in

COLLABORATORS			
	TITLE : in		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 9, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

in

1.1 The MIDI Modules

The following modules are available in the Music Web. The name in brackets is the actual filename of the module (in the MODULES subdirectory). All the modules except 'MIDI In' and 'Delay' are standard 'Filters' with a straight-through data path; they can accept up to three input path connections, but the packets that arrive on them will all emerge from the common output connector. 'MIDI In' is a 'Source', with a single output connector only. 'Delay' is a multiconnector element -- see its description.

* Read events ('messages') from MIDI In and supply them to a Web path
  MIDI In
* Send events arriving on a Web path to MIDI Out
  MIDI Out
* Change Volume of MIDI 'Note On' events
  Loudness
* Set all (channel-specific) events on the path to selected channel
  Set Channel
* Filter out (block) selected types of MIDI events
  Msg Filter
* Filter out selected notes of the scale
  Key Filter
* Pass only the selected Keyboard note range
  Key Range
* Transpose all notes up or down a specified number of semitones
  Transpose
* Block all 'System' type MIDI events (hex code Fx)
  Block Sys Msg
* Join up to three Web paths into one
  Join
* Generate a delayed version of an incoming event stream
  Delay
* Play the amiga's audio as a MIDI instrument (using 8SVX voice)
  Instrument
* Monitor (display in a screen panel) events on a path
  Monitor MIDI

Element 'Control Panels' are opened by clicking on the element while in 'Param' mode, or choosing 'Param' from the 'Select' mode panel

(see Control Buttons in the Reference.GUIDE). Each panel has a default position in which it will appear (and will overlay other instances of the same panel if not moved). If you move it to a different part of the screen it will remain there, even if closed and reopened.

Remember that each module is a separate program that must be run (normally by the startup script) for the Web to be able to access it. For more information on the relation of Modules to the Web as a whole, please refer to the main Reference.GUIDE .

* Some modules not shown in the buttons (MIDIBranch, FileReq, Load8SVX) should also be running: Ancillary modules

1.2 MIDI In

MIDI In [MIDILink]

Converts MIDI messages ('Events') arriving at the Serial Port into Web packets for processing by other elements. It is obviously the first element in a configuration. Also, only one instance may be placed at a time (unlike most other elements of which there may be many instances). Both this and 'MIDI Out' are actually in one module. It has no control panel. This revision does not yet handle 'System Exclusive' MIDI messages (Sorry -- but my present setup doesn't generate them...) Any SysEx sequences that arrive will be ignored (untested).

1.3 MIDI Out

MIDI Out [MIDILink]

Converts (relevant) Web packets arriving on an input path to MIDI messages which it sends out the Serial Port. As with 'MIDI In', only one instance may exist at a time, but more than one path (up to three) may converge on it -- and of course still other paths may have merged earlier in the diagram. Like other 'Filters' it has an output connector, so potentially it need not be the last element in the chain either; at the moment, about the only element that you could meaningfully connect beyond it is the 'Monitor'. 'MIDI Out' is with 'MIDI In' in a single module, and has no control panel.

1.4 Loudness (MIDI volume)

Loudness [Loudness]

Reduces the volume of Note events passing through it according to the slider setting in its control panel. '128' means no effect, '1' means that all notes will be reduced to minimum volume. No volume is ever brought down to zero unless it is already there, because you usually don't want played notes ever to *completely* disappear.

1.5 Set MIDI Channel

Set Channel [SetChan]

Sets the MIDI channel of all (relevant) events passing through it to that set in its control-panel slider. It does this regardless of the channel they might have on arrival. If you want it to apply only to certain channels or events, you should filter that path to contain only those events beforehand. (However, the current set does not contain a Channel Filter! With a single MIDI event source (i.e. keyboard) it isn't really needed, but it might make sense in more complex setups.)

1.6 Message Filter (blocks specified event categories)

Msg Filter [MsgFilter]

Blocks events of specified categories from proceeding further along that path. There is a button for each class of MIDI message in its control panel. When a button is 'checked' that class of message is blocked. Be aware that "Note Off" is a special case, as there are two conventions for this (a 'Note Off' message class or a 'Note On' with a volume of 0). This filter treats both these as "Note Off".

1.7 Key Filter (block specified notes of scale)

Key Filter [KeyFilter]

Blocks Note events corresponding to particular keys within the octave from proceeding further along the path. There is a button for each of the twelve notes in the octave; 'checked' means blocked. Each button applies to all octaves equivalently; you can achieve other effects by combining with the 'Key Range' filter, say, in multiple parallel paths.
Events other than notes are always passed unaffected.

1.8 Key Range limiting

Key Range [KeyRange]

Sets upper and lower limits to Note events that will be passed. The values set in the two sliders on the control panel refer to MIDI note numbers (60 being middle C). Use complementary settings in parallel paths to create a simple keyboard split; multiple and overlapping 'splits' may also be useful. Events other than notes are always passed unaffected.

1.9 Transpose notes

Transpose [Transpose]

Shifts Note events up or down by the number of semitones specified by its slider. The range is plus or minus two octaves (24 notes). Be aware that you may go off the end of your instrument's range with large shifts; the result might be ugly, pleasant, or just silence... (If the shift would go outside the MIDI range of 0..127, the original will be left unchanged.)

1.10 Block System Messages

Block Sys Msg [NoSysMes]

This is a simple way of blocking out unneeded 'System' MIDI messages from a path when you aren't concerned with them. In many setups these can be quite frequent ('Clock' or 'Active Sense' events for instance) and could create an unnecessary load on a multipath configuration. There's no control panel -- just place it in a path if you want its effect.

1.11 Join Streams

Join [Joiner]

A 'null' filter that simply lets you merge paths when it would be inconvenient to do so on any particular type of filter. It has very little load on activity.

1.12 Delay Events

Delay [MIDIDelay]

A multiconnector element that generates copies of arriving events after a specified delay. It has a single input connection, and two outputs. The default output is a Source of delayed events; the other is just the continuation output for input events so that they may be fed to other elements as well.
Short delays can be used to 'fatten' sounds (provided that your instrument supports multiple voicing of the same note, or you can feed it to a different channel); longer ones can give you 'multiple beats'.
Two particular points: only *one* immediate path may be connected to the input (any merging must be done on filters earlier in the path), and delayed events are *copies* of the originals, so both are available for further use (e.g. merging).
Delay is specified by the control panel slider in 600ths of a second, up to one second. 'Cutout' disables the delayed events if selected (the originals continue as usual).
Normally, 'System' messages are *not* copied to the delayed stream: if for some reason you should need these, select 'Sys Mes Too'.
[If you have a lot of delay elements in a configuration, you might want to raise the module's process priority when started (to about 30), though in most cases -- unless perhaps on a slower machine -- this won't be necessary. If you do this, beware of locking yourself out with a zero-delay loop!]

1.13 Amiga Audio `Instrument' (plays 8SVX IFF files)

Instrument [MIDI_Instr]

This element lets you play the Amiga's own audio channels as a MIDI-controlled instrument (or instruments -- you can have several elements active at once, subject to the hardware-imposed overall maximum of four simultaneous separate sounds). You need an 8SVX IFF file as the source of the instrument's voice; this can be loaded by clicking on the 'New...' button, which invokes a file requester through which you can specify the file you want. (Note that two 'server' modules -- FileReq and Load8SVX -- must be running, otherwise the request will not work.)
The module responds to Note On and Note Off messages, and also to 'Sustain pedal' messages (MIDI Controller 64) if the 'Sustain' gadget is set to 'Pedal'. If the gadget is set 'Off', pedal messages are ignored, and notes are silenced by Note Off events; if 'On', Note Off messages are ignored as well -- effecting a permanent Sustain [this is just the thing for percussion, which sounds ugly if cut off abruptly, but you probably don't want to use it with instruments that repeat indefinitely!]. If, for this or other reasons, you get a 'stuck note', clicking

'Shut Up!' will turn off all sounding notes (in that instrument).

As only four voices can be sounded at once, the module must make decisions as to which ones. The rule followed here is simply to discard the oldest (regardless of instrument) when a new sound needs to be added and there is no free channel. Channels are not tied to a particular instrument: any channel may be used for a note as it becomes available.

If the instrument voice in use is unable to play high octaves (a function of how the file was recorded), the module will shift those notes down one or more octaves into the playable range. If you find this objectionable, insert a suitable Key Range limiter into the preceding path to silence those notes altogether.

(The module makes no attempt to play samples longer than those that can be handled directly by the Audio Device, as an instrument file will usually be well within this restriction. The limit is 128K each for the oneshot and repeat parts.) At present, it ignores any ATAK and RLSE chunks the instrument may provide. Certain purported 'instrument' files may not follow proper conventions, and therefore may not get played as they are intended to be. See 8SVX for more...

This module accepts an optional startup command line parameter (most do not) to specify the default directory in which instrument files are stored.

1.14 Monitor MIDI Events

Monitor MIDI [MIDIMon]

This element lets you observe the MIDI events passing through it (provided that they don't come faster that it can handle). Its panel has no control gadgets -- just displays of the attributes of arriving events. The bottom line gives a complete numeric breakdown of the event (number of bytes and their contents in decimal and hex). The two left hand displays above it show a more human-oriented description of event types. The lower of these is devoted solely to 'Active Sense' messages, showing when these are arriving; all the other displays ignore these, so that they don't get overwritten before you can read them. (My setup does not generate 'Clocks' -- I guess these will have the same problem...)
The upper display shows the class of messages other than these. The two displays to the right of these are devoted only to note events, showing note number and volume respectively.

1.15 Ancillary Modules

```
Ancillary Modules
=================

There are a few other modules besides the Web itself and the visible MIDI
modules that must be running for a complete system.  (And don't forget
that the Traveller module must be resident for any messages at all to
flow!)    Package Components      Travelling the Web


MIDIBranch      is the module that manages branch points in the diagram;
                you will not be able to place them if it is not running.
                (It is not quite the same as the 'standard' Web Brancher,
                as it is adapted to speed the processing of MIDI events.)
                This module should be included in the Web startup script.


FileReq         is a server that provides a standard File Requester (only
                for the Instrument module currently).  Having this as a
                separate module prevents a traffic jam when you open
                a requester.
                This module should be included in the Web startup script.


Load8SVX        is the server that actually reads an 8SVX file into
                memory on request from Instrument (or other modules
                later).  [The current version does *not* know about
                packed sound files; this is an omission, but I haven't
                yet actually encountered any packed ones...]
                This module should be included in the Web startup script.
```

1.16 8SVX Instrument Files

```
 8SVX 'Instruments'
====================

    The 8SVX 'Instrument' format seems to be a very confusing one
    -- not least to those who have previously implemented players
    for such files!  Both the Sequencer and Sampler software I own
    -- both of which shall remain anonymous here -- make a complete
    mess of it.  So far I've only found DMCS 2.0 (I have the demo)
    that seems to get it right.  As a result you may well have some
    supposed 'instrument' samples that won't play properly.  This
    outline may help you to understand any problems.

    The main components of an 8SVX file are its 'header' and 'body'.
    (There are also 'attack' and 'release' segments, which may be
    included, but they don't seem to be respected much at the moment
    -- even by DMCS... the demo samples have them, but produce no
    detectable effect. I won't discuss them further.)  The body
    contains blocks of waveform data for one or more octave ranges
    of the instrument; the header describes this data so that it can
    be played at the correct pitch and so on.
```

The waveform data is a representation of the sound produced by the original instrument, sampled at a specific rate to produce a sequence of numeric values; if these are played back at the same rate, you get (a reasonable facsimile of) the original note. If you play the sample back at a different appropriate rate you'll get another note; hence the computer can generate a complete musical scale from just one block of data.

To decide at exactly what rate you need to play it back to get the desired note, all you really need to know is the number of samples for each cycle of the fundamental frequency. You know the cycles per second for the note you want to produce (e.g. 261.6 Hz for middle-C) so the playback rate will just be this times the number of samples per cycle. A proper Instrument file should therefore have this information in its header. [And the above nameless Sampler software give the user no means of supplying this -- my reaction to this discovery is probably best left to the imagination...]

So what's this about 'octave ranges'? Why can't we just use one original sample and produce all the notes we want? Well, there are two problems, only one of which is really solved by the 8SVX scheme. The one that *isn't* is that a real instrument probably produces a different waveform for each note -- not just the same mix of frequencies shifted up or down a bit. The ultimate sample format would probably have one waveform for each individual note of the instrument's range (as I believe a lot of synths in fact do) but this would be pretty heavy on memory. A reasonable compromise is to have one sample block for each octave, and to restrict the range for each to just its own octave. This *can* be done with 8SVX but there are other restrictions, which I won't go into, that may make it difficult to do. [Especially with the software I have seen...] Nevertheless, some available instrument files seem to have a good sound over their range, and probably do it properly.

The other part of the problem is that if we record a note at a certain sample rate, to play it an octave higher we have to use a playback rate twice as fast. If the original sample rate is adequate to give good quality we won't be able to increase it all that far before we run into the harware limits of the machine. To go any higher we need another set of data with a more suitable sampling rate. So, typically we will provide a series of samples, each of which can take over from the previous one as the playback rate reaches its limit. In 8SVX these samples must be an octave apart, and each must be exactly half the size of the one preceding. Often each is produced simply by chopping out every alternate value from the preceding block.

One thing 8SVX doesn't explicitly specify is which octave range should be played at which physical octave -- say relative to middle-C. You *could*, if you wanted, play a waveform with two sample per cycle at middle-C, using a slow enough rate; you wouldn't get much tone-colour, though! However, it turns out -- given the system clock rate and audio limits of the Amiga -- that there is a convenient set of 'standard' playback rates for the notes of the scale; using these, middle-C and its octave (octave 5 of the MIDI

range) have 32 samples per cycle, and hence a 2 samples-per-cycle waveform will play at MIDI octave 9.

On this basis, and assuming that the most detailed sample possible should be used for a note, you can make a suitable choice of octave data. This is what MIDI_Instr does. However, both the sequencer and sampler I have make other idiosyncratic choices -- a flute can end up sounding quite like an accordion!

One thing I do not yet do is take any account of the *original* sample rate (available in the header). It is possible for the basic sample to have been recorded at a higher rate than the 'standard' playback base rate, so that it is never quite played at the original fidelity. It should be possible to have a scheme that extends the set of playback rates in such a case.

The remaining point you should probably be aware of is the difference between 'one-shot' and 'sustained' instruments. The waveform data can have two parts (either of which may be missing): an initial 'oneshot' section that is sounded just once when the key is struck, and a 'repeat' section that will cycle continuously as long as the key is held down (or while 'sustain' is on if that feature is enabled). Things like drum and piano sounds normally only have a oneshot part, while an organ will certainly have a repeat part but might not bother with a oneshot. Electric guitars need the oneshot, but usually also have an indefinite sustain using a repeat.

For the best effect, you should choose the 'Sustain' mode to suit the instrument. Drums sound far better if every beat is allowed to run to completion, so Sustain should be 'On'. Pianos of course should have the 'Pedal' mode set, but even guitars with repeat sections sound good with a judicious use of pedal. For proper simulation of organs and the like, Sustain should be off, but again occasional use of the pedal may be attractive.

+ + + + +