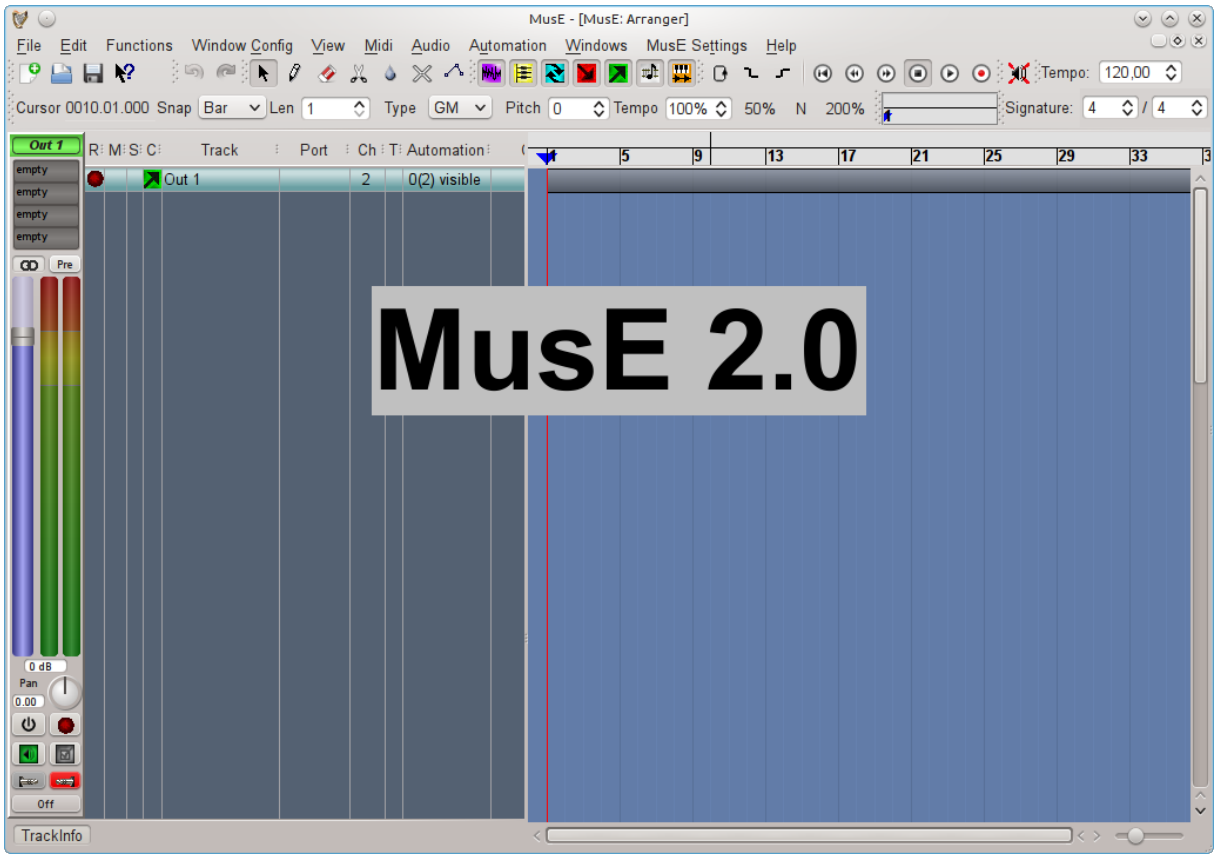

MusE - [MusE: Arranger]
File Edit Functions Window Config View Midi Audio Automation Windows MusE Settings Help
Tempo: 120,00
Cursor 0010.01.000 Snap Bar Len 1 Type GM Pitch 0 Tempo 100% 50% N 200%
Signature: 4 / 4
Out 1
R M S C Track Port Ch T Automation
Out 1 2 0(2) visible
empty
empty
empty
empty
Pre
0 dB
Pan
0.00
Off
TrackInfo
MusE 2.0

Contents

Chapter 1

What is this?

You are, if you have printed this document, holding in your hand the written documentation for the audio and midi sequencer MusE version 2.
http://www.muse-sequencer.org is MusE's home on the internet where everything MusE releated should be possible to find, software, this documentation, forums, mailing lists, bug reporting, FAQs. If you have this document but not the software head on over there to find what it's all about.

Chapter 2

User's manual

2.1 Introduction

2.1.1 A brief history of computer audio and MusE

To quickly summarize over a decades open source development: in 1999 Werner Schweer released the first version of MusE, muse-0.0.1.tar.gz, in it's first few releases (actually not few, Werner relentlessly churned out new releases) MusE was only a midi sequencer. The target was to create a fully fledged midi sequencer for the Linux operating system. Over the years audio was added among with other things implemented and sometimes abandoned. Today MusE is a stable and feature rich music creation environment which strives to encompass most of the music recording process, creation, editing, mastering.

2.1.2 Definitions

CTRL refers to the control key on the keyboard, e.g. **CTRL+C** means to press and hold the control key while pressing the c key. Make sure you know where you have it so you won't accidentally lose control (bad jokes are the best jokes, so say we all!).
SHIFT refers to the shift key on the keyboard, see above for usage
ALT refers to the alt key on the keyboard, see above for usage
$> is used as a generic definition for a terminal prompt. When the manual lists a command that shall be typed, the prompt is not part of the command.
Keys are always referred to in bold uppercase, e.g. **A**. For instance **SHIFT+A** for the key a pressed together with the shift key.
Sometimes terminal examples are written tabbed in with a fixed font to visualize more closely what something looks like on the screen. E.g.

```
$> muse2
```

2.1.3 Getting up and running for impatient people

Install MusE from the repository of your chosen distribution. To get decent performance start Jack with the following command in a terminal:

```
$> jackd -d alsa -d hw:0 -p 256
```

Or, if you prefer, use the launcher utility QJackCtl to get some help starting Jack. After this, start MusE from the menu or fire up another terminal and type

`muse2`.

If this didn't work out read on for the slightly more complete route for getting things started.

2.1.4 Getting up and running

Installation from binaries

There are several ways to install MusE depending on your situation. The most convenient way is to install a prepackaged version from your chosen distribution. The drawback of this is that it may not be the most recent version, though often there is a more recent package from a private packager.

Installation from source

Building MusE from source is not hard, there are a number of prerequistes that must be met but the actual building should be painless (ha, famous last words).
Please follow the README in the source package and/or read the instructions on the homepage: http://muse-sequencer.org/index.php/Installation

Hardware

MusE on the Linux platform supports midi through ALSA and Jack-midi and audio through Jack. For information on what hardware is supported there are some convenient places to check:

- Alsa soundcard matrix at http://www.alsa-project.org/main/index.php/Matrix:Main
- http://FFADO.org for firewire devices.

Also, as is often a very good approach for Linux and open source, the various forums available on the internet often contain good information. Chances are someone has already tried your configuration and/or had your specific problem and the solution is already written down.

Launching

After installation the binary muse2 is installed on the computer. If MusE was installed from a distribution repository the binary may have a different name depending on the distribution policies. Most distributions do however install a menu entry so MusE should be conveniently available from there.

Audio preconditions

In the standard case MusE expects to find and connect to the Jack audio server http://jackaudio.org. Make sure jack is installed (if MusE was installed with a distribution-package Jack will very likely already be installed) For Jack to run with best performance your system should be sufficiently tuned to allow it to run with realtime capabilities. The realtime configuration is configuration of the operating system and roughly consists of two parts.

1. By default on most distros only the superuser lets applications setup realtime capabilities. Please see the APPENDIX for setting up realtime
2. Maximizing performance. A standard linux installation may not able to reach the performance required by a power user. This requires exchanging the linux kernel for a so called lowlatency kernel, this is also covered by the realtime APPENDIX.

Running MusE

Find MusE in the menu or open a terminal and enter muse2.

```
$> muse2
```

A splash screen should pop up followed by the main application window and you are off!
If an error like the screenshot below pops up the Jack audio server is either not running or started as a different user than what you are trying to start MusE as.

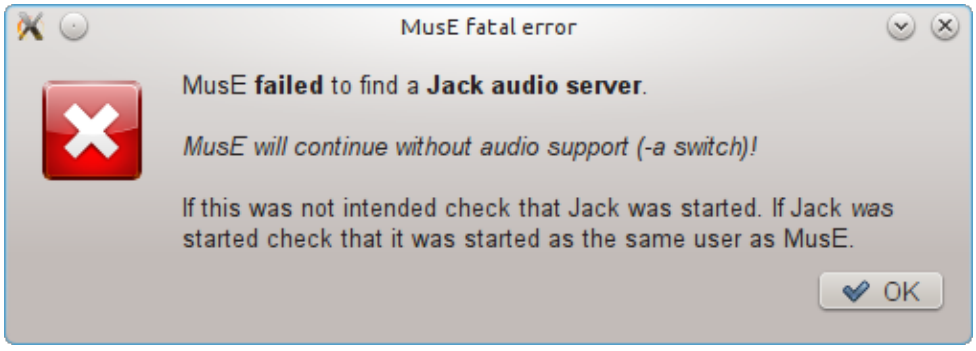


Figure 2.1: Jack server missing

Midi only

MusE can be started in Midi-only mode where MusE does not have any external dependencies apart from ALSA midi. In this case start MusE from a terminal: `$> muse2 -a`

ALSA midi with Jack

If Jack is running, by default MusE will not use ALSA devices, preferring Jack midi instead. To force ALSA devices to be used as well as Jack midi, start MusE with the -A option: `$> muse2 -A`

2.1.5 Beginners tutorial

To get a quick grip of what MusE can achieve please follow this beginners tutorial.

Midi Setup

First off, fire up MusE as was described in the previous chapter, making sure that the jack audio server is started with sufficient configuration to allow for audio output without breakup. Also make sure your system can make sound.

Soft synth test

With MusE up and running right click in the Track-pane (see Fig. 2.8) and select **Add Synth > MESS > vam soft synth**. A Soft Synth track called vam-0 should appear as well as a separate GUI for the synthesizer.

Now right click once more in the Track-pane and select **Add Midi Track**. Another track appears called Track 1, and its track list Port column should show it is bound to the synth that was just created vam-0. If it is not, click on the Track 1 Port column to open a drop-down list of available devices and choose vam-0.

Now select the drawing tool icon from the toolbar, alternatively press the shortcut key **D**. Move the mouse over to the arranger canvas as referenced in Fig. 2.8 and point at the midi track, the mouse should have changed to a small pencil. Draw a Part along the midi track using the mouse. For this exercise it is not important where or how large the drawn Part is. When you are done double click on the drawn part. This will open up the Piano Roll editor. To the left of the Piano Roll there are piano keys in a vertical line, try clicking on the keys in this virtual keyboard each click should be rewarded with a synth sound (maybe of questionable quality, a sound nevertheless)

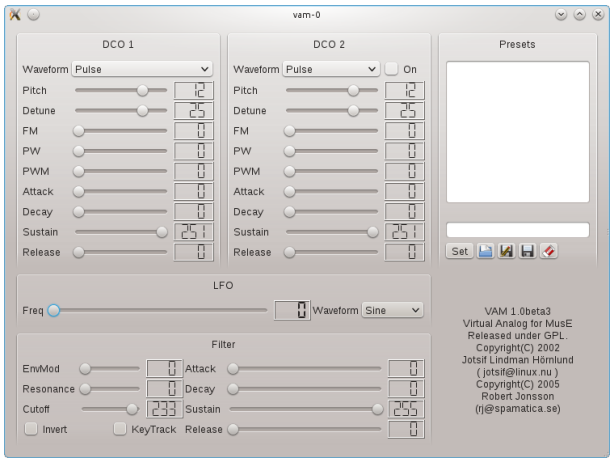


Figure 2.2: vam synthesizer

Missing sound

If you got sound from the previous exercise you can carry on to the next, or keep reading for further enlightenment in case you come upon trouble later on. If there is no sound we need to do some fault hunting. First off, click on Arranger window once more and select the vam-0 track in the track-pane. Now

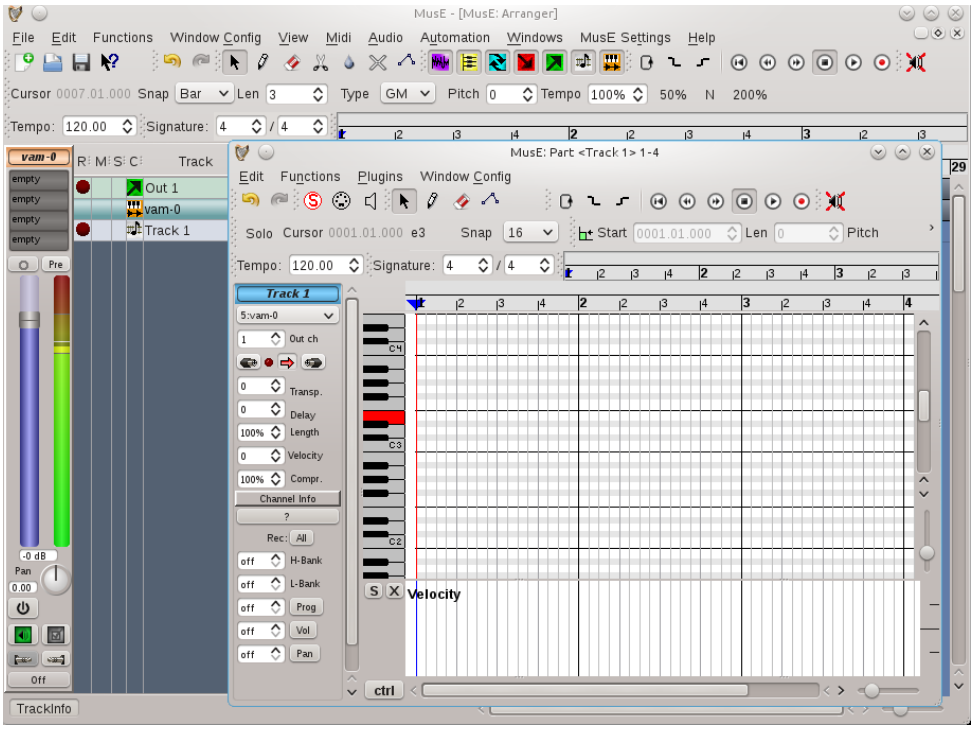
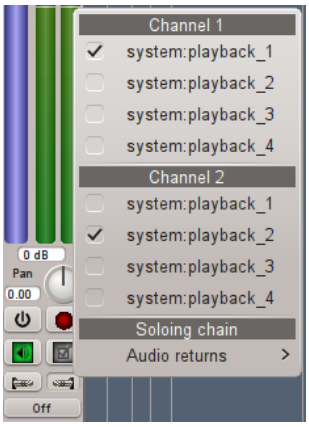


Figure 2.3: Midi editor view

bring back Piano Roll window and align the windows so you can see the piano keys as well as the Meter on the Mixer Strip (see the 5 Function by function chapter for more information on these windows). The result should be something like the following:

When pressing one of the keys on virtual Keyboard the Meter on the Mixer Strip should light up in green to visualize that the Synth is making sound, if it is not try to trace back your steps and and see if you did anything differently than described. Now, if the Meter lights up but there is still no sound we need to check the routing between the tracks. Click on the Arranger window again and select the Out 1 track, this is the predefined output which MusE by default loads at startup, at the bottom of Mixer Strip there are two buttons looking like tele- jacks, these bring up the inputs and outputs of the track, click on the right one, the output and make sure that it is connected to some valid outputs on your system. Click on the outputs to select them, if you did changes here go back and try clicking on the Piano Roll keyboard again, hopefully it helped. If there still are problems make sure your system actually can make sound through Jack, this is however getting outside the scope of this manual.



This might be the time to bring up the concept of community support. Open source software could never be what it is without the support given by individuals on forums and mailinglists, if the information given in this document is not enough, try googling your problem and/or get in touch with one of the online forums for MusE or Linux audio in general. See some pointers in the Support chapter.

Recording Midi

TBD

Recording Audio

At this point we'll make a a slight detour into full on audio recording. Getting audio out of MusE has already been covered in the previous chapters so we will concentrate on the additional steps needed to record onto an audio track.

When MusE is first fired up, the output track has already been created (more about this in the chapter about templates), to proceed with audio recording we need to add two additional tracks, a wave track and an input track.
When MusE is first started right click in an empty space on the track view and select **Add Audio**

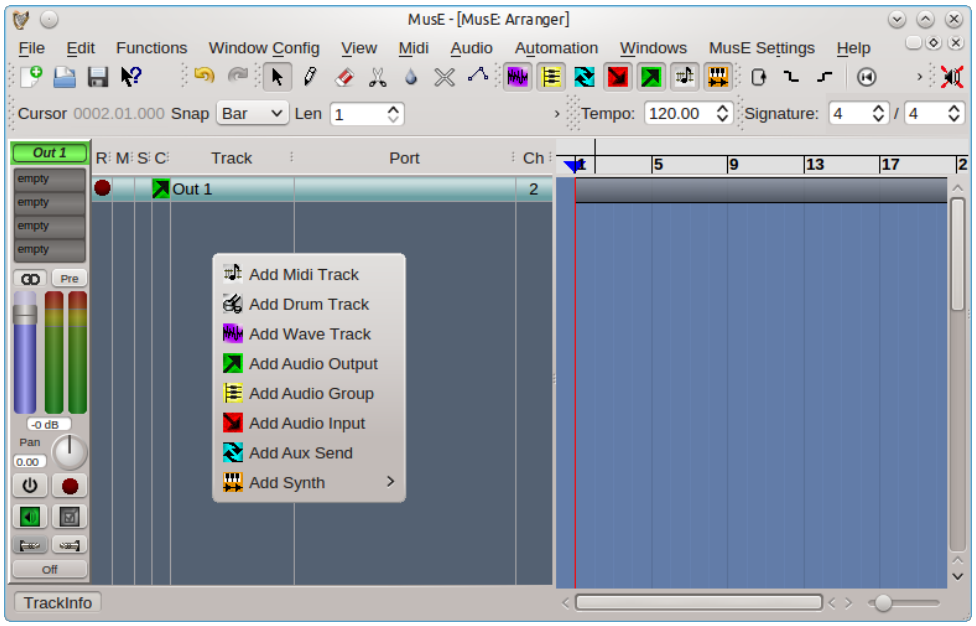


Figure 2.4: Add track

Input. Right click again and also select **Add Wave Track**. Two additional tracks are now visible in the Arranger, "Input 1" and "Track 1", bring up the mixer with **F10** and you should see the following configuration.

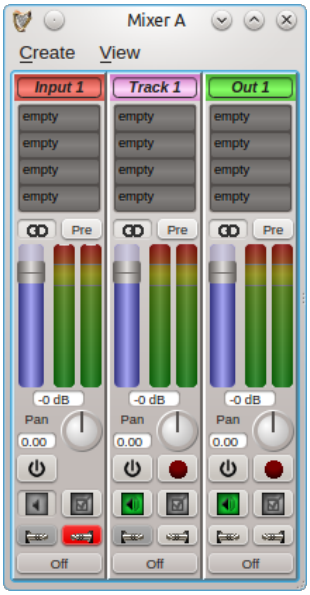


Figure 2.5: Mixer with one input

Note the buttons on each mixer strip. hover over them to see their functionality. For more information on all the buttons see coming chapters about the mixer. For now lets just do what we must.
1. click on the stereo symbol over the slider to change the input to a mono track.
2. do the same for the wave track (optional)
3. click on the Mute (gray speaker) icon on the input track to unmute it.
4. click on the input routing button (see the tooltip, it looks like a tele plug) on the input track and

select an appropriate connection from your system.
5. click on the output routing button on the input track and select **Track 1**

Already after the meter on the input track should be able to display that there is incoming sound from your sound source. If there actually is sound coming from your sound source, that is.
We are now nearly ready to start recording. First we need to select a location to store the files. MusE does not use a centralized storage of soundfiles but uses the path of the song-file (extension .med) as guidance as to where the audio files should be placed. Now as it happens MusE will prohibit us from starting a recording until the songfile has been stored. So lets take advantage of this behaviour and just go ahead and try to record. Let's get started.
In the mixer click on the red **record** dot on the Audio Track to arm it for recording (or enable if you will). Now when there is audio coming into the input it will also show up on the Audio Track. Also note that all the input and output routing buttons on the tracks now have the same gray color, this means that all of the tracks have a proper connection.

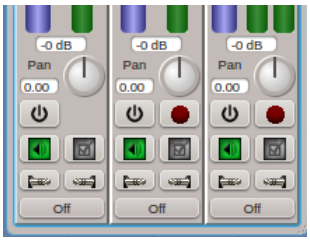


Figure 2.6: Mixer buttons

All fine and dandy. Now bring up the arranger window and find the round, red on white **record** button and click on it. This is your queue to MusE to prepare for recording. However since we have not saved our song we are presented with a dialog to do just that.
Note the check box for creating a project folder, when working with audio this is very much recom-

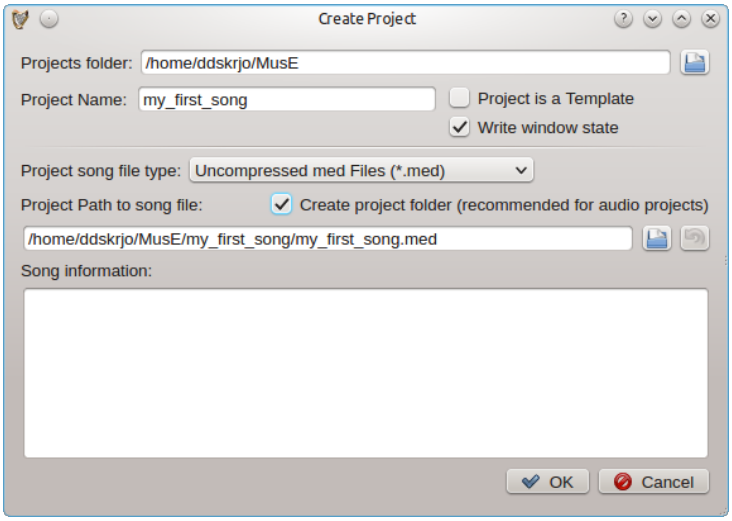


Figure 2.7: Save song

mended or you may soon loose track of what audio files belong to which song.
Finally we are ready to start recording! The process is completed by clicking on the **Play** button in the Arranger. If all went well MusE then starts to record a wave file from the Input Track placed in your song directory.
When you wish to stop recording press **Stop** in the Arranger, now the resulting waveform should be visible in the Arranger. After rewinding the Play position and pressing **Play** again the resulting sound should be audible through the connected output.

2.2 Basic overview

In this section we will make a step by step walk-through of all the different editors, their purpose and what functions they support.

2.2.1 Main/Arranger

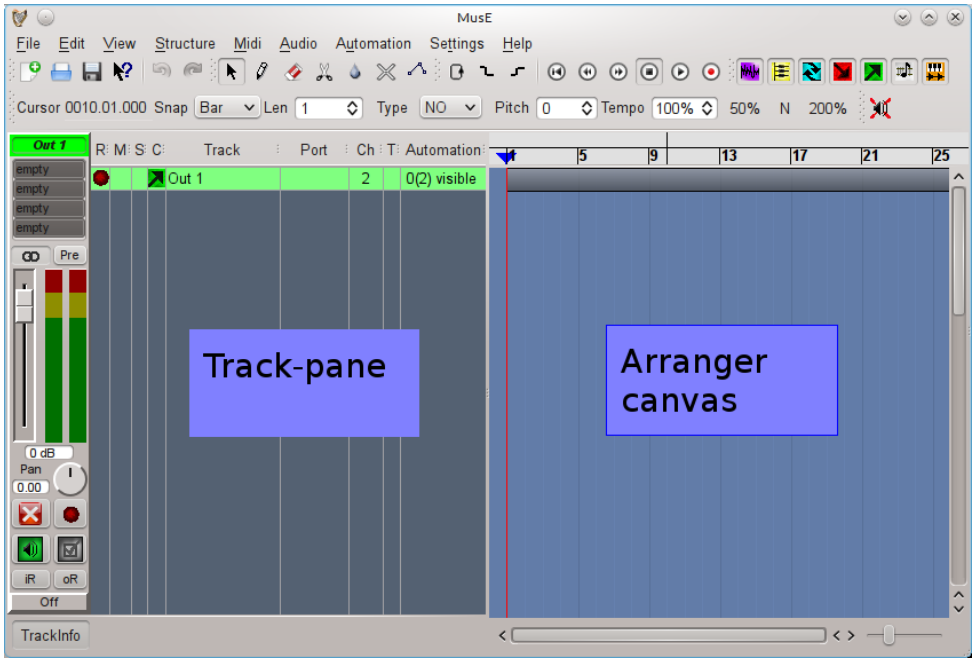


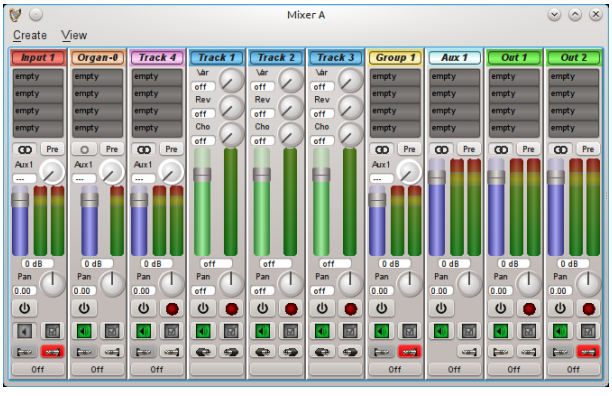
Figure 2.8: MusE main window

Above is the main window of MusE, the Arranger, this is what greets you when launching MusE. The Arranger consists of two main parts, the Track-pane and the Arranger canvas. The Track-pane lists all currently visible tracks and the Arranger canvas contains all Parts of the composition. The screenshot above shows an empty project. Below is MusE with a song in progress, turns out it wasn't a very good song, but for our purposes it is fine. In the below screenshot there are a lot of tracks visible in the Track-pane, each have an icon which indicate it's type, wave-track, input, output etcetera, more about that later. In the Arranger canvas a number of parts are visible, the ones in yellow are in this composition wave files, the multicolored line are different Parts of a drum track.

2.2.2 Mixer

Choosing **View > Mixer A** or **B** from the menu in the main window will bring up the mixer as viewed below. The mixer will open with all options enabled, showing channel strips for all tracks in the current setup, depending on how far you have gotten this view may become very large, at which point it may be a good idea to limit what is viewed in the Mixer. From the view menu all the different kinds of tracks can be toggled on/off from the mixer. Some may find it a good idea to use the two mixers A and B setup with different setup and store this in your song template(s), more about this in the Song Template section. It can be argued that everything in MusE is a track analogous to the Unix idiom that everything is a file. The types of tracks visible in the mixer (and track-pane) are:

- Audio output
- Audio input
- Group track
- Aux track



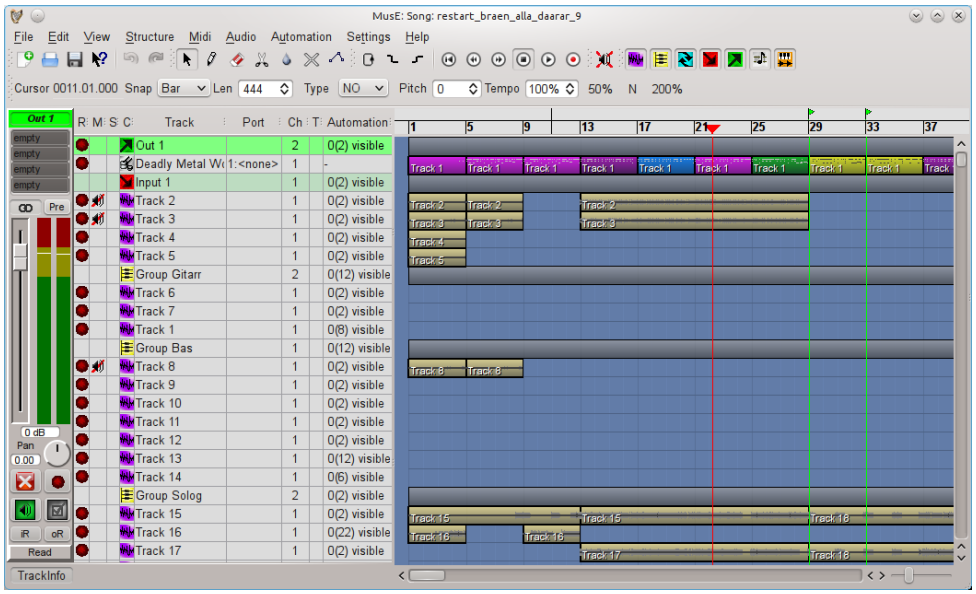


Figure 2.9: MusE main window with arrangement

- Wave track
- Synth track
- Midi track

There is also a Midi Track variation called Drum Track, they are however not distinguishable from Midi Tracks in the Mixer. Also the strips for midi tracks are different in the Mixer than in the Track-pane view.

2.3 Tracks and parts

MusE arranges your music in *tracks* and *parts*.
The following section shall provide 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.

2.3.1 Tracks

There are two general classes of tracks: MIDI tracks and audio tracks. MIDI tracks (and drum tracks which are internally MIDI tracks) can hold note data. The Wave track is a type of audio track which holds wave data. There are also several other kinds of audio tracks.

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 input tracks 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 input track with, and then route the input tracks to various other tracks such as wave tracks.

Audio output tracks These provide the path for your project's audio data to outside. Set up the physical audio outputs you want to connect your audio out track with, and then route various other tracks, such as wave tracks, to the output tracks.

Audio group tracks Group tracks are like busses, where you can route other tracks to them, then route the groups to other tracks. Since group tracks have all the features of other audio tracks, like volume and pan, they provide a convenient common routing point where you have control of the sound before it is passed to other tracks.

Audio aux tracks These provide a more convenient way to mix several audio tracks together. With each audio aux track added, other audio tracks will gain a common send knob for adjusting the level sent to the aux track. This can be more convenient than using several group tracks.

Synthesizer tracks This type of track is a software synthesizer which MIDI and drum tracks can be assigned to.

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) singles out a track for listening. It mutes some other tracks but may phantom solo others. For more info see the section on soloing: 2.5 and phantom soloing: 2.5.1

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 to edit the track name.

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 synths, 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. (See automation 2.7.1). Clicking this will provide you with a popup menu with lots of submenus. Clicking on a submenu will select or unselect it showing or hiding the automation parameter as a graph overlaid on top of the track.
The submenus let you select the color you want to associate with the automation parameter. There you can also assign midi controllers to the parameters, a dialog is shown where you can manually choose the midi controller, with a *learn* button to 'listen for' and automatically recognize any midi controller operated by you.

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 trackinfo sidebar The MIDI trackinfo sidebar lets you change program, volume, pan and more. This sidebar can also be viewed at the left of the pianoroll editor.

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

Because of these limitations, we introduced the new-style drum tracks. They're not fully compatible with the old drum tracks, so the old are still retained. Under "Global Settings", "GUI settings", you can set up whether you prefer the old or new.

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.

MIDI trackinfo controls:

Output port: This drop-down list selects the midi port to send midi output from this track.

Output channel: This box selects the midi channel to be used on the output port.

Input and output routing: Selects midi ports and channels to receive midi from, and soloing paths. (See Routes 2.4).

Midi through: This button selects whether midi input is passed through to the selected output port. Depending on your midi devices and settings, there are cases when this should be off such as using the same port and channel for input and output (otherwise a double-note *echo* will be heard), and cases when it must be on such as when using a synthesizer track as output device.

Input detect indicator: Blinks when midi activity is detected on the selected midi channels on the selected midi input ports.

Transpose: This transposes midi input notes up or down in pitch. This is very useful if your midi keyboard hasn't enough keys or the selected output device plays an octave too low or high, and you would like to shift the octave of the incoming notes to compensate.

Delay: Adjusts the delay of the notes.

Length: Adjusts the length of the notes.

Velocity: Adjusts the velocity of incoming notes. Use it to compensate for a too-loud or too-soft keyboard.

Compression: Adjusts the compression of incoming note velocities. Use it to make soft incoming notes louder, and loud notes not so loud.

Instrument: Selects the midi instrument patch to be used by the selected output port. This is equivalent of dialing the patch in the bank and program boxes, except it displays a more friendly patch *name* as defined by the selected output port's midi instrument. See instruments, or port configuration 2.8.1

H-Bank: Selects the high bank number of the current patch.

L-Bank: Selects the low bank number of the current patch.

Prog: Selects the program number of the current patch.

Volume: Adjusts the midi volume controller.

Pan: Adjusts the midi pan controller.

The buttons beside the Prog, Volume, and Pan boxes store the value, at the current transport position, for midi automation. (See automation 2.7.1).

Note that the 'Prog' button stores H-Bank and L-Bank along with 'Prog' value, so there are no H-Bank and L-Bank buttons.

The 'All' button simply stores all three Program (and banks), Volume, and Pan values at once.

Tip: If the Song Type is GM, GS, or XG, you may need to store desired values at transport position zero, otherwise your adjustments may be overriden by the instrument when the transport is moved back to position zero. If this behaviour is undesired, you can set the Song Type to 'NO' meaning no song type.

Audio trackinfo sidebar Unlike the midi trackinfo sidebar, the audio trackinfo side bar is nothing more than an embedded audio mixer strip, the exact same strip as found in the mixers. (See mixer 2.2.2).

Effects rack: On the top of the audio trackinfo sidebar, there is an effects rack which allows you to apply various plugins on the audio. For more information on this, refer to 2.6.1.

2.3.2 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.

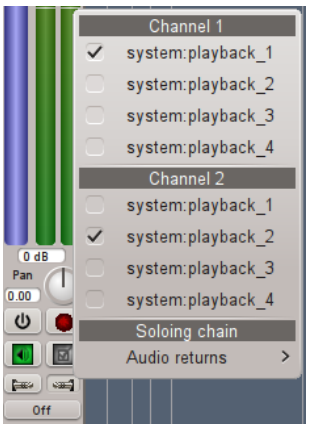
Parts are created by selecting the pencil tool and then drawing onto the right part area in the arranger. You can move them with the arrow tool, delete them using the **DEL** key, and a right-click opens a popup menu. This menu allows you even more stuff, such as setting the part's color, saving the 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.

2.4 Routes

Routes are how tracks are connected together and to the outside world. (They are also how Jack midi ports connect to the outside world. See midi port configuration 2.8.1). Each track strip has two buttons whose icons look like plugs. One button is for input routing and the other is for output routing. Clicking on these buttons will pop up a menu of available input or output routes that you can connect to. Most audio tracks list other tracks to connect to, but audio input and output tracks are special: Audio input track input routing menus list available Jack audio input ports. Conversely audio output track output routing menus list available Jack audio output ports.

Meanwhile MIDI and drum tracks allow you to route available MIDI ports and channels to the track using a handy popup matrix.



2.4.1 Anti circular routing

Any routing menu item which would cause a circular routing condition is grayed out. Find out why the condition would exist by examining routing paths involved and correct the situation if required.

Also, you cannot use a track's aux sends if the track has an input route path from ANY Aux Track. (See aux tracks 2.3.1). Aux send knobs and labels are disabled in that case.

2.4.2 Soloing chain routes

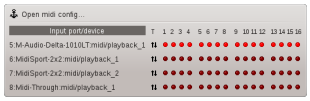
Soloing chains (see solo chains 2.5.2) are really just routes like any other. The available solo chaining paths are displayed in the routing popup menus.

2.5 Track soloing

Soloing allows you to single out a track for listening while muting others, without you having to mute the other tracks. (See soloing track attribute 2.3.1).

2.5.1 Phantom soloing

In order to solo a track and mute others so that it is heard, MusE employs 'phantom' soloing: When a track is soloed, MusE automatically solos all tracks routed to and from this track. (See routes 2.4). A phantom soloed track is indicated by a black square in the track pane solo column. (See track attributes 2.3.1).



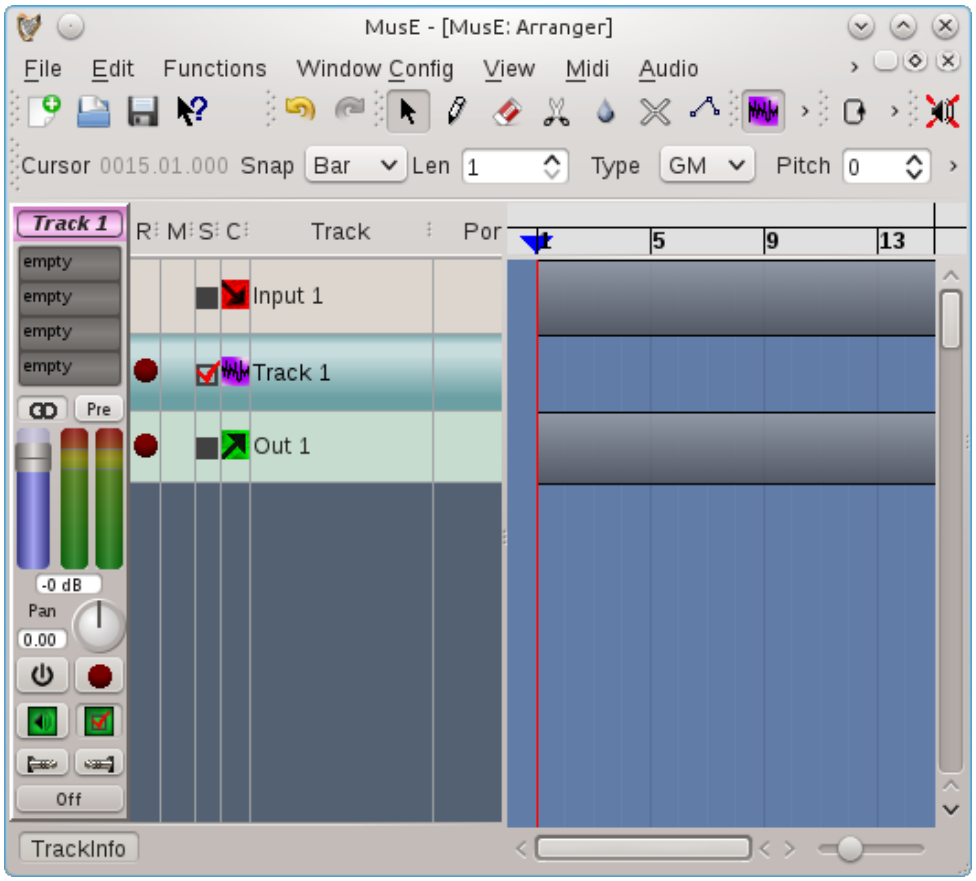


Figure 2.10: Soloing, with phantom soloing

2.5.2 Soloing chains

When an audio output track sends audio to some external entity, such as an external running application, and audio from the external entity is fed back into a MusE audio input track, solo chains allow you to solo the input or output, and MusE will complete the path automatically soloing the other, and all paths that came before or after it.

Solo chains also work with MIDI tracks chained to audio inputs: When a MIDI track drives some MIDI device whose audio is fed into MusE, solo chains allow the entire chain to be soloed.

Solo chains are accessed via routing menus. (See solo chain routes 2.4.2).

2.6 Plugins

Plugins are small add-ons which can process a track's data.

MIDI plugins operate on midi and drum tracks, and are found in the **Midi** menu.

Audio plugins can be applied to any track handling audio (that is, inputs, outputs, wave tracks, synth tracks). The effects rack section describes this. (See effects rack 2.6.1).

2.6.1 The audio effects rack

All audio track types (Input, Output, Group, Wave, Synth, and Aux) have an effects rack into which audio plugins can be inserted in a chain. Currently each rack can accomodate up to four plugins.

MusE currently supports LADSPA plugins and DSSI synth and effects plugins.

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 the rack items offers a self-explanatory popup menu.

All plugin controls can be automated. (See audio automation 2.7.1).

One must carefully consider how many audio inputs and outputs a plugin has, and how may channels the particular audio track has (1 mono or 2 stereo), and how MusE uses the plugins in the rack.

Learn more about this in the appendix Understanding the Effects Rack: 3

Audio plugin Graphical User Interfaces (GUIs)

Once a plugin is added, you need a way to manipulate its controls, which affect its behaviour and operate on the sound.

MusE can show a generic GUI which contains all of the plugin's controls arranged in a rather plain generic fashion.

Some plugins may also have a native GUI which looks much better (it was specifically designed for the plugin).

Both GUI types are opened from the effects rack right-click popup menu.

2.7 Automation

Automation is the ability to record (or construct) and playback exact sequences of control movements.

MIDI and audio automation are each currently uniquely different, but share some similarities.

2.7.1 Audio automation

Almost all graphical audio controls in MusE can be automated.

This includes an audio track's volume and pan, and the controls of any plugins in the effects rack, and if the track is a synthesizer track, all of the synth's controls.

Each control has a manual adjustment value. This value is shown when there is no automation data at all, or automation has been disabled.

For plugin and synth controls, it is usually more desirable to manipulate automation with the generic plugin GUIs, because MusE has full control over their behaviour. (See plugin GUIs 2.6.1).

There are a few ways to enter audio automation data:

- By adjusting audio controls while the transport is rolling. MusE will record the exact movements.
- By adjusting audio controls while the transport is stopped, at different transport positions. TOUCH mode allows this.
- By right-clicking any audio control and choosing an operation from the automation popup menu. This includes storing, erasing, and clearing automation events, and seeking the next or previous event.
- By drawing the data on the audio track's automation graphs. (See track automation 2.3.1).

Audio automation modes Each audio track strip has an automation mode button at the bottom. There are four automation modes:

OFF: Disables all automation, uses manual value always.

READ: Automation data is applied to controls. If any automation data exists, the manual value is overridden and has no effect.

TOUCH: Allows you to alter a control at any time, while transport is stopped or rolling, If rolling, when the control is released it returns to reading from automation data.

WRITE: Allows to adjust an initial value before rolling the transport. While rolling, when the control is released it does not return to reading from automation data.

Here is a screenshot of automation WRITE mode, and some automation data, with the track pane automation popup menu showing (see track automation 2.3.1):

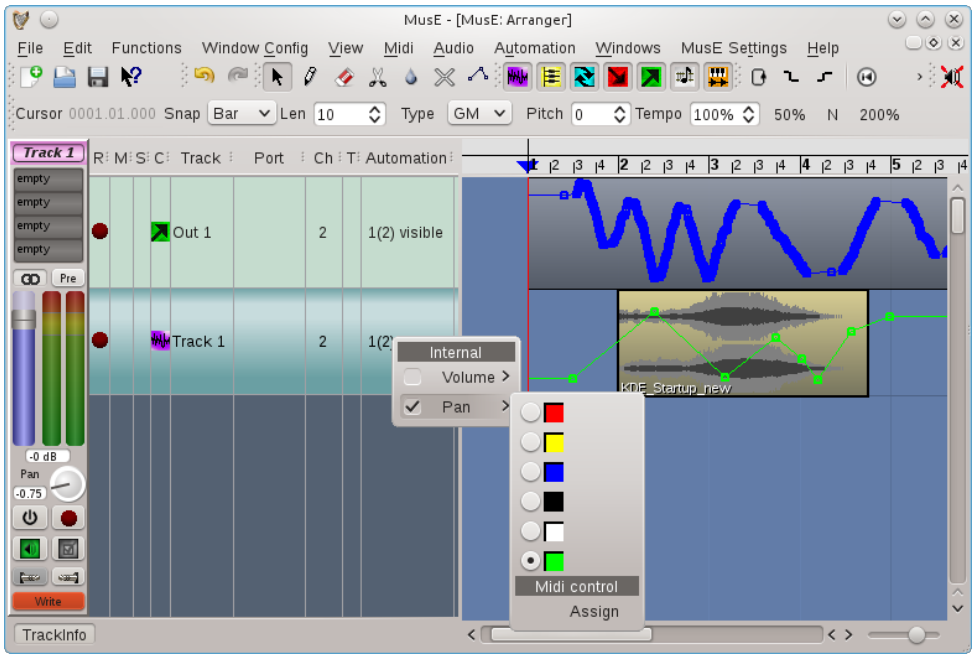


Figure 2.11: Audio automation graphs

2.7.2 Midi automation

MIDI automation is a slightly different concept: Unlike audio automation, currently there is no automation 'mode' and it doesn't record graphical control movements. Data is viewed from within the pianoroll and drum editors, by clicking on the 'Ctrl' button on those canvases.

Similar to audio controls, each midi control has a manual adjustment value. This value is overridden when there is midi automation data.

There are a few ways to enter MIDI automation data:

- By adjusting external MIDI controls (such as a midi keyboard pitch or modulation wheel) while the transport is rolling and both the transport and midi track are in record mode. MusE will record the exact movements. As mentioned earlier, note that graphical control movements are not recorded.
- By right-clicking any midi control and choosing an operation from the automation popup menu. This includes storing and erasing automation events.
- By adjusting volume, pan, bank or program boxes in the midi trackinfo panel and clicking the corresponding volume, pan, or program buttons. (See midi trackinfo 2.3.1).
- By drawing the data on a midi part's automation graphs.

Here is a screen shot of a midi track, containing a midi part which has been opened with the pianoroll editor and automation data showing.

The 'Ctrl' popup menu (bottom left) shows available midi controllers and the green dot indicates there is some data.

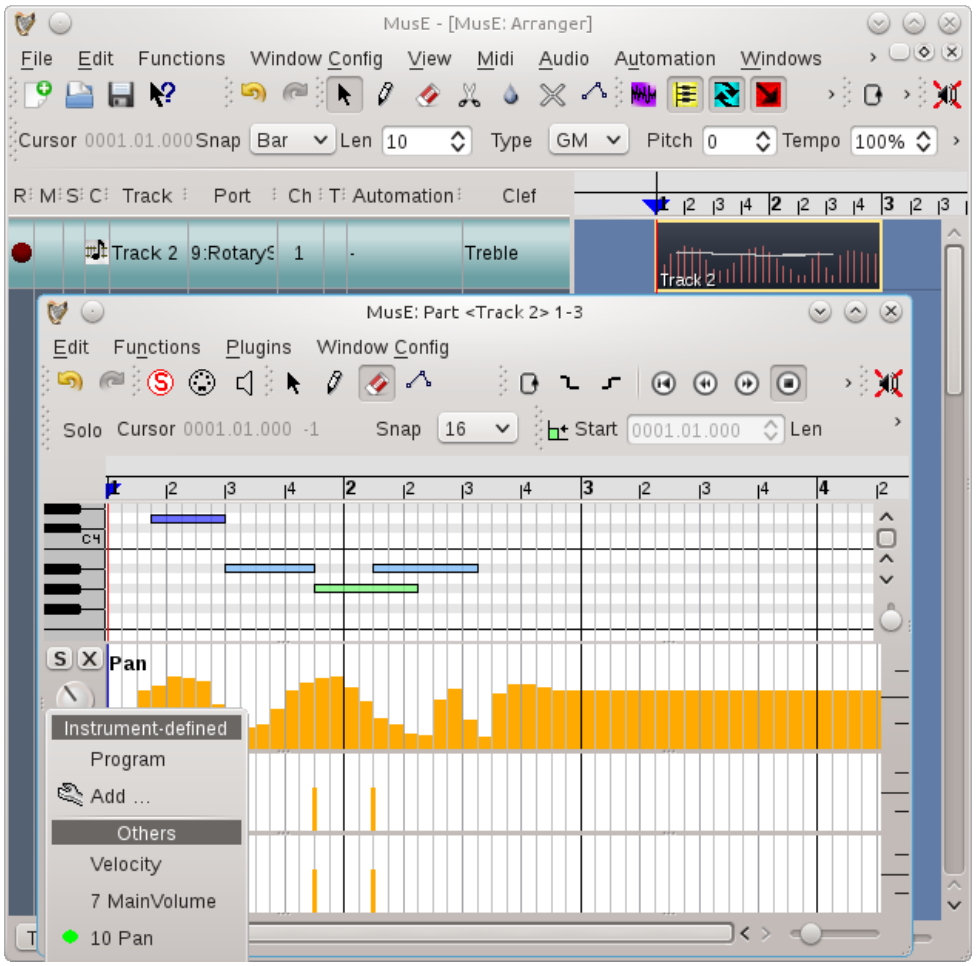


Figure 2.12: MIDI automation graphs

2.8 Configuration

2.8.1 MIDI ports

MIDI ports provide an abstraction layer for your MIDI hardware and synthesizers (which can be both software and hardware synthesizers), and other MIDI applications. 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.

The advantage of this abstraction layer is that if your system changes, for example you change MIDI hardware, then you need only modify the ports instead of all the tracks using those ports. This is similar to the audio input and output track abstraction to the outside world.

MIDI port configuration In the midi/softsynth configuration menu, you must map the port numbers to the actual 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 synth's GUI.

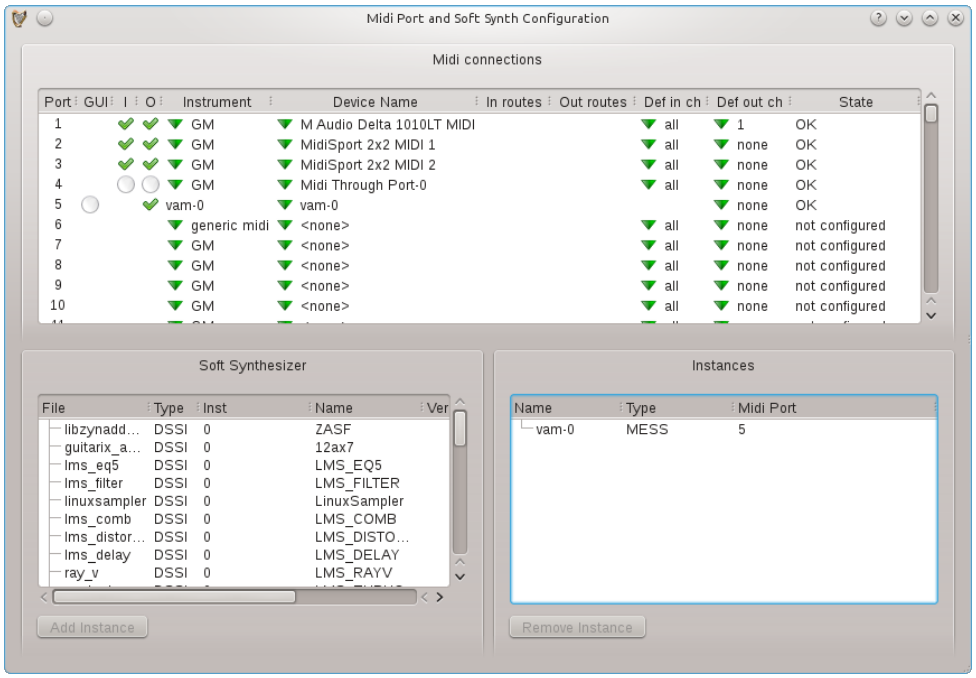


Figure 2.13: Midi configuration window

Columns in the MIDI configuration ports list:

GUI: For synthesizer devices, indicates if a gui is available and if it is showing. Click to show.

I: If present, the port can accept MIDI input. Click to enable or disable it.

O: If present, the port can send MIDI output. Click to enable or disable it.

Instrument: Selects the instrument to be used when MIDI is played through the port.

Device name: Selects or creates a MIDI device assigned to the port. These can be Jack MIDI devices or ALSA MIDI devices (if ALSA is enabled), or soft synthesizers. Jack MIDI devices are created by selecting Create Jack Device from the Device name drop-down menu. Jack MIDI devices can be renamed as you wish by clicking the device name. Soft synthesizers are created by clicking in the soft synthesizer list and then Add Instance. Or you can simply create a new synthesizer track from the arranger track list, or even the mixer menus.

In and Out routes: These are for Jack MIDI devices, they are the routes to and from available Jack MIDI ports. Jack may provide different alias names for these ports, you can select which alias is shown.

Default in channels: Auto-connect these port channels to new midi or drum tracks.

Default out channel: Auto-connect new midi or drum tracks to this channel on the port.

State: Indicates the state of the port including any errors opening it.

2.8.2 Global settings

Audio settings

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 3

Appendix

3.1 Understanding the effects rack

One must carefully consider how many audio inputs and outputs a plugin has, and how may channels the particular audio track has (1 mono or 2 stereo), and how MusE uses the plugins in the rack.

MusE will try to internally create as many independent copies (instances) of a plugin as necessary, to satisfy the number of channels in the audio track. Basically it divides the number of track channels by the number of plugin audio inputs or outputs to determine how many copies to make. First it examines the number of plugin audio outputs, and if there are none, it will examine the number of audio inputs, and if there are none, it will simply use just one plugin copy.

For mono tracks with plugins having more than one audio input or output, MusE uses the first input or output and ignores the rest.

For stereo tracks:

plugin inputs	outputs	copies	track in route channels	track out route channels
0	0	1	0	0
0	1	2	0	2
0	>=2	1	0	2
1	0	2	2	0
1	1	2	2	2
1	>=2	1	1 (L only)	2
>=2	0	1	2	0
>=2	1	2	2	2
>=2	>=2	1	2	2

Notice that on a stereo track with a plugin having one audio input and two audio outputs, only the first track input route channel is used (left only).

These same rules apply to inter-plugin audio when more than one plugin is in the rack chain. Extra audio outputs of one plugin may be ignored by the next plugin if not used.

Currently specialized plugins with many inputs and/or outputs are not really useful in MusE.

Nor are so-called 'realtime' control plugins which use audio inputs and outputs for control signals.

Loud noise alert! Beware of using such plugins in an audio effects rack.

Example: Consider a stereo Audio Input track with these effect rack LADSPA plugins:

- comb_splitter Comb Splitter by Steve Harris
- tap_stereo_echo Tap Stereo Echo by Tom Szilagyi

The Comb Splitter has one audio input and two audio outputs. The Stereo Echo has two audio inputs and two audio outputs.

The stereo Audio Input track will therefore ignore its second input route connection. It will process the left input only, separating it into stereo with the Comb Splitter, passing the split stereo signal into the Stereo Echo, finally producing stereo output available at the Audio Input track's output routes.

One improvement would be not creating unused redundant plugin copies between plugins in stereo tracks. For example, for a plugin having one audio input and one audio output, feeding a plugin having one audio input and two audio outputs, the extra copy of the first plugin is redundant and not required, but currently it is created anyway.

Chapter 4

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.

4.1 User interface programming

We use the QT Toolkit for GUI- and other programming. The *QT-Assistant* is an important tool for getting help. Almost everything can be looked up there.

GUIs can be either be hardcoded (see `arranger.cpp` for an example) or can be created using *QT-Designer* (see the dialogs under `widgets/function_dialogs/` for mostly cleanly-written examples). Don't forget to add your `cpp`, `h` and `ui` files to the corresponding sections in the `CMakeLists.txt`!

Additionally, MusE offers some custom widgets, like menu title items etc. Following, there will be a small, unordered list about custom widgets:

- `MusEGui::MenuTitleItem`: Provides a title-bar in a `QMenu`.
  Usage: `someMenu->addAction(new MusEGui::MenuTitleItem(tr("fnord"), someMenu));`
  Defined in `widgets/menutitleitem.h`.
- `MusEGui::PopupMenu`: Provides a `QMenu`-like menu which can stay open after the user checks a checkable action.
  Usage: just create a `new PopupMenu( true|false )` instead of a `new QMenu()`. (`true` means 'stay open')
  Defined in `widgets/popupmenu.h`.

4.2 Configuration

Configuration is a bit pesky in MusE in its current state. If you get confused by reading this chapter, that's a sign of a sane mind.

There are three kinds of configuration items:

- (1) Global configuration, like coloring schemes, plugin categories, MDI-ness settings
- (2) Per-Song configuration, like whether to show or hide certain track types in the arranger
- (3) Something in between, like MIDI port settings etc. They obviously actually are global configuration issues (or ought to be), but also obviously must be stored in the song file for portability. (This problem could possibly be solved by the feature proposal in 6.7.

Reading configuration `void MusECore::readConfiguration(Xml&, bool, bool)` in `conf.cpp` is the central point of reading configuration. It is called when MusE is started first (by `bool MusECore::readConfiguration()`), and also when a song is loaded.
It can be instructed whether to read MIDI ports (3), global configuration and MIDI ports (1+3). Per-Song configuration is always read (2).

When adding new configuration items and thus altering `readConfiguration()`, you must take care to place your item into the correct section. The code is divided into the following sections:

- Global and/or per-song configuration (3)
- Global configuration (1)
- Code for skipping obsolete entries

The sections are divided by comments (they contain `--`, so just search for them). Please do not just remove code for reading obsolete entries, but always add an appropriate entry to the 'skipping' section in order to prevent error messages when reading old configs.

Writing configuration Global configuration is written using the `MusEGui::MusE::writeGlobalConfiguration()` functions, while per-song-config is written by `MusEGui::MusE::writeConfiguration()` (notice the missing `Global`; both implemented in `conf.cpp`).

`writeConfiguration` is actually just a subset of the code in `writeGlobalConfiguration`. **Duplicate code!**

Song state Additionally to per-song configuration, there is the song's state. This contains "the song", that is all tracks, parts and note events, together with information about the currently opened windows, their position, size, settings and so on. Adding new items here is actually pretty painless: Configuration is read and written using `MusECore::Song::read` and `::write`, both implemented in `songfile.cpp`. There are no caveats.

How to add new items When adding global configuration items, then add them into the second block ("global configuration") in `readConfiguration` and into `writeGlobalConfiguration`.

When adding just-per-song items, better don't bother to touch the "configuration" code and just add it to the song's state (there might be rare exceptions).

When adding global configuration items, make sure you add them into the correct section of `readConfiguration`, and into `writeGlobalConfiguration`.

4.3 User controls and automation

4.3.1 Handling user input

Plugins and synthesizers

Overview 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 GUI element, first the corresponding control is disabled, making MusE not steadily update it through automation while the user operates it. Then MusE will update the plugin's parameter value, and also record the new value. When appropriate, the controller is enabled again.

Processing the input, recording Upon operating a slider, `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, `AudioTrack::recordAutomation` is called, which either directly modifies the controller lists or writes the change into a "to be recorded"-list (`AudioTrack::_recEvents`) (depending on whether the song is stopped or played).

The `AudioTrack::_recEvents` list consists of `CtrlRecVal` items (see `ctrl.h`), which hold the following data:

- the frame where the change occurred
- the value
- the type, which can be `ARVT_START`, `ARVT_VAL` or `ARVT_STOP`. `ARVT_VAL` are written by every `AudioTrack::recordAutomation` call, `ARVT_START` and `ARVT_STOP` are generated by `AudioTrack::startAutoRecord` and `stopAutoRecord`, respectively.
- and the id of the controller which is affected

It is processed when the song is stopped. The call path for this is: `Song::stopRolling` calls `Song::processAutomationEvents` calls `AudioTrack::processAutomationEvents`. This function removes the old events from the track's controller list and replaces them with the new events from `_recEvents`. In `AUTO_WRITE` mode, just all controller events within the recorded range are erased; in `AUTO_TOUCH` mode, the `ARVT_START` and `ARVT_STOP` types of the `CtrlRecVal` events are used to determine the range(s) which should be wiped.

How it's stored 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 continuous (a new control point results in the value slowly sloping to the new value). Furthermore, it stores a `_curVal` (accessed by `curVal()`), which holds the currently active value, which can be different from the actually stored value because of user interaction. This value is also used when there is no stored automation data.

`AudioTrack::addController` and `removeController` are used to add/remove whole controller types; the most important functions which access `_controller` are:

- `processAutomationEvents`, `recordAutomation`, `startAutoRecord`, `stopAutoRecord`: see above.
- `seekPrevACEvent`, `seekNextACEvent`, `eraseACEvent`, `eraseRangeACEvents`, `addACEvent`, `changeACEvent`, which do the obvious
- `pluginCtrlVal`, `setPluginCtrlVal`: the first returns the current value according to the `_controller` list, the second only sets the `curVal`, but does not insert any events.

Whenever a `CtrlList` has been manipulated, `MusEGlobal::song->controllerChange(Track*)` shall be called, which emits the `MusEGlobal::song->controllerChanged(Track*)` signal in order to inform any parts of MusE about the change (currently, only the arranger's part canvas utilizes this).

Enabling and disabling controllers 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 or seeked.

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.

The responsible functions are: `PluginI::oscControl` and `DssiSynthIF::oscControl` for handling native GUIs, `PluginI::ctrlPressed` and `ctrlReleased` for MusE default GUIs and `PluginI::guiParamPressed`, `guiParamReleased`, `guiSliderPressed` and `guiSliderReleased` for MusE GUIs read from a UI file; `guiSlider*` obviously handle sliders, while `guiParam*` handle everything else which is not a slider. They call `PluginI::enableController` to enable/disable it.

Furthermore, on every song stop or seek, `PluginI::enableAllControllers` is called, which re-enables all controllers again. The call paths for this are:

- For stop: `Song::stopRolling` calls `Song::processAutomationEvents` calls `Song::clearRecAutomation` calls `Track::clearRecAutomation` calls `PluginI::enableAllControllers`
- For seek: `Audio::seek` sends a message (`"G"`) to `Song::seqSignal` which calls `Song::clearRecAutomation` which calls `PluginI::enableAllControllers`

Chapter 5

Design decisions

5.1 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 `CtrlLists`, 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 `CtrlLists`. 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 6

Feature requests

6.1 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).

6.2 Pre-Rendering tracks

6.2.1 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.)

6.2.2 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).

6.2.3 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.

6.2.4 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 multi-track 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.

6.3 Slotted editors

Currently, MusE has the pianoroll editor, drum editor, score editor, then the controller editor which is inside the pianoroll/drum editor. All these editors have a very similar concept: the "time axis" is vertical and (almost) linear, they handle parts, and events are manipulated similarly.

A unified editor shall be created which allows you to combine different kinds of editors in one window, properly aligned against each other. These "different kinds of editors" shall be handled as "slots"; one unified editor window consists of:

- A menu bar, containing stuff suitable for the complete window, which might include window name, MDI-ness etc.
- A toolbar which contains controls suitable for every single slot.
- A container with one or more slots; the slots can be scrolled in y-direction if there are multipe slots.
- A time-scrollbar with zoom

Each slot contains the following:

- A menu button, button box or control panel for setting up this particular slot. This could contain "note head colors", "show a transposing instrument" etc for score edit slots, "event rectangle color", "grid size" and "snap to grid" for pianoroll/ drum editors.
- The actual canvas
- A y-direction scroll bar, possibly with zoom control (for pianoroll editor)

The main window does not show its scroll bar if there is only one slot, because the slot's scrollbar is sufficient then.

Slots can be added, destroyed, moved around, maybe even merged (if the slot types allow it); basically, you can compare them with the staves in the score editor.

The slots shall align against each other, that is, if a score editor slot displays a key change with lots of accidentials, then all other slots shall either also display the key change (if they're score slots) or display a gap. Events which happen at the same time shall be at the same x-coordinate, regardless which slot they are.

6.4 Controller master values

All controllers (MIDI-controllers and also automation controllers) shall have one set of "master values" which allow you to set a gain and a bias. Instead of the actual set value, value $*$ bias + $textrmbias$ shall be sent to the MIDI device / the plugin. For controllers like "pan", the unbiased values shall be transformed, that is, a pan of 64, with bias = 2 and gain = 0.5, shall be transformed to 66 (because 64 is actually 0, while 0 is actually -64). These values shall be set in the arranger and whereever the actual controller/automation values can be edited.

6.5 Enabled-indicator while recording

The MusE-plugin-GUIs shall display a small LED displaying whether a controller is currently enabled or disabled. By clicking this LED, the enabled state shall be switched.

Furthermore, there shall be a dedicated window which only lets you switch enabled/disabled states. This will be useful when using external GUIs or the MIDI-controller-to-automation feature, to re-enable a controller when in `AUTO_TOUCH` mode.

6.6 Linear automation editing

While holding some modifier key (like shift), operating the MusE-native- GUI sliders shall only generate control points when clicking and when releasing the slider. This will result in linear graphs for continous controllers, and in large steps for discrete controllers (which is in particular useful for stuff like "which low/high-pass filter type to use").

Maybe make this behaviour default for discrete controllers?

6.7 Symbolic names for MIDI ports

MIDI ports shall have a user-defined symbolic name (like "Korg" or "Yamaha DX 7"). The mapping between these symbolic names and the hardware port (like "ALSA midi out port") is stored in the global configuration.

Song files only specify the symbolic names as the ports associated with their tracks. No information about physical devices/port names, but only symbolic names are stored in the song file.

This resolves the issues mentioned in 4.2, and also allows the user to share his pieces with other people: They would only have to set up that symbolic-to-hardware mapping once (collisions are unlikely, because an equal symbolic name should usually mean the same device) and are happy, instead of having to re-map *every* port for *every* song.