

Playnote Window

In the **playnote** window you put the actual data for accessing the soundfiles. Each **playnote()** command is usefully thought of as a **note**. It instructs the sound driver to play part or all of a soundfile at a given time, on a given track, with certain properties.

Each **playnote()** command must appear on a separate line in the **playnote window**. It needn't begin in the first column.

Each **note** as specified by a **playnote()** command has the following properties, specified with these names:

skip= : amount of time to skip into that soundfile before beginning to read it. Default is **skip=0**.

at = : time in the output mix to start playing that segment of the soundfile. Default, **at = 0**.

track= : the track number that soundfile will occupy. Note that only one sound can be played on a track at a given time. There is no default, the track number must be specified. If two playnotes occupy the same track at the same time, or overlap for a period of time, the one that was entered second will destroy the overlapping contents of the first one. Note that this is important if you are using the **load playnotes** button in the **control window**.

snd= : the actual number of the soundfile, as referenced by its position in the soundfile window. There is no default, the snd number must be specified.

dur= : amount of time to play that segment. The *Default* is to the end of the soundfile. When the **transp=** argument is invoked, a positive dur value will indicate the amount of time to read the input file, so that for downward transpositions the actual duration will end up being longer, and for upward transpositions, shorter. If the argument is a negative value it represents the absolute amount of time to play that note, regardless of the transposition (e.g. it represents the output rather than input time.)

end= : the time on the input file to stop playing. i.e. $\text{end} = \text{skip} + \text{dur}$. It can be specified instead of dur. The *Default* is the end of the soundfile. If **transp** is in effect and the argument is positive it represents the end on the input (as with the dur argument), and if negative it represents the end time on output, relative to the starting time of the note.

transp=: this is a single valued gliss function. e.g. **transp = -2** is equivalent to `gliss(0,-2,1,-2)`. It will simply transpose the whole passage down by 2 semitones.

amp(time,amp,time,amp etc): an envelope for the note which will control the amplitude as it plays. The times are normally relative to the length of that note (unless you use a **stickpoint**, see below) so any arbitrary timescale can be used. The maximum amplitude

which can be used is 4. (This is caused by the fact that we don't have time to do floating point multiplies.) Fractional values can be used. Note that this value is multiplied by the gain factor Default: all amps set to 1. Note that you just specify numbers within the parentheses. e.g. **amp(0,0, 1,.5, 2,0)** will create an envelope which goes from 0 at the beginning of the note, to .5 in the middle, to 0 at the end.

gliss(time,interval,time,interval etc): this will cause the apparent sampling rate of the sound to change over time. Note, however, that simple interpolation is used so spectral artifacts will be created. *Default:* no gliss. (Note that transposition up will increase the data transfer rates.) The interval arguments are given in semitones (with fractional parts allowed) up or down. For example, **gliss(0,1,1,2,2,-1.5)** will cause the note to start one semitone higher than the input file, go up to two semitones higher in the middle of the note and end 1.5 semitones lower.

If you have both a **gliss()** and **transp=** argument, then they will be added. All pitch values in the **gliss()** curve will have the **transp=** value added to them.

ampl(time,amp,time,amp, etc): a separate amplitude control for the left channel. Default: set to 1. Syntax and time scaling are the same as for the **amp()** command.

ampr(): is similarly a separate control for the right channel. Default, set to 1.

STICKPOINT: This feature allows any of the above commands which have a string of time/value arguments to use absolute time rather than relative time, for the beginning and end of the notes. The **stickpoint** is a | sign. You specify it in the middle of a set of arguments to "spread" or "shrink" the middle of the note, while leaving the times around the stickpoint fixed.

amp(0,.2,2,.5, | , 9 1, 10, 0) for example, means that this note will have a rise from .2 to .5 over the first 2 seconds, and have a decay from 1 to 0 over the last 1 second of the note, (assuming its duration is at least 3 seconds), and between these two times the amplitude will go from .5 to 1. If, for example the note is 20 seconds long, this is equivalent to saying **amp(0,.2,2,.5,19,1,20,0)**.

If there were not a stickpoint in the above example and the note were 20 seconds long, it would be equivalent to saying **amp(0,.2,4,.5,18,1,20,0)**. Here you can see that the times are warped relative to the actual duration of the note. There can be only one stickpoint in an argument set, but there can be any number of argument pairs around it.

gain=: this will **override** the gain specified in the **Soundfiles** window for that note only. The maximum total gain is 4. Be careful to watch that the product of all the different specifiable gain factors does not exceed 4.

pan= : Normally a mono input file will be split evenly between the two stereo output channels. A stereo input file will be mapped channel by channel. If you specify the **pan** argument this is equivalent to multiplying the gain of the left channel by the pan value and

on the right channel by 1.0 - the pan value. It is equivalent to using **ampl()** and **ampr()**. In other words, for a mono signal **pan=.1** is equivalent to shifting the signal so that it appears to be about 10% of the way between the left and right channels. If you are using the **pan** feature the amplitude of the signal will be boosted to accomodate the loss in power a signal has when it is shared between two speakers. In this case therefore do not specify a gain factor greater than about 2.8, since the gain of a signal with a **pan** of .5 will be multiplied by the square root of 2, to compensate for the power loss. The formula used is $1./\sqrt{\text{pan}*\text{pan} + (1-\text{pan})*(1-\text{pan})}$. These arguments can be placed in any order.

rev: This word by itself, (or **reverse**) will cause the segment to be played backwards. (Thanks to Pete Yadlowsky for this hack.)

playnote(snd=1,track=1,skip=2.5,at=3.5,amp(0,0,1,1,2,0),dur=4)

will cause snd 1 to be played on track 1, starting at 2.5 seconds into the sound, starting at time 3.5 in the mix, for 4 seconds with an envelope going from 0 to 1 and back down.

playnote(track=1,snd=1,transp=-2.5)

will simply play all of snd 1 on track 1 down 2.5 semitones.

The commands in the **playnote window** are initially loaded by hitting the **Load Driver**, or **Load Playnotes** buttons in the control window. If some of the text in the **playnote window** is selected, only that segment will be loaded. You can subsequently select a subset of the **playnote()** commands in the **playnote()** window to be added to the mix, or altered within the mix by simply selecting them with the cursor and hitting the **Load Playnotes** button in the control window. This will save a bit of time since it sometimes takes a few seconds to load in a bunch of files and playnote commands. Note that if a subsequently loaded playnote command occupies the same time portion of a track as a previous one, then the first one will be eliminated.

Some Secret Lansky Features

I put the following two arguments in to make my life easier, but I am not sure they will be useful for everyone. Since they are there, however, I might as well tell you about them

last=: This is a substitute for the **at=** command. It will remember the *ending time* of the last **playnote()** and begin this playnote at that time plus the value set by **last=**. So, for example, if you specify **last=0**, the current note will begin exactly where the previous one left off (to within the current timescale), or **last=-1**, will set the current note to begin 1 second before the previous note ends. This is quite useful if you are doing splicing. You can create a butt-end splice and then bevel and overlap without having to do any arithmetic.

overlap=: This is the same as the **last=** command except that it worries about the beginning of the previous **playnote()** rather than the end. So **overlap=0** will start the current **playnote()** at the same time as the beginning of the previous **playnote()**.

When either of these arguments is used the actual computed **at** value will appear in the output window. I recommend pasting this into your playnote, replacing the **last** or **overlap** definitions, to avoid too much confusion. Things can get messy, particularly in connection with the **offset=** command (see below).

Additional commands which can appear in the Playnote window

These arguments are specified on separate lines, not within a **playnote()** argument list, and will effect all **playnote()** commands which follow.

offset=x

This will result in the value **x** being added to the **at=** value of any **playnote()** which follows this statement. This is a convenient way to alter the starting time of contiguous groups of playnotes.

addoffset=x

This will result in the value x being added to the currently loaded **offset** value. This is a convenient way to increment or decrement currently defined **offset** values.

trackgain(starttrack,endtrack, time1,amp1, time2,amp2... timen,ampn)

This will put a global amplitude curve on all notes occupying the tracks between and including **starttrack** and **endtrack**. Here the times are absolute. For these tracks, any referenced times before **time1** will be assigned **amp1**, and after **timen**, will be assigned **ampn**. The default amplitude for all tracks is 1. For this feature it is best to use amplitudes greater than 0 and less than or equal to 1. This will keep you out of trouble with respect to amplitude overflow since these values will be multiplied by the gain set on individual tracks in the track window, the gain for each sound, and the amplitude settings for each playnote. There can only be one **trackgain()** call for any track or subset of tracks. (I'm working on a better system.) **Note:** you do not use names in this call, just numbers.

tempo=x

This statement will cause all times subsequently referred to by **playnote()** **at=** statements to be multiplied by a factor. If x is less than 15 it is a simple multiplier. If greater than 15 the factor is formed by $60/x$, so standard tempo speeds can be used.

comments

any text on any line which is preceded by **//** will be ignored by the driver. e.g.

//this is a comment

playnote(track=1,snd=1)*//this is another comment, but the playnote is taken seriously*

A cute trick

The white/black text at the top of the **playnote** window is actually a template which you can use and edit to load in playnote commands. In the **Edit** submenu there is an item which will copy this to the bottom of the **playnote** window, so that you can edit it, adding your own parameters instead of typing all the information again and again. You can edit the template in the black section to customize the format to whatever you currently need.