are allowed to go outside the 'range'
clipSoftness: an amount of softening applied to the edges of the range when in 'clip' or 'fold' mode

shared_out

** this UMapDef should be used in conjunction with 'shared_in' **

The shared_out UMapDef creates an UMap that is able to share a value with other UMaps used further in the chain. This is useful when multiple units or unit parameters need to use the same value, or derrive things from it. The shared_out would always need to be first in the order of the chain; it sends the value to a bus. To retreive the value in an other unit or parameter use the 'shared_in' UMapDef,

and make sure the 'id' setting is the same on both. The 'shared_in' UMapDef will automatically map the value to the range of the parameter it is connected to, which means it is possible to use for example the 'freq' value of one unit to influence the 'amp' value of another (or the same) unit. This will only work _within_ a single chain.
value: the value to be shared (can be an UMap)
id: the id (0-99) by which the point can be retreived by 'shared_in'
range: the range of the value to be shared.

shared_point_in
** this UMapDef should be used in conjunction with 'shared_point_out' **
The shared_in_point UMapDef can receive point information from UMaps used earlier in the chain. This is useful when multiple units or unit parameters need to use the same point information, or derrive things from it. The shared_in_point would always need to come after a 'shared_point_out' in the order of the chain; it recieves the point data from a bus. To send the data in an other unit or parameter use the 'shared_point_out' UMapDef, and make sure the 'id' setting is the same on both. There can be multiple 'shared_point_in' UMaps with the same 'id' in one chain, which would all receive data from the same 'shared_out_point'. All this will only work _within_ a single chain.
id: the id (0-99) by which the point can be retreived from 'shared_out_point'

shared_point_out
** this UMapDef should be used in conjunction with 'shared_point_in' **
The shared_point_out UMapDef creates an UMap that is able to share point information with other UMaps used further in the chain. This is useful when multiple units or unit parameters need to use the same point information, or derrive things from it. The shared_point_out would always need to come first in the order of the chain; it sends the point data to a bus. To retreive the data in an other unit or parameter use the 'shared_in_point' UMapDef, and make sure the 'id' setting is the same on both. This will only work _within_ a single chain.
point: the point to be shared (can be an UMap)
id: the id (0-99) by which the point can be retreived by 'shared_point_in'

trajectory

circle_trajectory

Creates an UMap for generating a circular or elliptical trajectory for modulatable point parameters.
speed: frequency (cycles per second).
startAngle: starting angle of the path in degrees.
clockwise: select to turn clockwise, otherwise the path turns anti-clockwise.
center: the center of the circle (Point).
radius: x and y radius of the circle/ellipse (Point).

line_trajectory

Creates an UMap for generating a line trajectory between two points. The points can also be UMaps.
a: the start point of the line
b: the end point of the line
curve: a curve value (x,y) for the line. A 0 (zero) curve creates a linear line, a positive curve value makes the line tend towards the low value, and v.v.
easeIn: a positive value makes the movement start slow, a negative value makes it start fast
easeOut: a positive value makes the movement end slow, a negative value makes it end fast
duration: duration of the line in seconds (can be modulated)
loop: loopmode (0: off, 1: loop, 2: alternate)
delay: delay time before the line starts
trigger: a trigger that restarts the line

motion_trajectory

Creates an UMap for generating a motion based on speed and acceleration, contained in a rectangular area.
startPoint: start point of the movement, can be changed over time
startSpeed: initial speed, can be changed over time to produce relative speed changes
acceleration: (in m/s^2) amount of acceleration over x and y axes
center: the center of the clipping area
radius: the radius of the clipping area
clipMode: can be one of the following modes:
0 - 'clip': clip coordinates outside clipping area
1 - 'fold' (default): fold/mirror coordinates outside clipping area - this produces a typical hard wall bounce
2 - 'wrap': wrap around coordinates outside clipping area
3 - 'none': no clipping applied (values can go outside clipping area)
clipSoftness: softens the edges of 'clip' and 'fold' modes.
reset: jump to startPoint/startSpeed

random_trajectory

Creates an UMap for generating a random trajectory for modulatable point parameters.
speed: frequency by which new random positions are generated, in Hz.
center: (Point) the center of the rectangle within which a random path can be generated.
radius: (Point) x and y radius of the rectangle within which the random path is generated.
type: the type of noise used:
0: step or sample-and-hold noise; hard jumps at each value change
1: linear interpolated noise
2: cubic interpolated noise
lag: a smoothing time for changes if 'type' == 0.
seed: Use this to generate a different random path. Paths with the same seed are exactly the same.

trajectory

A player for 2D spatial trajectories. The WFSPathGUI; an editor for trajectories, can be called up via the [edit] button in the UChain window. This umap is intended for use on units with PointSpec or WFSPointSpec based args.

trajectory: a WFSPathBuffer object. This object creates and points to a buffer with the data of the trajectory. This buffer is either filled with data sent directly, or data read from a .wfspath file (generated via the [write data] button in the GUI). Data from file is usually more reliable than sending via network.

trigger: a trigger to (re)start the trajectory

addPoint: a point (or point UMap) can be added to the trajectory position, effectively moving the trajectory as a whole.

trigger_test

direction_change

This UMap can switch between two values, depending on an incoming value. If the value is changing upwards (getting higher) the result is 'up', if the value is changing downwards it becomes 'down'. If the value doesn't change, the current state is kept. All parameters can be UMaps.
value: the value to test (0-1)
upValue: the value (or UMap) to output when the value changes upwards
downValue: the value (or UMap) to output when the value changes downwards

greater_than

A comparator. This UMap can switch between two values, depending on two incoming values; a and b. If a > b the UMap outputs trueValue, otherwise it outputs falseValue. All values can be UMaps.
a: the first value
b: the second value
trueValue: the value (or UMap) to output when a > b
falseValue: the value (or UMap) to output when a <= b

in_range

This UMap can switch between two values, depending on an incoming value. If the value is in between the lower and higher value of the range, the result is 'true', otherwise it is 'false'. All parameters can be UMaps.
value: the value to test (0-1)
range: the range (explanation above)
trueValue: the value (or UMap) to output when the result is true
falseValue: the value (or UMap) to output when the result is false

logical

An UMap for applying logical operations upon two binary values.
a: the first value (boolean)
b: the second value (boolean)
mode: value (0-5) for type of operation
0: 'and'; and/&& operation
1: 'or'; or operation
2: 'nand'; nand (not and) operation
3: 'xor'; xor operation (or and nand)
4: 'a'; return first value
5: 'b'; return second value
trueValue: the value (or UMap) to output when the result is true
falseValue: the value (or UMap) to output when the result is false

schmidt

This UMap can switch between two values, depending on an incoming value. If the value passes the upper range boundary, the output will become 'true'. If there-after the value passes the lower range value, the output will become 'false'. The range between the upper and lower value is known as "hysteresis". If the lower and upper value of the range are the same, the UMap will behave as a simple treshold comparator. All parameters can be UMaps.
value: the value to test (0-1)
range: the Schmidt range (explanation above)
trueValue: the value (or UMap) to output when the result is true
falseValue: the value (or UMap) to output when the result is false

trigger_to_value

pulse_counter

This UMapDef implements a pulse counter.
trigger: the trigger to be counted
reset: a trigger resets the count
max: the maximum value to be counted
step: the size of the steps (can be negative too)
startValue: the value from where the counting starts (default value before any pulse is counted)
mode: there are three possible modes:
\clip - clip the output to the max number
\wrap - when the max number is reached, jump to zero
\fold - after the max number is reached, count backwards, and when zero is reached count forwards again
range: the values to which zero and max are mapped.

sample_and_hold

This UMap implements sample-and-hold process. Every time a trigger is received the output value becomes the input value of that moment.
value: the value or UMap to sample
trigger: the trigger
time: a linear lag time to move to the new value

set_reset_ff

This UMap implements a flip-flop switch mechanism.
trigger: the set trigger
reset: the reset trigger
lag: a linear lag time between value1 and value2 in seconds
value1: the first value (or UMap)
value2: the second value (or UMap)

timer

This UMap measures the elapsed time between triggers. Every time a trigger is received, the UMap will output the time between that and the previous trigger. The value can be scaled from a within a time range to the output range of the UMap.
trigger: the trigger
minTime: minimum time, will be mapped to the lower value of 'range'
maxTime: maximum time, will be mapped to the higher value of 'range'.
range: the output range to scale the time values to. If minTime > maxTime the range gets inverted.
curve: a curvature value for the mapped range. If curve == 0, the mapping is linear.
clipMode: can be one of the following modes:
0 - 'clip' (default): clip / limit values outside range
1 - 'fold': fold / mirror values outside range
2 - 'wrap': wrap around / modulo values outside range
3 - 'none': no clipping applied (values can go outside range)
clipSoftness: only used for 'clip' and 'fold' (0-1).

toggle_ff

This UMap implements a toggle flip-flop mechanism. Each time a trigger is received the UMap switches to the other value.
trigger: the trigger
lag: a linear lag time between value1 and value2 in seconds
value1: the first value (or UMap)
value2: the second value (or UMap)

trig_random

This UMap implements triggered random value generator. Every time a trigger is received the output value a random value within the set 'range' of the UMap.
trigger: the trigger
range: the range of the value
time: a linear lag time to move to the new value

trig_select_16

Creates an UMap that can select one from 8 values via a trigger(or UMaps). When the end is reached, it wraps around to the first value. If you need to select from > 8 values, use \trig_select_16 instead.
trigger: switch to next value
reset: return to first value
step: step size
reverse: if true, step backwards instead of forward
range: min and max index to choose from
lag: lag time between steps (linear lag)
value0 - value7: the values of the steps (can be UMaps)

trig_select_8

Creates an UMap that can select one from 8 values via a trigger(or UMaps). When the end is reached, it wraps around to the first value. If you need to select from > 8 values, use \trig_select_16 instead.
trigger: switch to next value
reset: return to first value
step: step size
reverse: if true, step backwards instead of forward
range: min and max index to choose from
lag: lag time between steps (linear lag)
value0 - value7: the values of the steps (can be UMaps)

trigger_utility

dust

Creates an UMap for producing random single-frame impulses.
density: average number of impulses per second
range: the output range

impulse

Creates an UMap for producing single-frame impulses
freq: the frequency of the impulses wave
phase: the initial phase (0-1) of the pulse wave
range: the output range

pulse_divider

This UMap implements a pulse divider. The incoming trigger pulses are counted and every 'division' times one is passed through.
trigger: the trigger
division: number of pulses to divide
offset: start offset for pulse division (0 means first trigger always gets through)
time: the time to hold the trigger. If a new trigger occurs within this time it is ignored.
range: the output range; a passed through trigger makes the output jump from the range minimum to maximum.

trigger

This UMap implements a simple trigger signal with a certain duration. For convenience also a linear rise/decay filter, so that users can use this UMap for simple envelope creation
trigger: the trigger

time: trigger duration (if 0, it will be a single frame)
rise: a linear lag time before the value reaches it's maximum
decay: a linear lag time before the value reaches it's minimum after the trigger time has ended
range: the value range (when triggered, the value moves from the lower to the upper limit of the range)

utility

clip_scale

Creates an UMap for scaling a value to another range, with clipping and wrapping functionality.
value: the value upon which the scaling is applied (this can be another UMap as well)
fromRange: the used range of the input value
toRange: the range to map it to
invert: if true, the input range is inverted (i.e. low values become high and v.v.)
curve: a curvature value for the mapped range. If curve == 0, the range is linear.
clipMode: can be one of the following modes:
0 - 'clip' (default): clip values outside fromRange
1 - 'fold': fold values outside fromRange
2 - 'wrap': wrap around values outside fromRange
3 - 'none': no clipping applied (values can go outside toRange)
clipSoftness: only used for 'clip' and 'fold' (0-1).

crossfade

Creates an UMap for linear crossfading between two values (or other UMaps)
a: the first value
b: the second value
crossfade: (0-1) the crossfading position (a to b)

envelope_map

Creates an UMap for indexing an envelope.
index: the index (0-1, modulatable)
env: an Env or EnvM object, containing time and level values.

gate

This UMap implements a gating process. It allows a modulating input value to pass when gate is on, otherwise holds last value.
value: the value or UMap to sample
gate: if true (on), the signal is passed through, if false the signal is kept at the last value
the value arg range is mapped to that of the parameter to which the UMap is connected.

poll

An UMap that posts incoming values in the 'post' window. The value itself is passed through unchanged.
value: the value to be posted
trigger: a trigger causes the value to be posted
onChange: when true, the value will be posted whenever it changes (can result in many posts)
speed: number of times per second to post the value automatically (default 0 - no automatic posting).

select_16

Creates an UMap that can select one from 16 values (or UMaps). If you need to select from <= 8 values, use \select_8 instead.
index: the index of the value (0-15)
smooth: smoothening parameter (0: hard steps, 1: linear interpolation)
value0 - value15: the values of the steps (can be UMaps)

select_8

Creates an UMap that can select one from 8 values (or UMaps). If you need to select from > 8 values, use \select_16 instead.
index: the index of the value (0-7)
smooth: smoothening parameter (0: hard steps, 1: linear interpolation)
value0 - value7: the values of the steps (can be UMaps)

Appendix B: A walk through the helpfiles

Helpfiles

In the Help menu you can find a few help files. Look under getting started with WFSCollider: here you will find the WFSCollider V2.0 Overview.

You can also get to these helpfiles by opening a SuperCollider window, (menu File > New (Cmd-N)). Copy in this window the name of the helpfile, 'A WFSCollider tutorial' for instance (without the single quotes), and select the whole text. Then go to menu Help > SuperCollider Help (Cmd-D). In this case the window will show the results found. If you copy for instance 'UChain' it will go directly to the help page.

If you want to stop a sound that was generated by code use menu Lang > Stop (Cmd-.).

SuperCollider Help (Cmd-D)

On the top of this window is a 'Quick lookup' field. Here you can type the texts as stated below after 'Type:' to find the currently existing help guides. Not all guides are as useful, but this will be indicated as well as errors that still exist.

WFSCollider

Tutorials

Type: WFSCollider

A WFSCollider tutorial

This is a good startingpoint when beginning to use WFSCollider.
There are a few errors: to open a new Score, go to menu Scores > New.
To add a new Event, click on the + in the Score window.
Actually, an Event is the same as a Chain. The default Chain of an Event is a bufSoundFile and a wfsStaticPoint (this last one is a WFS Udef).
diskSndFile = diskSoundFile.
bufSndFile = bufSoundFile.
To see how wfsCirclePath is written, in the UDefs window, click on the arrow next to wfs_control, and next to wfsCirclePath, click on the icon.
To add a Udef inbetween to other Udefs in the Chain window: drag a Udef from the UDefs window to just below the bottom of the Udef where you want to place it, and watch out for a blue line. Drop the Udef on this blue line.

Guides

Type: WFSCollider

WFSCollider V2.0 Overview

A good introduction to how the software is organised.
A Score inside a Score is the same as a Folder inside a Score.
To open a new Score, go to menu Scores > New.
UScoreLists are still under construction.

WFSCollider Visual Guide

The interfaces of the Chain and Score window explained.
New additions to the Chain window: single window, and code.
Single window opens the Chains of the Events you double-click in the same window, if you turn this off, each Chain will open in its own window.
If you click on code and the Defs are editable, you can edit the code of the plug-in, or just have a look at it.
With the remark about the Score that double-clicking on an empty area changes the start position of the Score is meant that the current play position changes.
The OSC button in the Score window is a new addition that can be used when controlling WFSCollider live.

WFSCollider Panners

A good introduction to the different WFS-panners.
The statement that the decrease in level is -6 dB when doubling the distance for point sources in real world situations is not completely accurate. Inside a building with a lot of reverb this may be as little as -3 dB. It depends on how much the sound that is directed away from you is reflected by the surroundings.

WFSCollider Code interface

An example not for the weak of mind. It uses a list comprehension. The new method of UChain is called with the arguments as an Array: UChain(*arg).
Select all the code and press Enter or Shift Return to see what happens

References

Type: Wave Field

Wave Field Synthesis

The first part of this explanation is wrong because inside and outside in the statements is reversed. The interesting point is not that sounds can be placed behind the speakers, this is also possible in stereo, but that the sound can be placed in front, that means, inside the speakersystem.

Unit Lib

Classes

Type: U

U

This is the 'Unit'. The relation between Udefs and U's is explained and how to set the parameters as well. The U's are combined into a UChain to make sound.
There is also some explanation about the GUI of the Chain window. The U's and the UChains are programmed according to the MVC Design Pattern, so it does not matter from where a paramater is changed, all GUI's will change accordingly.

Type: UChain

UChain

This has a good explanation of UChain and the Chain window.
New additions to the Chain window: single window, and code.
Some of the class and instance methods are described as well.
At the end is a code example. This example shows how to create some Udefs. To try the settings of fadeIn and fadeOut, execute prepareAndStart and release again. The same hold for both examples with dur.
The example with setAudioOut makes no difference because these settings were already made in de declaration of the UChain. To see the effect, go to the Chain window and click on i/o in the toolbar. Now change audio out 0 of one of the sines and then execute setAudioOut.

Type: Mass

MassEditU

At the bottom there is an example in code that shows how you can create 10 sinewave oscilators and show them in a window, and then how to create a MassEditU window to control all 10 sinewave oscillators at the same time.

MassEditUChain

There's a typo in the example at the bottom of this helpfile: \whitenoise should be \whiteNoise.
In the mass-edit window you see 10 Chain windows. In the MassEditUChain window you can control 10 Chains at once.

Type: ObjectWith

ObjectWithArgs

This is the Superclass of U and MassEditU. There is nothing added to this helpfile.

Type: UScore

UScore

There is an example at the bottom of the helpfile. The argument of UScore is formatted as an Array: (UScore(*Function.value)). Here you see how you can generate a Score with code.
Double-click on the first parenthesis and execute the code. You see a Score window. Press play in the Score window.
Now double-click on the second opening parenthesis. Do not press play in the Score window, but execute the last lines of code.

Type: UScoreE
UScoreEditor
Manages the editing possibilities of the Score window. There is nothing added to this helpfile.

Type: USess
USession
There is an example at the bottom of the helpfile. Here you can see how all this knowledge can be combined in a Session window.

References

Type: Unit L

Unit Library
This is the reference helpfile that explains how to startup ULib, Udefs and U's, what smart arguments are, how to set the Specs of the Units, FreeUdefs, MultiUdefs, Multiple Servers, OSC control, UnitRacks and some other things that are also expanded upon in other helpfiles.
In the Arg values example, if you execute the last line of code several times, you can see that the arguments of the sinewave oscilator are observed.

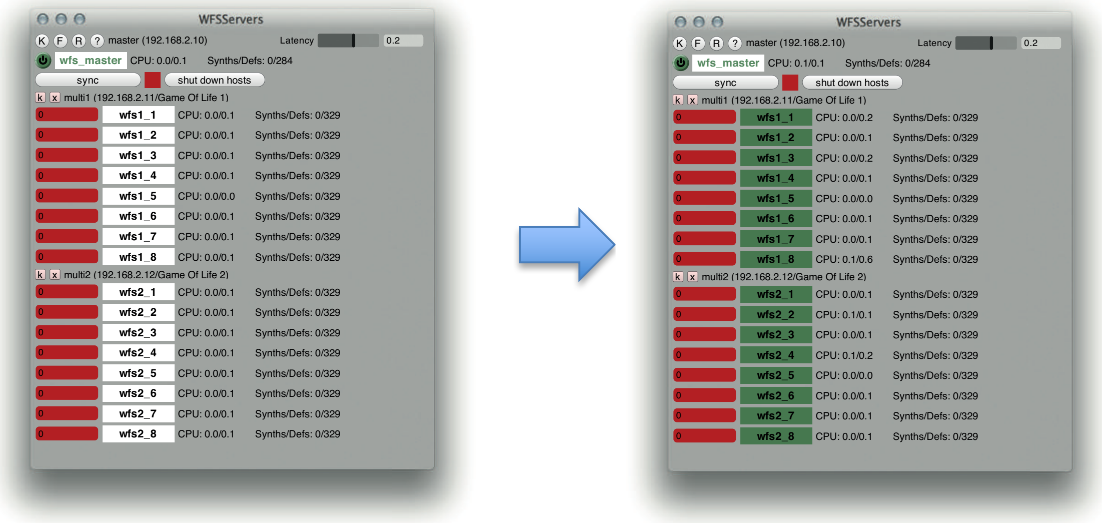
Appendix C:
Working with the Game of Life system

Starting up the system

- Write date, name and time in the logbook.
- Turn ON the switch on the power strip (in a corner) named ‘Master All’.
- Turn ON the switches on the 2 power strips (in both corners) named ‘MOTU + Comp’ and turn on the 2 Servers (in flight cases in both corners) (on the front of the computer: <- ON).
- Make or check the connections to the MacBook:
    - FW400 cable named 'to MacBook'.
    - Blue Ethernet cable.
    - Power cable.
- Start up the MacBook.
- Move with your mouse to the bottom of the screen to open the Dock and click on 'WFSCollider’

- In WFSCollider:

    - Look at the Server Window ('WFSServers') and watch the fields named ‘wfs1_1’ (on the top) and ‘wfs2_1’ (the first of the second 8 rows) turn green. The other ‘wfsX_X’ fields will also turn green, one by one, with apx. 10 seconds intervals. Wait until all fields are green

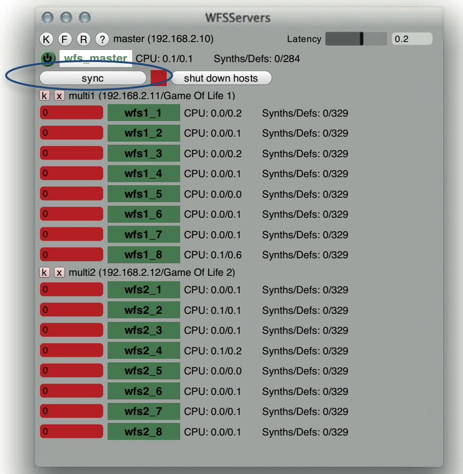


```
WFSServers
K F R ? master (192.168.2.10)                    Latency        0.2
wfs_master  CPU: 0.0/0.1      Synths/Defs: 0/284
sync    shut down hosts
k x multi1 (192.168.2.11/Game Of Life 1)
0    wfs1_1   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_2   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_3   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_4   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_5   CPU: 0.0/0.0    Synths/Defs: 0/329
0    wfs1_6   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_7   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_8   CPU: 0.0/0.1    Synths/Defs: 0/329
k x multi2 (192.168.2.12/Game Of Life 2)
0    wfs2_1   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_2   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_3   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_4   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_5   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_6   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_7   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_8   CPU: 0.0/0.1    Synths/Defs: 0/329
```

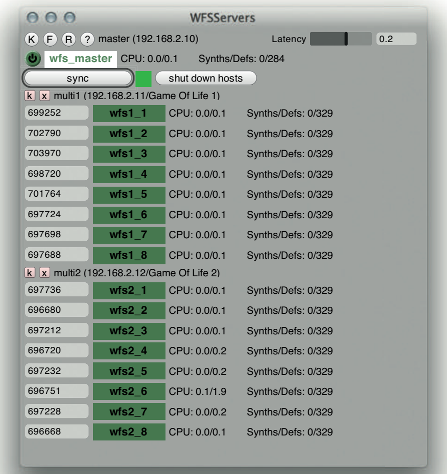
```
WFSServers
K F R ? master (192.168.2.10)                    Latency        0.2
wfs_master  CPU: 0.1/0.1      Synths/Defs: 0/284
sync    shut down hosts
k x multi1 (192.168.2.11/Game Of Life 1)
0    wfs1_1   CPU: 0.0/0.2    Synths/Defs: 0/329
0    wfs1_2   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_3   CPU: 0.0/0.2    Synths/Defs: 0/329
0    wfs1_4   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_5   CPU: 0.0/0.0    Synths/Defs: 0/329
0    wfs1_6   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_7   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs1_8   CPU: 0.1/0.6    Synths/Defs: 0/329
k x multi2 (192.168.2.12/Game Of Life 2)
0    wfs2_1   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_2   CPU: 0.1/0.1    Synths/Defs: 0/329
0    wfs2_3   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_4   CPU: 0.1/0.2    Synths/Defs: 0/329
0    wfs2_5   CPU: 0.0/0.0    Synths/Defs: 0/329
0    wfs2_6   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_7   CPU: 0.0/0.1    Synths/Defs: 0/329
0    wfs2_8   CPU: 0.0/0.1    Synths/Defs: 0/329
```

- Look at the Server Window ('WFSServers') and watch all the way to the left of the fields named ‘wfs1_1’, ‘wfs2_1’, etc.: you see red numberboxes.

Now click on the ‘sync’ button. Repeat this a few times, and if everything is working correctly the numbers you see shouldn't change and the red numberboxes should now become white.



```
WFSServers
K F R ? master (192.168.2.10)                 Latency      0.2
wfs_master  CPU: 0.1/0.1     Synths/Defs: 0/284
sync   shut down hosts
k x multi1 (192.168.2.11/Game Of Life 1)
0    wfs1_1  CPU: 0.0/0.2   Synths/Defs: 0/329
0    wfs1_2  CPU: 0.0/0.1   Synths/Defs: 0/329
0    wfs1_3  CPU: 0.0/0.2   Synths/Defs: 0/329
0    wfs1_4  CPU: 0.0/0.1   Synths/Defs: 0/329
0    wfs1_5  CPU: 0.0/0.0   Synths/Defs: 0/329
0    wfs1_6  CPU: 0.0/0.1   Synths/Defs: 0/329
0    wfs1_7  CPU: 0.0/0.1   Synths/Defs: 0/329
0    wfs1_8  CPU: 0.1/0.6   Synths/Defs: 0/329
k x multi2 (192.168.2.12/Game Of Life 2)
0    wfs2_1  CPU: 0.0/0.1   Synths/Defs: 0/329
0    wfs2_2  CPU: 0.1/0.1   Synths/Defs: 0/329
0    wfs2_3  CPU: 0.0/0.1   Synths/Defs: 0/329
0    wfs2_4  CPU: 0.1/0.2   Synths/Defs: 0/329
0    wfs2_5  CPU: 0.0/0.0   Synths/Defs: 0/329
0    wfs2_6  CPU: 0.0/0.1   Synths/Defs: 0/329
0    wfs2_7  CPU: 0.0/0.1   Synths/Defs: 0/329
0    wfs2_8  CPU: 0.0/0.1   Synths/Defs: 0/329
```



```
WFSServers
K F R ? master (192.168.2.10)                 Latency      0.2
wfs_master  CPU: 0.0/0.1     Synths/Defs: 0/284
sync   shut down hosts
k x multi1 (192.168.2.11/Game Of Life 1)
699252   wfs1_1  CPU: 0.0/0.1   Synths/Defs: 0/329
702790   wfs1_2  CPU: 0.0/0.1   Synths/Defs: 0/329
703970   wfs1_3  CPU: 0.0/0.1   Synths/Defs: 0/329
698720   wfs1_4  CPU: 0.0/0.1   Synths/Defs: 0/329
701764   wfs1_5  CPU: 0.0/0.1   Synths/Defs: 0/329
697724   wfs1_6  CPU: 0.0/0.1   Synths/Defs: 0/329
697698   wfs1_7  CPU: 0.0/0.1   Synths/Defs: 0/329
697688   wfs1_8  CPU: 0.0/0.1   Synths/Defs: 0/329
k x multi2 (192.168.2.12/Game Of Life 2)
697736   wfs2_1  CPU: 0.0/0.1   Synths/Defs: 0/329
696680   wfs2_2  CPU: 0.0/0.1   Synths/Defs: 0/329
697212   wfs2_3  CPU: 0.0/0.1   Synths/Defs: 0/329
696720   wfs2_4  CPU: 0.0/0.2   Synths/Defs: 0/329
697232   wfs2_5  CPU: 0.0/0.2   Synths/Defs: 0/329
696751   wfs2_6  CPU: 0.1/1.9   Synths/Defs: 0/329
697228   wfs2_7  CPU: 0.0/0.2   Synths/Defs: 0/329
696668   wfs2_8  CPU: 0.0/0.1   Synths/Defs: 0/329
```

- After 15 minutes, click again on the 'sync' button to check if the system is stable.
(if something goes wrong, check for loose cables)
(if the CPU in the Server Window ('WFSServers') in WFSCollider gets over 100% at some point, click again on the 'sync' button to restore synchronization)
- If everything is in working order, turn ON the switch on the power strips (in both corners) named ‘Amps’.
- On the Desktop click on ‘WFSSoundFiles MOUNT.command’. A volume appears on the Desktop named 'WFSSoundFiles'.
The system is now in operation.

Working with the system

If this is your first time on the system see the chapter ‘Putting files on the system’.
If you have already made a score:
go to the menu Scores en select Open.... Go to your own folder, choose the score you want to load and click on Open.
If you want to work on a new score:
In WFSCollider in the menu Scores select New. In the Menu Scores select Add Event (Shift-Option-Command-A), or click the (+) button in the Score Editor window, and next select the Event in the Score Editor and click on the (i) button in the toolbar (or in menu Scores select Edit (Option-Command-I)). In the UChain Window, go to soundFile and click on the Folder icon. Go to your folder, select a sound file and click on Open.
Click on the Play buton at the bottom of the Score Editor and you are ready to go.

Putting files on the system

In the Desktop click on ‘WFSSoundFiles MOUNT.command’ (if you have not done this yet). A volume appears on the Desktop named 'WFSSoundFiles'.
Doubleclick on the volume ‘WFSSoundFiles’.

If you have not done this before, make a folder in the volume 'WFSSoundFiles' with **your own name**.
Drag your soundfiles in this folder (audiofiles, scorefiles, etc.). Inside this folder you can use any folder organization you like. If you are finished with your project, clean up the folder, since we do not have infinite amounts of disk space...
In the Desktop double-click on 'Distribute soundfiles.command'. The files are copied from the folder 'WFSSoundFiles' on Server 1 to the folder 'WFSSoundFiles' on 'Server 2'. (If you already have files with the same name in the same folder on 'Server 2' they will be replaced if changed).
Problems:
Sometimes the mounting of the disk does not work correctly resulting in warnings in the Post window that the path could not be found. Restarting the MacBook should solve this problem.

On your own computer

If you create your score files at home using WFSCollider, make sure that you put keep your whole project in one folder. The score file(s) should be at the top level of this folder. Sound files and other data files used for the score should all be in the folder as well, possibly in subfolders. This way you can move your project to the system by copying that whole folder. All file references in the score are saved as relative references.
If you use a sound file in your score that is located somewhere else on your disk, you can use the operations -> copy to... function in the small menu that appears in the UChain window. With this function you can copy your soundfile to the folder where your score is in. Make sure to save the score again after doing this. There is also a function named “copy all”, which appears if you press the (i) button in the Score Editor while nothing is selected, or while a group of events is selected.

Shutting down the system

- Turn OFF the switches on the powerstrips named ‘Amps’ (in both corners).
- Unmount the disk ‘WFSSoundFiles’:
    select 'WFSSoundFiles' on the Desktop and type Command-E, or drag it to the eject icon in the dock *)
- In the Server Window ('WFSServers') click on 'shut down hosts'.
    - An Alert appears. Click on 'Shut Down' in the Alert.
    - Another warning appears: ‘Are the AMPLIFIERS switched OFF?’. If the amplifiers are switched off, click on ‘yes, shut down now’.
    - Wait until all the green fields in the Server Window have turned white, and the Servers have shut down (you hear a clicking sound).
- Turn off the MacBook.
- Turn off the switch on the power strips named ‘MOTU + Comp’ (in both corners).
- Turn off the switch on the power strip named ‘Master All’ (in the corner).
- Write a report of your findings in the logbook.