Chapter 1

User's manual

someone should really write this!

1.1 Basic overview

1.1.1 Tracks

MusE arranges your music in *tracks* and *parts*. The following section shall provice you an overview of how things are done with MusE. If you are or were a Cubase or Cakewalk user, you will feel familiar with this. There are MIDI and drum tracks (which are internally MIDI tracks) which can hold note data, wave tracks which hold wave data, and several kinds of audio tracks, synthesizers etc.

MIDI tracks MIDI and drum tracks hold MIDI event data. They don't differ much, except that drum tracks offer a special editor which is more suitable for drum editing.

Wave tracks They hold audio data which can be just played back or be piped through effect plugin chains. They offer automation for these plugins.

Audio inputs These provide the path for your audio data from outside into your project. Set up the physical audio inputs you want to connect your audio in track with, and then route them to some wave tracks.

1.1.2 Parts

Tracks are split in parts. These contain MIDI events or wave chunks. You can copy, clone them, move them around, delete them etc.

1.1.3 MIDI ports

Ports provide an abstraction layer for your MIDI synthesizers (which can be both software and hardware synthesizers). Port are numbered. In order to produce sound, each MIDI track is assigned to exactly one MIDI port, to which the MIDI events are then sent.

In the configuration menu, you must map the port numbers to the actual synth devices (by selecting ALSA or jack midi ports, or synth plugins).

Try left-clicking on the "Ports" column of some MIDI track. If you use a soft synth, right-clicking the Ports column of the synth or any track using the synth lets you launch the synthe's GUI.

1.2 Tracks and parts

1.2.1 Tracks

Creation You can create a track by either right-clicking in the arranger's track list and then adding the desired track, or via the edit menu.

Attributes Tracks have several attributes.

Mute If you click on the *Mute* field (denoted with a "M" column header), the track gets muted and stops producing sound.

Solo The solo button ("S" column header) mutes all other tracks.

Record The R column "arms" your track for recording. When you rec-arm your song and have no tracks rec-armed, you won't be able to record anything. See also the config option "move rec-arm with selection".

Track name Double-click and you will see.

Port For MIDI tracks, this lets you select the MIDI port to which the events should be routed. This can be your physical synthesizer or a software synthesizer. For soft synthes, this is the port the synth is associated to. For other track types, this is disabled.

Channel For MIDI tracks, this is the MIDI channel the output is sent to. For any kind of audio tracks, this is the number of channels (mono, stereo).

Automation For audio tracks, this lets you set up the automation display in the arranger. Clicking this will provide you with a popup menu with lots of submenus. The submenus let you select the color you want to associate with the automation parameter. *Clicking on a submenu* will select or unselect it, making the automation parameter shown or hidden.

Clef For MIDI tracks, you can specify a clef here. This only affects the score editor.

The trackinfo side bar In the arranger and the part editors, you'll have a trackinfo sidebar on the left side. You can set up track-type specific things there.

MIDI tracks

MIDI parts have no automation settings. However, they support various controllers, program changes etc.. The MIDI trackinfo sidebar lets you change program, volume, pan and more. Just editing the value in the boxes will send an event, but not write it to the track. In order to write it, you will need to click on the corresponding button ("Vol" for writing down volume information).

Old style drum tracks

These are MIDI tracks as well, but with a few differences. They allow you to map certain drum sounds with different input notes, and you can change the output settings of a certain "drum instrument" without having to alter each single event.

However, they have certain limitations: They only can handle 128 sounds (even if you have more synthes), they aren't really compatible with MIDI tracks (you can interchange parts between them, but if you touched the drum list, you'll get unexpected results), you can't set a program for the used channel and more.

New style drum tracks

That's why there will be new-style drum tracks in the next development version. They are handled exactly like plain MIDI tracks (staying compatible with them), and offer all of the functionality, though in a different way. They allow you to re-order the drum map efficiently, you can open parts from multiple drum tracks in *one* drum editor (MusE will separate the sounds from different tracks according to your settings, see the "Window Config" menu), and you can set programs as with normal MIDI tracks.

Audio tracks

Effect rack On the top of the sidebar, there is an effect rack which allows you to apply various plugins on the audio. For more information on this, refer to 1.3.

Controls Lorem ipsum

1.2.2 Parts

Within MIDI, drum and wave tracks, you can create *parts*. Parts are chunks of coherent notes or wave data which can be moved around, copied, cloned and deleted independent from other parts.

Parts are created by selecting the pencil tool and then drawing onto the right part area in the arranger. You can move them with the arrow tool, delete them using the `DEL` key, and a right-click opens a popup menu. This menu allows you even more stuff, such as setting the part's color, saving the part to disk etc.. You can use `CTRL+C` and `CTRL+V` for copying and pasting parts. `CTRL+B` pastes the part as a clone. Pressing `SHIFT` additionally provides you a dialog which allows you to paste the part multiple times and set more stuff.

You can also copy parts with the mouse by moving the part with the mouse while holding down the `CTRL` key.

1.3 Plugins and automation

There are several kinds of plugins. First, there are audio plugins, which can be applied to any track handling audio (that is, inputs, outputs, wave tracks, synth tracks). Plugins can be added by double-clicking on an entry in the effect rack in the track info pane (which is shown at the left side of the arranger when the according track is selected). Right-clicking them offers a self-explanatory popup menu.

1.4 Configuration

Minimum control period Plugins can usually process an arbitrarily small (or large) amount of samples. If some plugin control value changes continously, to provide ideal listening experience, MusE would need to call the plugin 44100 times a second, asking for one single value at a time. With the minimum control period setting, the user can force MusE to ask the plugin for at least N values. Setting this value to 64 would in this situation make MusE call the plugin $689 = \frac{44100}{64})$ times a second, asking for 64 values at a time. While doing this will reduce accuracy of control changes, it may also reduce CPU usage, because calling the plugin more often, requesting smaller chunks, is more expensive than calling it seldomly, requesting larger chunks.

Recommendation If you have no performance problems, or if you want to do the final downmix of your project, set this to a low value. If you're experiencing performance problems, increasing this value might help.

Chapter 2

Internals – how it works

This chapter explains how MusE is built internally, and is meant to be an aid for developers wanting to quickly start up with MusE. For details on *why* stuff is done please refer to the following chapter.

2.1 User controls and automation

2.1.1 Handling user input

Plugins and synthesizers When the user launches a plugin's GUI, either a MusE-window with the relevant controls is shown, or the native GUI is launched. MusE will communicate with this native GUI through OSC (Open Sound Control).

The relevant classes are `PluginGui`, `PluginIBase` (in `plugin.h`) and `OscIF` (in `osc.h`).

If the user changes a slider, first the corresponding control is disabled, making MusE not steadily update it through automation while the user operates it. `PluginIBase::setParam` is called, which usually writes the control change into the ringbuffer `PluginI::_controlFifo`. (`PluginI::apply()`, `DssiSynthIF::getData()` will read this ringbuffer and do the processing accordingly). Furthermore, the change is written into a "to be recorded"-list (done by calling `AudioTrack::recordAutomation`). This list is processed after recording has finished.

Disabling the controller is both dependent from the current automation mode and from whether the GUI is native or not. In `AUTO_WRITE` mode, once a slider is touched (for MusE-GUIs) or once a OSC control change is received (for native GUIs), the control is disabled until the song is stopped. (not sure)

In `AUTO_TOUCH` (and currently (r1492) `AUTO_READ`, but that's to be fixed) mode, once a MusE-GUI's slider is pressed down, the corresponding control is disabled. Once the slider is released, the control is re-enabled again. Checkboxes remain in "disabled" mode, however they only affect the recorded automation until the last toggle of the checkbox. (Example: start the song, toggle the checkbox, toggle it again, wait 10 seconds, stop the song. This will NOT overwrite the last 10 seconds of automation data, but everything between the first and the last toggle.). For native GUIs, this is a bit tricky, because we don't have direct access to the GUI widgets. That is, we have no way to find out whether the user doesn't touch a control at all, or whether he has it held down, but just doesn't operate it. The current behaviour for native GUIs is to behave like in `AUTO_WRITE` mode.

Chapter 3

Design decisions

3.1 Automation

As of revision 1490, automation is handled in two ways: User-generated (live) automation data (generated by the user moving sliders while playing) is fed into `PluginI::_controlFifo`. Automation data is kept in `AudioTrack::_controller`, which is a `CtrlListList`, that is, a list of `CtrlList`s, that is, a list of lists of controller-objects which hold the control points of the automation graph. The `CtrlList` also stores whether the list is meant discrete (a new control point results in a value-jump) or continous (a new control point results in the value slowly sloping to the new value).

While `PluginI::_controlFifo` can be queried very quickly and thus is processed with a very high resolution (only limited by the minimum control period setting), the automation value are expensive to query, and are only processed once in an audio *driver* period. This might lead to noticeable jumps in value.

This could possibly be solved in two ways:

Maintaining a slave control list This approach would maintain a fully redundant slave control list, similar to `PluginI::_controlFifo`. This list must be updated every time any automation-related thing is changed, and shall contain every controller change as a tuple of controller number and value. This could be processed in the same loop as `PluginI::_controlFifo`, making it comfortable to implement; furthermore, it allows to cleanly offer automation-settings at other places in future (such as storing automation data in parts or similar).

Holding iterators We also could hold a list of iterators of the single `CtrlList`s. This would also cause low CPU usage, because usually, the iterators only need to be incremented once. However, it is pretty complex to implement, because the iterators may become totally wrong (because of a seek in the song), and we must iterate through a whole list of iterators.

Just use the current data access functions By just using the current functions for accessing automation data, we might get a quick-and-dirty solution, which however wastes way too much CPU ressources. This is because on *every single frame*, we need to do a binary search on multiple controller lists.

Chapter 4

Feature requests

4.1 Per-Part automation and more on automation

Automation shall be undo-able. Automation shall reside in parts which are exchangeable, clonable etc (like the MIDI- and Wave-Parts). Global per-synth/per-audiotrack automation shall also be available, but this can also be implemented as special case of part automation (one long part).

4.2 Pre-Rendering tracks

4.2.1 The feature

All tracks shall be able to be "pre-renderable". Pre-rendering shall be "layered". Pre-rendering shall act like a transparent audio cache: Audio data is (redundantly) stored, wasting memory in order to save CPU.

That is: Each track owns one or more wave-recordings of the length of the song. If the user calls "pre-render" on a track, then this track is played quasi-solo (see below), and the raw audio data is recorded and stored in the "layer 0" wave recording. If the user has any effects set up to be applied, then each effect is applied on a different layer (creating layer 1, layer 2 etc).

This means, that also MIDI and drum tracks can have effects (which usually only operate on audio, but we HAVE audio data because of this prerendering).

Furthermore, MusE by default does not send MIDI events to the synthesizers but instead just plays back the last layer of the prerecording (for MIDI tracks), or does not pipe the audio data through the whole plugin chain (causing cpu usage), but instead just plays back the last layer. The hearable result shall be the same.

Once the user changes any parameter (automation data or plugins for wave tracks, MIDI events or effect plugin stuff for MIDI tracks), then MusE shall generate the sound for this particular track in the "old" way (send MIDI data to synthes, or pipe audio data through plugins). (So that the user will not even notice that MusE actually pre-renderered stuff.) Either MusE automatically records this while playback (if possible) or prompts the user to accordingly set up his cabling and then record it. Or (temporarily) disables prerecording for this track, falling back to the plain old way of generating sound.

Quasi-solo means: For wave tracks, just solo the track. For MIDI tracks, mute all tracks which are not on the same synth (channel?), and mute all *note* events which are not on the quasi-soloed track. This causes MusE to still play any controller events from different tracks, because they might have effects on the quasi-soloed track. (You can have notes on channel 1 on one track and controller stuff on channel 1 on another track; then you would need quasi-solo to get proper results.)

4.2.2 Use cases

Saving CPU On slow systems, this is neccessary for songs with lots of, or demanding (or both) soft synthes / plugins. Even if the synth or plugin is so demanding that your system is not able to produce sound in real-time, then with this feature you'll be able to use the synth (this will make editing pretty laggish, because for a change you need to re-render at least a part before you can listen to it, but better than being unable to use the synth at all!)

Exporting as audio project Using pre-rendering on all tracks, you easily can export your project as multi-track audio file (for use with Ardour or similar DAWs). Just take the last layer of each track, and write the raw audio data into the file, and you're done. (Maybe we are even able to write down the raw-raw layer0 audio data plus information about used plugins and settings etc..?)

Mobile audio workstations You might want to work a bit on your audio projects on your notebook while you're not at home, not having access to your hardware synthesizers. Using this feature, you could have pre-recorded the stuff in your studio before, and now can at least fiddle around with the non-hw-synth-dependent parts of your song, while still having your *full* song with you.

Applying effects on MIDI tracks If you have many physical audio inputs, you might already be able to apply effect chains on MIDI tracks, by wiring the synthes' audio outputs to your soundcard's inputs, and applying the effects on dedicated input tracks you have to create. This requires you to have expensive hardware, and is pretty complicated, because you need one additional track per MIDI synth.

This feature allows you to apply effects on single MIDI tracks, and not only on full MIDI synthes, and doesn't require you to be have that many physical audio inputs (you need to manually replug your synthes, however).

4.2.3 Possible scenarios

Setting it up Create a wave track, MusE will allow you to set or unset prerendering for every plugin in the plugin rack (recording the actual track is useless because it would be a plain copy). Create a MIDI track, MusE will ask you on which physical audio input your synth is connected. Setting up multiple synthes on one physical audio in is allowed, see below.

Pre-rendering stuff When the user presses the "pre-render" button, all tracks which have been changed since their last pre-rendering will be re-rendered. If you have multiple hardware synthes set up as they were connected to one physical audio input port, MusE will prompt you to first plug the proper cable in.

Making changes Change a note in a MIDI part, move or delete a part or change automation parameters. MusE will temporarily disable the pre-rendered information and instead generate the sound via sending out MIDI events, piping stuff through effect chains or similar. If you play back the whole song, or if you manually trigger a re-rendering of a track via the context menu, MusE will play back the stuff, record it again and re-enable the pre-rendered information.

4.3 Extensions

Automatic discovery of physical audio connections The user plugs all (or only some) synthes' audio outs into the available audio inputs, then runs automatic discovery. This will send MIDI events to each synthesizer, and look on which audio in there's activity. Then it will assume that the synthesizer is connected to that particular audio in. Audio ins which show activity before any MIDI events were sent are not considered, as they're probably connected to microphones or other noise-generating non-synthes.

Audio export As described in the Use cases, MusE can allow you to export your song in some multitrack audio format.

Cheap/Faked changes For expensive or unavailable synthes, changing the Volume midi controller, the Pan controller or similar "easy" controllers will not trigger a complete re-rendering, but instead "fake" the change, by changing the volume data directly on the recorded wave. This might require some learning and might even get pretty complicated.

Intelligent re-recording For tiny changes, MusE shall only re-render the relevant part. If you change some MIDI notes, then begin re-recording shortly before the changes, and end re-recording as soon as the recorded stuff doesn't differ to much from the stuff coming from the synth. Then properly blend the old recording with the updated part.