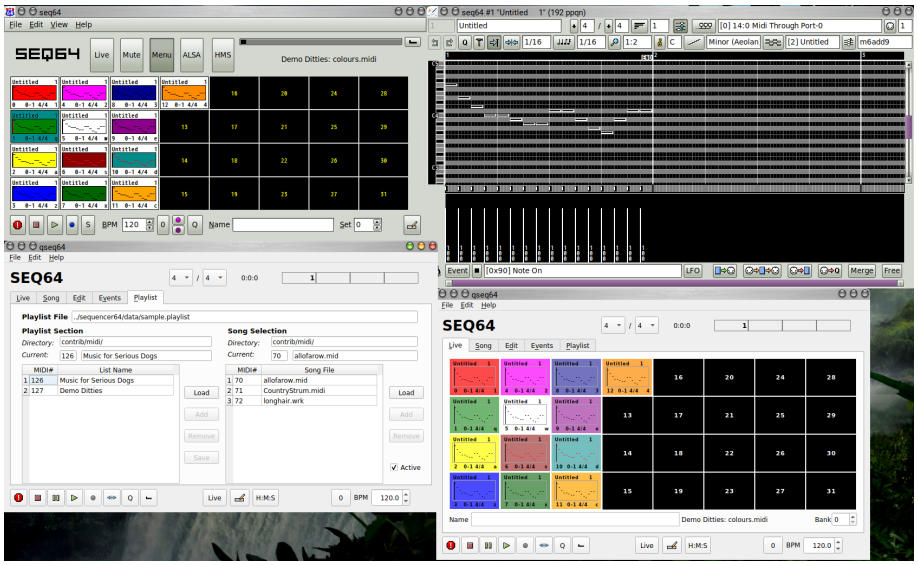
# Sequencer64 User Manual 0.96.1

Chris Ahlstrom
(ahlstromcj@gmail.com)

November 8, 2018




"Old and New"

# Contents

## List of Figures

## List of Tables

# 1 Introduction

This document describes *Sequencer64* [29], through version 0.96.1. The following projects support *Sequencer64*:

- https://github.com/ahlstromcj/sequencer64.git.
- https://github.com/ahlstromcj/sequencer64-doc.git.
- https://github.com/ahlstromcj/sequencer64-packages.git.

If impatient to get started mastering *Sequencer64*, proceed to section 1.5 "Let's Go!" on page 12. Otherwise, read about the features of *Sequencer64*.

*Sequencer64* adds native JACK MIDI support, fixes bugs in JACK transport, MIDI clocking, JACK Master mode, a daemon mode, play-list support with MIDI control, and more. *Sequencer64* can also be build to run as a "headless" application, from the command-line or as a Linux daemon, controlled via MIDI. More editing and navigation can be done using keystrokes, for faster editing. There are zoom keys for the editors, current-sequence highlighting, important bug fixes, and some optimization. With version 0.95.0, *Sequencer64* supports *Microsoft Windows*, via a modified version of PortMidi and by adopting and extending the *Qt 5* user-interface of the *Kepler34* project ([9]). See section 13 "Sequencer64 Qt 5 and Windows" on page 144, right away, if *Qt 5* and *Windows* support excites you. Please note that support for *Mac OSX* should be easy now, but we need help on that support, since we do not have a *Mac*.

We have many contributors to acknowledge. Please see section 20 "Kudos" on page 190.

## 1.1 Sequencer64: What?

*Sequencer64* is an reboot of *Seq24*, a live-looping sequencer with an interface more like a hardware sequencer than track-based MIDI sequencers. *Seq24* was a very active project, with a number of contributors, who created patches, additional functionality, and ports to Windows. We searched for these updates, and incorporated them into *Sequencer64* where feasible.

*Sequencer64* is not a synthesizer. It requires a hardware synthesizer, or a software synthesizer such as Timidity [33], FluidSynth [6], or the Windows built-in GS wave-table synthesizer.

*Sequencer64*, like *Seq24*, works like an *Alesis SR16* drum machine, which, for some, is a very intuitive and fast way to do MIDI. If one has worked with trackers like *SoundTracker* and *ShakeTracker*, then "you are a tracker guy and it gonna go fast". With *Sequencer64*, one creates several patterns, and then combines them. They can be layed out as a performance, and exported to a standard MIDI file.

*Seq24* spent some time inactive (2010 was the last major update, with a recent update to fix a MIDI clock issue). There are a number of forks for it on *GitHub*, and some conversions to code in other languages, and some patches. There is also a Gtkmm port to Windows. So, why bother creating yet another fork of *Seq24*?

## 1.2 Sequencer64: Why?

The first reason to reboot *Seq24* is consolidation of many of the features and fixes that it has accumulated in various forks over the years. Also, although "feature-complete", additional features would be useful.

Secondly, *Seq24* is a kind of "vi" of MIDI editing... spare, lean, and powerful. It deserves to be a living project. Without *Seq24* and its authors, *Sequencer64* would never have come into being.

### 1.3 Improvements

The following improvements are some that have been made in *Sequencer64*.

- Native JACK MIDI support!
- Tempo events are recordable and editable, in the first track.
- A "Tap" button sets the BPM by tapping a button or a key.
- Supports multiple screen-sets ("multi-set") or live-frame windows.
- Supports screen-set dimensions other than 8x4 ("variset").
- The *event editor* provides basic viewing and editing of MIDI events as text.
- *Sequencer64* reads SMF 0 MIDI files, split into slots based on channel number.
- Many new MIDI controls, such as coarse versus fine BPM control.
- Support for basic reading (not writing) of *Cakewalk* "WRK" files.
- Play-list support which can be controlled via MIDI.
- Some features ported from *Stazed*'s *Seq32* [26] project:
  - Chord-generation.
  - Pattern-transposing.
  - MIDI performance export to a standard MIDI file.
  - Extended undo/redo support, better detection of changes.
  - Splitting triggers at the nearest snap, if desired.
  - Modulating data events with a low-frequency oscillator (LFO).
  - Enhanced JACK transport support.
  - Diverting multi-channel MIDI into into channel-specific slots.
- Set Tempo and Time Signature events are handled, shown, and saved as standard MIDI data.
- Scale, key, and background sequences saved to the *Sequencer64* MIDI file format.
- Pattern coloring, saved in the sequence.
- Piano roll scrolling to keep up with the progress bar.
- Additional mouse and keystroke support for faster editing.
- The "user" configuration can be saved (`--user-save` option); it has *many* new settings.
- An option to map events to a single MIDI buss for testing.
- Support for additional PPQN values, or the PPQN of the MIDI file.
- Improvements in appearance, too many to mention here.
- Many, many bug fixes, faster than we create them!
- Recording of performance changes into the Song Editor.
- More musical scales have been added.
- A pause feature, with a pause key (".") and pause button.
- Internal improvements.
  - Provided a new, more MIDI-compliant output format for the MIDI files.
  - Reworked the MIDI event container, to speed the loading of a MIDI file.
  - Note-transpose now also works on aftertouch events.
  - Non-note events are copied/moved/pasted in the pattern editor.
  - Debian packaging added into the project.
  - The Qt 5 version (the future of *Sequencer64*) for Windows is built using Qmake and MingW.
  - *Sequencer64 for Windows* provides a portable zip package and NSIS-based installer.

For developers, *Sequencer64* is extremely customizable via C macros, by enabling/disabling options at build-configuration time, and by many command-line arguments. We cannot show all permutations of

settings in this document, so don't be surprised if some screenshots don't quite match one's setup. Distro maintainers might pick their favorite build configurations.

## 1.4 Document Structure

The structure of this document follows the user-interface of *Sequencer64*. The sections are provided in the order their contents appear in the user-interface of *Sequencer64*. To help the reader jump around this document, it provides multiple links, references, and index entries.

## 1.5 Let's Go!

Make sure no other sound application is running, for the first run. Start *Sequencer64* to use JACK for MIDI, or on *Windows*, just run it (`qpseq64.exe`; on *Windows*, PortMidi is used). The port settings will be different. Provide a MIDI file. On our system, the synthesizer (*Yoshimi*) comes up on MIDI buss 5; an option remaps all events to that buss:

```
$ seq64 --jack-midi --buss 5 contrib/midi/b4uacuse-seq24.midi
C:\> qpseq64 --buss 1  contrib/midi/b4uacuse-seq24.midi
```

If the `--alsa` option is used instead of `--jack-midi`, then the "JACK" button shows "ALSA" instead (Linux only). The following figure is for the Linux version.

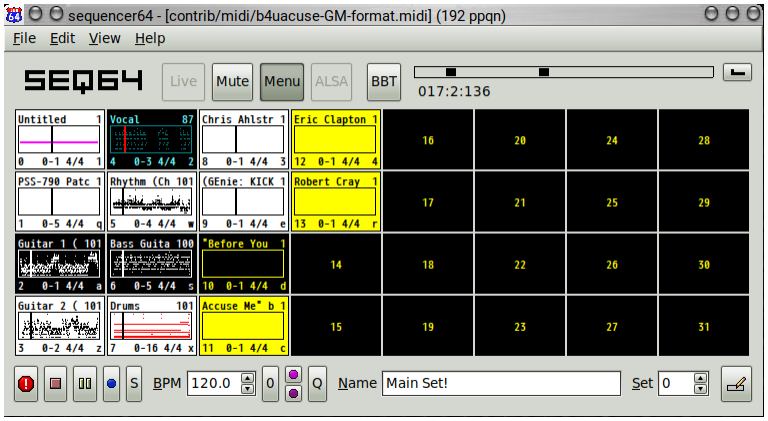



Figure 1: Sequencer64 Main Screen, Linux ALSA MIDI

The following figure shows the user-interface used for *Windows*. It uses the Qt 5 framework.

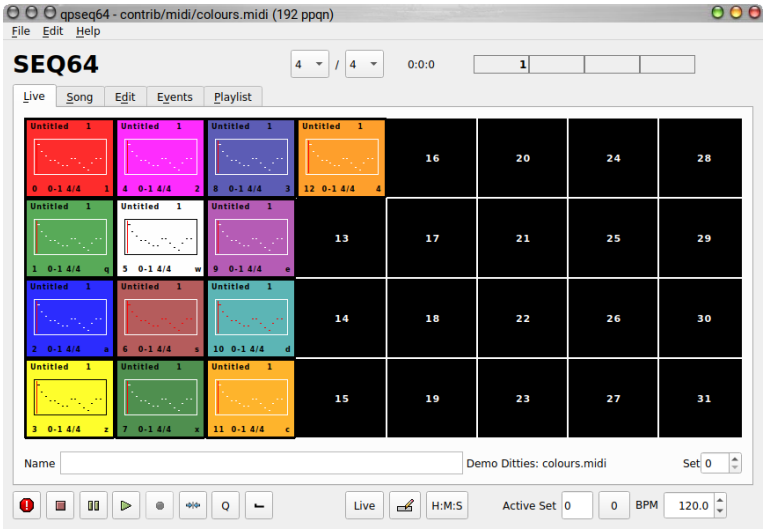



Figure 2: Sequencer64 Main Screen, Qt 5 Interface, With Colors

The *Sequencer64* main window appears, as shown above. These figures have some differences from the *Seq24* main window and from each other, but the functionality is about the same. Most features, including the "look" of the application, can be configured via the "rc" and "user" configuration files or command-line options. There are many new front-panel items in *Sequencer64*, but the Qt and Gtkmm user-interfaces differ in details.

- Control buttons:
  - Start, Stop, and Pause.
  - Toggle and show the status of "Live" mode versus "Song" mode.
  - Mute/show the mute status of all tracks.
  - Enable/disable the menu bar and show its status.
  - Set JACK Slave/Master transport, and ALSA/JACK (native) mode.
  - Set the kind of time display, between "bars:beat:ticks" and "hours:minute:seconds".
  - Panic button, to stop all tracks and turn off all notes.
  - Song-recording snap, the **S** button.
  - Tap Tempo, the **0** (zero) button.
  - Keep-queue toggling and status.
- Current time in bars, beats, and ticks.
- Song recording records all muting changes to the Song Editor.
- Log Tempo, which inserts the current tempo into the tempo track as an event.
- Tempo recording, which inserts all tempo changes as tempo events.

Many of these buttons have configurable keystrokes as well. See section 3.3 "Patterns / Bottom Panel" on page 52. Some of these items are not present in the Qt interface.

## 2 Menu

The *Sequencer64* menu ( Figure 1 "Sequencer64 Main Screen, Linux ALSA MIDI" on page 12) is simple, but important to understand.

## 2.1 Menu / File

The **File** menu is used to save and load MIDI files (Standard MIDI Format 0 or 1) and *Sequencer64* MIDI files. The *Sequencer64* menu entry contains the sub-items shown below. The next few sub-sections discuss the sub-items in the *File* sub-menu.

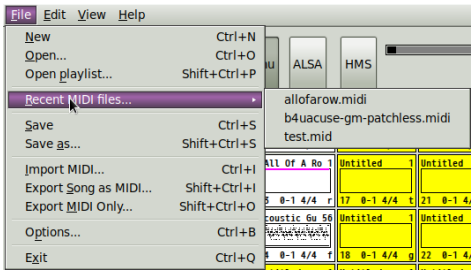



Figure 3: Sequencer64 File Menu Items

There are some minor differences between the *Qt* and the *Gtkmm* (shown in the figure) versions of this menu. We comment on differences only if important to do so.

1. **New**
2. **Open**
3. **Open Playlist**
4. **Recent MIDI files**
5. **Save**
6. **Save As**
7. **Export Song as MIDI**
8. **Export MIDI Only**
9. **Import MIDI**
10. **Options**
11. **Exit or Quit**

The Qt version of this menu is similar, except that the **Options...** menu is placed in **Edit / Preferences**.

## 2.2 Menu / File / New

The **New** menu entry clears out the current song or play-list. If unsaved changes are pending, the user is prompted to save the changes. Prompting for changes is more comprehensive than *Seq24*. However, when in doubt, save! Keep backups of your tunes!

### 2.2.1 Menu / File / Open

The **Open** menu entry opens a song (MIDI file), replacing the current song. It opens up a standard GTKmm-2.4 or Qt 5 file dialog:

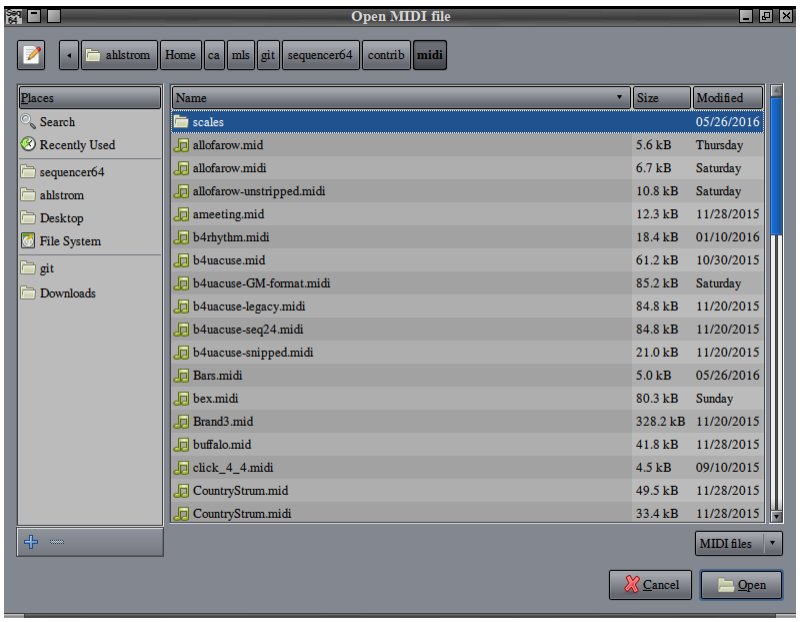



Figure 4: File / Open (Gtkmm version)

This dialog lets one type a file-name, highlighting the first file (if any) that matches the characters typed so far. *Sequencer64* can open regular MIDI files and Cakewalk WRK files. It will read and store various MIDI Meta events that were not supported before.

### 2.2.2 Menu / File / Open Playlist

The **Open Playlist...** menu entry opens a *Sequencer* play-list file. This file contains a list of "playlist sections", each listing a number of MIDI songs. These playlists and songs can be selected by the arrow keys or by MIDI control. See section 12 "Sequencer64 Play-Lists" on page 141.

Once activated, the current play-list file-name is saved in the `[playlist]` section of the "rc" configuration file when *Sequencer64* exits. The Qt user-interface will eventually support some editing of the play-list file. Currently, the user should understand the play-list format, and use a text editor to edit the play-list file.

### 2.2.3 Menu / File / Recent MIDI files

This menu entry provides a list of the last few MIDI files created or opened; play-list selections are *not* included. This list is saved in the `[recent-files]` section of the "rc" configuration file. In the menu, only the last part of the file-name is shown, but in the "rc" configuration file, the full path to the file-name is stored. This path is in "UNIX" format, using the forward slash, or solidus, as the path separator, even in *Windows*. Only unique entries are included in the recent-files list. The limit is 10 recent-file entries. This is another feature from *Kepler34* ([9]). Here is an example from an "rc" file:

```
[recent-files]
# Holds a list of the last few recently-loaded MIDI files.
3
/home/chris/git/sequencer64/data/b4uacuse-gm-patchless.midi
```

```
/home/chris/git/sequencer64/contrib/midi/colours.midi
/home/chris/git/sequencer64/Julian-data/TestBeeps.midi
```

### 2.2.4 Menu / File / Save and Save As

The **Save** menu entry saves the song under its current file-name. If there is no current file-name, then it opens up a standard file dialog to name and save the file. The **Save As** menu entry saves a song under a different name. It opens up the following standard file dialog, very similar to the **File Open** dialog, with an additional **Name** text-edit field.

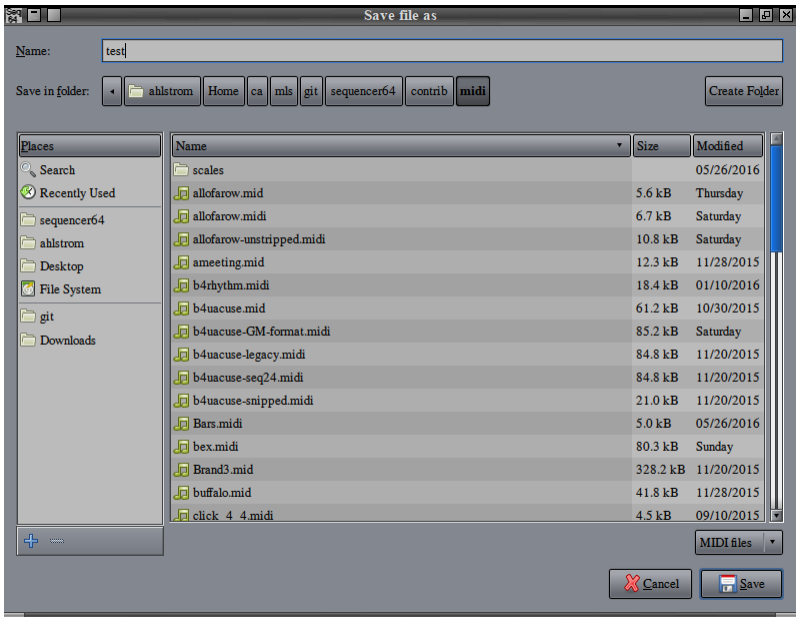



Figure 5: File / Save As (Gtkmm version)

To save a new file, or to save the current existing file to a new name, enter the name in the name field, without an extension. *Sequencer64* will append a `.midi` extension to the filename. The file will be saved in a format that the Linux *file* command will tag as something like:

```
myfile.midi: Standard MIDI data (format 1) using 16 tracks at 1/192
```

It looks like a simple MIDI file, and yet, if one re-opens it in *Sequencer64*, one sees that all of the labeling, pattern information, and song layout has been preserved in this file. This information is saved in a way that MIDI-compliant software should be able to use, or ignore without failure.

The output MIDI file after *Sequencer64* saves the original MIDI file is larger. After the last track in the file, a number of MIDI-compliant sequencer-specific (SeqSpec) items are saved, to preserve the extra information that *Sequencer64* adds. In legacy mode, *Sequencer64* saves this information in the same format as *Seq24*. Otherwise, it saves it in a more MIDI-compatible format. Normally, *Sequencer64* saves this information by marking each SeqSpec section as vendor-specific information, and marking this section as a regular MIDI track. The legacy and new formats of the final "track" are explained in section 18.2 "Legacy Proprietary Track Format" on page 173.

Meta events are now partially handled by *Sequencer64*. Meta events Set Tempo and Time Signature are now fully supported. Other Meta events, such as Meta MIDI Channel and Meta MIDI Port are now read as events, and are saved back when the file is saved. They cannot be edited in *Sequencer64*, but they are not lost. Please note that the channel and port meta events are considered *obsolete* in the MIDI standard.

### 2.2.5 Menu / File / Import MIDI

The **Import** menu entry imports an SMF 0 or SMF 1 MIDI file as one or more patterns, one pattern per track, into the specified screen-set. This functionality is explained in detail in section 7.1 "Import MIDI" on page 90.

### 2.2.6 Menu / File / Export Song as MIDI

Thanks to the *Seq32* project, the ability to export songs to MIDI format has been added. In this export, a complete song performance is recoded so that other MIDI sequencers can play the performance properly. This functionality is explained in detail in section 7.2 "Export Song as MIDI" on page 91.

### 2.2.7 Menu / File / Export MIDI Only

Sometimes it might be useful to export only the non-sequencer-specific (non-SeqSpec) data from a *Sequencer64* song, in order to reduce the size of the file or to accomodate non-compliant sequencers. This functionality is explained in detail in section 7.3 "Export MIDI Only" on page 94.

### 2.2.8 Menu / File / Options

In the *Portmidi / Windows / Qt 5* version of *Sequencer64*, the **Options** dialog has been moved to **Edit / Preferences**. See section 13.1.7 "Qt 5 Edit / Preferences" on page 152. There are still a few configuration items yet to be represented in that new user-interface.

**Options** provides a number of settings in one tabbed dialog, shown in the figures that follow. It allows one to set MIDI clocking, what incoming MIDI events control the sequencer, what keys are mapped to functions, how the mouse works, and some JACK parameters. Note that there is a new tab-page for **Ext Keys**, to support more keystroke controls.

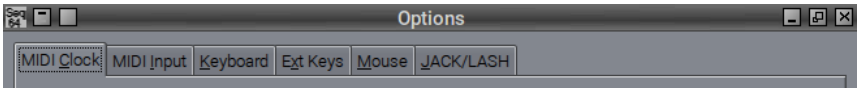



Figure 6: File / Options (Gtkmm Version)

#### 2.2.8.1 Menu / File / Options / MIDI Clock

The **MIDI Clock** tab provides a way to set MIDI clock for the available MIDI output busses. It configures to what busses the MIDI clock and data gets dumped. It also shows the devices that can play music. The items that appear in this tab depend the setup.

- What MIDI devices are connected to the computer. MIDI controllers, USB MIDI cables, applications with virtual ports, and other connected devices will add MIDI output devices (ports) to the system. In *Windows*, the available devices are shown as well.
- The setting of the "manual ALSA ports" option, which tells *Sequencer64* to set up virtual MIDI ports. It is enabled by the `--manual-alsa-ports` command-line option or the `[manual-alsa-ports]` section of the `sequencer64.rc` configuration file, described in section 10.12 ""rc" File / Manual ALSA Ports" on page 121.
- The setting of the *Sequencer64*-specific "reveal ALSA ports" option, `--reveal-alsa-ports` command-line option or the `[reveal-alsa-ports]` section of the `sequencer64.rc` configuration file, described in section 10.13 ""rc" File / Reveal ALSA Ports" on page 121.

For the current discussion, a USB MIDI cable was plugged into the system, and the *Timidity* and *Yoshimi* (in ALSA mode) software synthesizers were running. *Sequencer64* was also running, without virtual ports enabled, and `--alsa` turned on. Here are the devices shown when running *aplaymidi* from the command-line:

```
$ aplaymidi -l
 Port    Client name                      Port name
 14:0    Midi Through                     Midi Through Port-0
 24:0    E-MU XMidi1X1 Tab                E-MU XMidi1X1 Tab MIDI 1
128:0    TiMidity                         TiMidity port 0
128:1    TiMidity                         TiMidity port 1
128:2    TiMidity                         TiMidity port 2
128:3    TiMidity                         TiMidity port 3
129:0    seq64                            seq64 in
```

Note that *Yoshimi* does not appear. Perhaps `aplaymidi` does not subscribe properly to all ALSA notifications. *Sequencer64* will detect it. One can also run the following command instead:

```
$ aconnect -io
```

One other note... of late, we're seeing cases where running the *Timidity* daemon hides *Yoshimi*. Just be aware.

Turning to Figure 7 "MIDI Clock, Manual Option On (Gtkmm Version)" on page 19, with the option of "manual ALSA ports" (`-m` or `--manual-alsa-ports`) and ALSA in force, note the 16 devices provided by *Sequencer64*. This figure shows the result with the "manual ALSA option" turned on. Remember that this option also applies to the native JACK MIDI mode of *Sequencer64*.

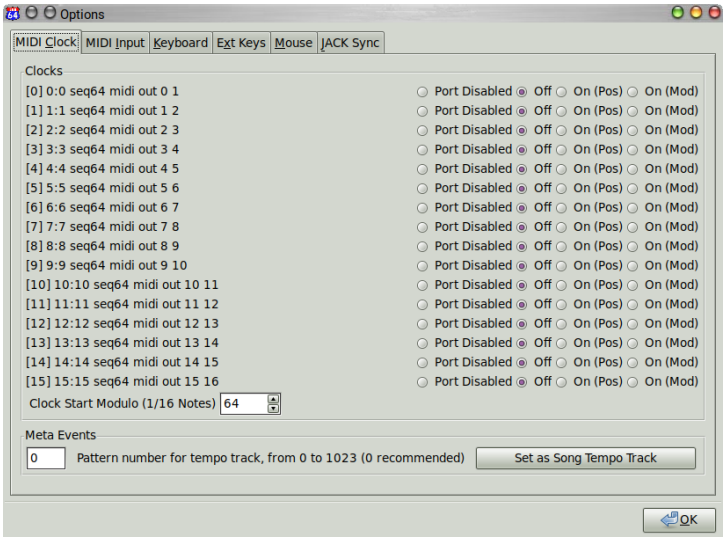



Figure 7: MIDI Clock, Manual Option On (Gtkmm Version)

It shows the 16 virtual MIDI output busses that *Sequencer64* can drive. One needs to use a JACK or ALSA MIDI connection application to connect a device on each of those outputs. The fact that the the buss names can start with different numbers, depending on the system setup, can complicate the playing of MIDI in this manner. Also, the "user" configuration file can change the visible names of the ports, causing further confusion. The following elements are present in this dialog:

1. **Index Number**
2. **Client Number**
3. **Port Number**
4. **Buss Name**
5. **Port Disabled**
6. **Off**
7. **On (Pos)**
8. **On (Mod)**
9. **Clock Start Modulo**

The format of the left side of the entry listing is like the following:

```
[4] 4:4 seq64 midi out 4:5
 ^   ^ ^ ^
 |   | | |
 |   | |  -------- Buss name
 |   |  ---------- Port number
 |    ------------ Client number
  --------------- Index number
```

**1. Index Number.** The number in square brackets is an ordinal indicating the position of the output buss in the list. It can be used with the `-b --buss --bus` option to redirect all output to that one buss, which is useful if only one buss is active, and the *Sequencer64* MIDI song patterns route to non-existent busses.

**2. Client Number.** The number that precedes the colon is the "client number". It is useful mainly in ALSA, where clients can have numbers like "14", "128", "129", etc. For native JACK mode, it matches the index number.

**3. Port Number.** The number that follows the colon is the "port number". It is useful mainly in ALSA. For native JACK mode, it matches the index number.

**4. Buss Name.** These labels indicate the output busses (ports) of *Sequencer64*. They range from **[1] seq64 1** to **[16] seq64 16**. in native JACK, when manual/virtual ports are active.

**5. Port Disabled.** The **Port Disabled** clock choice marks a port that one does not want to use or that the operating system (*Windows*, I'm looking at *you*!) is locking or disabling that output port. Normally, this inaccessible port would cause *Sequencer64* to exit. With the port disabled, the inaccessible port is ignored.

When the *Windows* version of *Sequencer64* (`qpseq64` is first started, it may error out. It will then write "erroneous.rc" and "erroneous.usr" configuration files, which can be examined to find the offending buss.

**6. Off.** Disables the MIDI *clock* for the given output buss. MIDI output is still sent to those ports, and each port that has a device connected to it will play music. Some synthesizers may require this setting.

**7. On (Pos).** MIDI clock will be sent to this buss. MIDI Song Position and MIDI Continue will be sent if playback starts at greater than tick 0 in Song mode. Otherwise, MIDI Start will be sent.

**8. On (Mod).** MIDI clock will be sent to this buss. MIDI Start will be sent, and clocking will begin once the Song Position has reached the start modulo of the specified size (see the next item's description). This setting is used for gear that does not respond to Song Position.

**9. Clock Start Modulo.** Clock Start Modulo (1/16 Notes). This value starts at 1 and ranges up to 16384, and defaults to 64. It is used by the **On (Mod)** setting discussed above. It is the `[midi-clock-mod-ticks]` option in the *Sequencer64* "rc" file as described in section 10.9 ""rc" File / MIDI Clock Mod Ticks" on page 120.

**10. Meta Events.** This section consists of one item, the Tempo Track number. It allows the user to move the tempo track from pattern 0 to another pattern. Changing this option is not recommended, since track 1 (0) is the official track for tempo events, but *Sequencer64* allows the user to record tempo events to another track. *Sequencer64* will process tempo events in any pattern. *Not supported yet in the Qt user-interface, but it can be set manually in the "rc" configuration file.*

In addition, the **Set as Song Tempo Track** button sets this track as part of the currently-load MIDI song file, and will be saved when exited. This value will override the global tempo-track value, which is stored in the 'rc' configuration file. However, if 0, it will be ignored, so that the global value will take hold. See section 10.5 ""rc" File / MIDI-Meta-Events Section" on page 116, which discusses this setting in the "rc" configuration file.

One thing to remember about tempo... there is a "global" tempo which is saved as part of the song in a SeqSpec section. However, that tempo is not saved in the tempo track. In order to do so, one can click the **Log Tempo** button to write it to the tempo track as a tempo event.

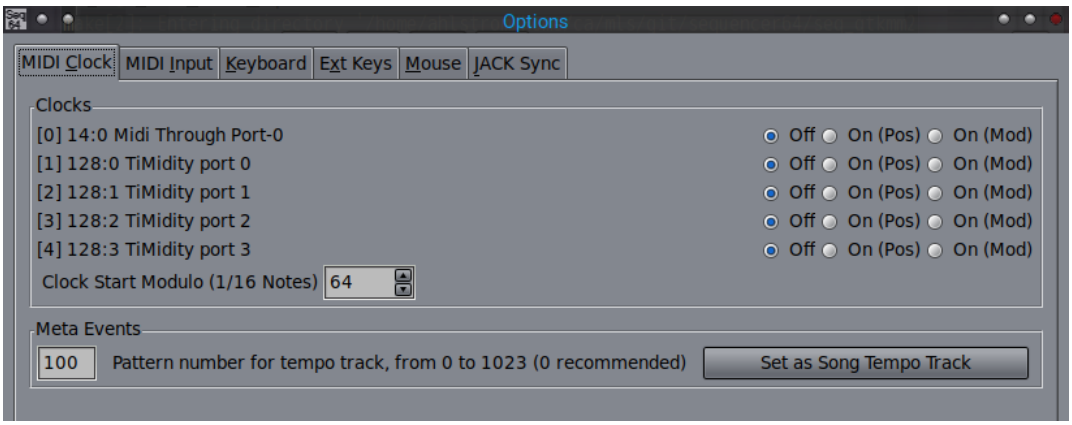



Figure 8: MIDI Clock, Manual Option Off (ALSA View, Old Screenshot)

In the figure above, with the "manual ALSA option" is turned off, and all of the real (non-virtual) devices that can be driven by MIDI output are shown, including the MIDI Thru port, and the four ports provided by *Timidity* on our setup. (The **Port Disabled** column is missing from this old screenshot. We need to fix that someday.)

See section 19.2 "Sequencer64 Native JACK MIDI" on page 185, for a lot more information about native JACK support, and examples of JACK MIDI ports and connections.

There is currently no user-interface item corresponding to the "manual ALSA" command-line and "rc" configuration file option. We should rename this option to "virtual" eventually, since it can also apply to JACK MIDI.

#### 2.2.8.2 Menu / File / Options / MIDI Input

To allow *Sequencer64* to record MIDI from MIDI devices such as controllers and keyboards, the output of the ALSA MIDI recording command-line application is relevant:

```
$ arecordmidi -l
 Port    Client name                      Port name
 14:0    Midi Through                     Midi Through Port-0
 24:0    USB2.0-MIDI                      USB2.0-MIDI MIDI 1
129:1    seq64                            seq64 midi out 0
129:2    seq64                            seq64 midi out 1
 . . .   . . .                               . . .
129:16   seq64                            seq64 midi out 15
```

We see that we can record MIDI from the MIDI Thru port, from the USB MIDI cable, and MIDI from any of the 16 output ports provided by the manual ALSA port mode of *Sequencer64*.

If the "manual ALSA ports" option is turned *off* (e.g. by using the `-a` option), then the only item in the **MIDI Input** tab is the single MIDI input buss provided by *Sequencer64*: **[0] seq64 0**. (The figure shown is currently out-of-date.)

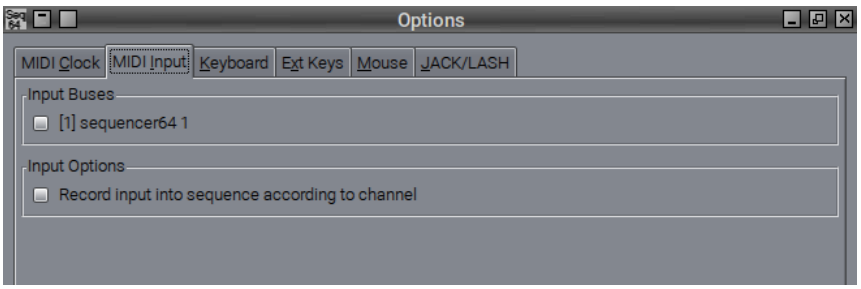



Figure 9: MIDI Input, Manual Ports Off (Condensed View)

Any item checked allows *Sequencer64* to record MIDI from another source, which must be connected to this port via another application).

**Warning:** If the `[user-midi-bus-definitions]` value in the "user" configuration file is non-zero, and the corresponding number of `[user-midi-bus-N]` settings are provided, then the list of existing hardware will be ignored, and those values will be shown instead. (This feature can be overridden with the `--reveal-alsa-ports` (`-r`) option.)

If the "auto ALSA ports" option is turned on, via the `-a` or `--auto-alsa-ports` option, then the input ports from the system are shown:

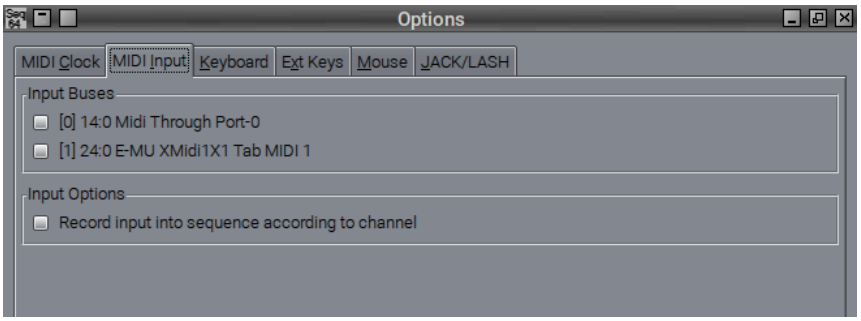



Figure 10: MIDI Input, `-a` Option (Condensed View)

For example, one could check input #1 to have *Sequencer64* record MIDI from an old-fashioned MIDI keyboard that is connected to another USB MIDI cable (the *E-MU Xmidi*). If the keyboard didn't have a sound generator, one would also want *Sequencer64* to pass this MIDI on to a sound generator, such as a software or hardware synthesizer attached to one of the ports shown in Figure 8 "MIDI Clock, Manual Option Off (ALSA View, Old Screenshot)" on page 21.

**Warning:** The "user" configuration file can override what is actually displayed as hardware. If you define these sections, they should match your hardware exactly, and your hardware should not change from session to session.

Note the two sections of this configuration page:

**Input Buses** delineates the MIDI input devices as noted above. **Input Options** adds further refinements to MIDI input.

**Record input into sequence according to channel** causes MIDI input with multiple channels to be distributed to each sequence according to MIDI channel number. When disabled, the legacy recording behavior dumps all data into the current sequence, regardless of channel.

#### 2.2.8.3 Menu / File / Options / Keyboard

*Seq24* allows extensive use of keyboard shortcuts to make operations go faster than with a mouse, and *Sequencer64* extends that tradition. The **Keyboard** tab (currently in Gtkmm only) allows for the configuration of these keyboard shortcuts.

These settings can also be modified by editing the appropriate "rc" configuration file, stored in one of the following directories, depending on the operating system:

```
/home/username/.config/sequencer64
C:/Users/username/AppData/Local/sequencer64
```

**Warning:** There are a number of "gotchas" to be aware of when assigning keys to the fields in the **Keyboard** tab:

- This configuration dialog is not yet present in the **Qt 5** version of *Sequencer64*. For now, one has to edit the "rc" file to configure the keystrokes. It is easiest to just use the sample `qpseq64.rc` or `qseq64.rc` from the `data` directory in the source-code package. Another option is to just run *Sequencer64* the first time and tweak the "rc" and "usr" files that are created in the directories noted above. Internally, the Qt key-codes are remapped to Gtk key-codes.
- Whenever one of the text fields in this dialog has the focus (and that is usually the case), then *any* keystroke, including keys like `Ctrl`, `Alt`, and `Super` (also known as Mod4 or the Windows key), can alter the value of a field to that of the keystroke. This change is very easy to do accidentally! Use the mouse to move this window and to click its **OK** button!
- Some of the keys traditionally used (or used by default) for control have been adapted for other uses, and are not configurable. One example is `Ctrl-L`, which brings up the learn mode that can be started using the "L" button or the "glearn" (group-learn) MIDI control. Some other hard-wired keystrokes are the "arrow" keys or "page up/down" keys.
- *Sequencer64* has appropriated the Shift key so that it modifies a click on a pattern so that all of the other patterns are *toggled*. Therefore, using characters that require the Shift key while clicking, such as { and }, when used to set the **Replace** function, becomes surprising. Instead, look to the remaining keys: `F11`, `F12`, and the "keypad" keys if more options are wanted. Be sure to look at the **Ext Keys** tab to see what other keys are in use. Also, for the group-learn feature, the `Shift` key is automatically enabled, using an "auto-shift" feature.

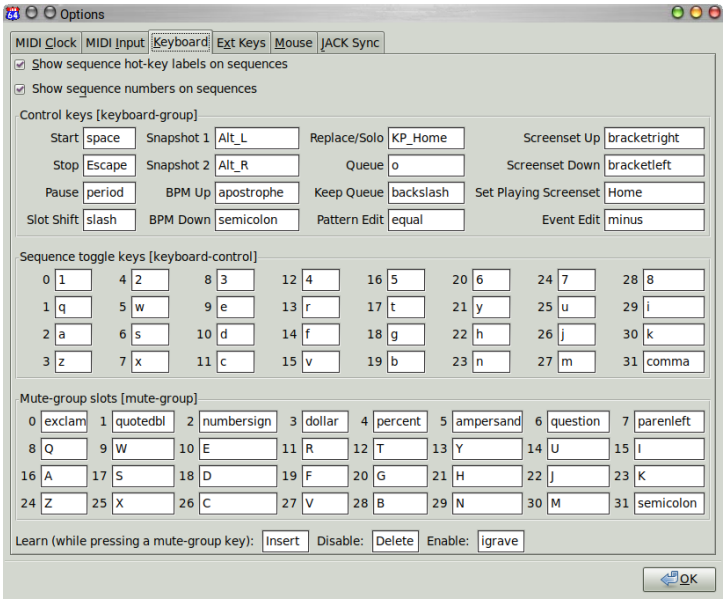



Figure 11: File / Options / Keyboard

`[keyboard-control]`. We won't attempt to cover every user-interface item in this busy dialog, just the categories. Some items might be discussed in other parts of this manual.

There are some things to note in the `[keyboard-control]` section. First, it is laid out like the main patterns panel, in an 8 x 4 grid. The keys in there correspond directly to the patterns panel slot, and the conventional keyboard layout is shown. Second, if one is looking for more keys to map to other functions, the default layout keeps open the following keys: `9 o l 0 p`.

The `[mute-group]` section is laid out in a similar manner, except that the **upper-case** versions of the keys are used. Additional keys are available for other functions: `( O L ) P`. Currently, one must be very carefully about assigning the same key to different functions. Confusion will ensue! We've done it!

An additional key definition is shown for the Pause key. By default, the Pause key is the period (”.”). An old version of the ”rc” file is automatically fixed to include this new option.

New features try to achieve being able to edit a pattern using only the keyboard. *Sequencer64* now supports two modifier keys. The first modifier key causes the usual pattern-toggle key (hot-key) for a given slot to instead bring up the pattern editor. By default, this key is the equals (”=”) key. The second modifier key causes the usual pattern-toggle key (hot-key) for a given slot to instead bring up the event editor. By default, this key is the minus (”-”) key. These keys are configurable in the **File / Options / Ext Keys** page.

To continue with a listing of the keyboard options:

1. **Show sequence hot-key labels on sequences**
2. **Show sequence numbers on sequences**
3. **Control keys [keyboard-group]**
4. **Sequence toggle keys [keyboard-control]**
5. **Mute-group slots [mute-group]**
6. **Learn**
7. **Disable**
8. **Enable**

These categories are described below.

**1. Show key labels on sequence.** This option shows the key labels in the lower-right corner of each loop/pattern slot in the Patterns window (the main window). It is useful for live playback and control of a song. It is configurable in the "rc" configuration file. It also enables the display of the pattern length, in measures, at the top right of the pattern slot.

**2. Show sequence numbers on sequence.** If this option is on, the empty slots in the pattern window show the prospective sequence number. See the following figure for one look of this feature.

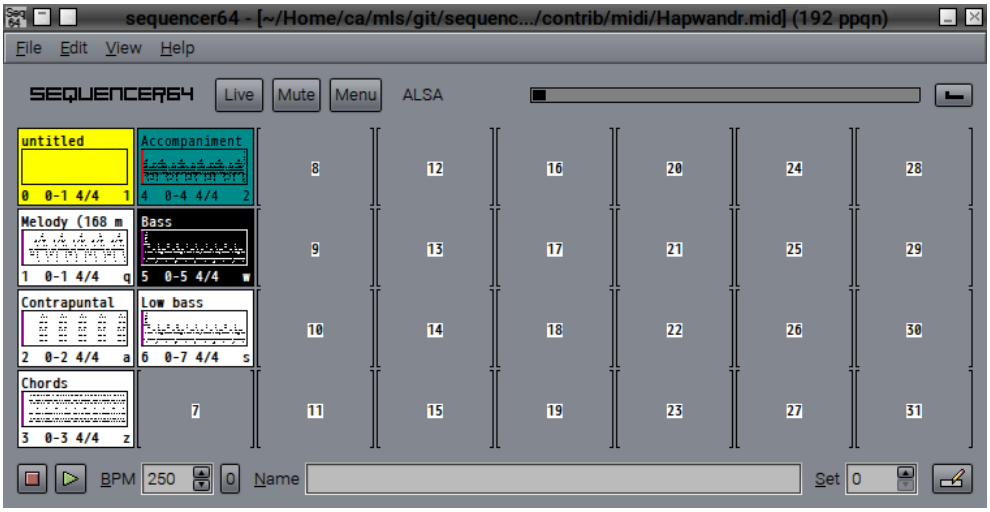



Figure 12: Pattern Window with One Kind of Numbering

The option also changes the visibility of sequence numbers in active sequences and in the Song Editor's names column. If one doesn't like it, turn off the option in the "rc" configuration file, or try other grid options in the "user" configuration file.

**3. Control keys.** `[keyboard-group]`. This block of fields in the **Options / Keyboard** tab provides shortcut keys for many operations of *Sequencer64*. There is a default mapping built into *Sequencer64*, but open the keyboard options tab to see the actual values.

1. **Start**.
2. **Stop**.
3. **Pause**.
4. **Slot Shift**.
5. **Snapshot 1**.
6. **Snapshot 2**.
7. **bpm up**.
8. **bpm down**.
9. **Replace**.
10. **Queue**.
11. **Keep queue**.
12. **Screenset down**.
13. **Screenset up**.
14. **Set playing screenset**.

Some of the keys have positional mnemonic value. For example, for BPM control, the semicolon is at the left (down), and the apostrophe is at the right (up). Note that the keys definable in this tab are only

a subset of the various keys that can be used, especially keys used with the `Ctrl` key or other modifier keys.

The **slot shift** key is useful when using pattern grids larger than 8 x 4 patterns. Pressing the slot-shift key basically adds 32 to the pattern number of the slot-key that is pressed. Not all builds of *Sequencer64* support this option.

A **snapshot** is a briefly-preserved state of the patterns. One can press a snapshot key, change the state of the patterns for live playback, and then release the snapshot key to revert to the state when the snapshot key was first pressed.

To **queue** a pattern means to ready it for playback upon the next repeat of a pattern. A pattern can be armed immediately with a hot-key, or it can be queued to play back the next time the pattern repeats. A pattern can be queued by holding the queue key (defined in **File / Options / Keyboard / queue**) and pressing a pattern-slot hot-key. Instead of the pattern turning on immediately, it turns on at the next repeat of the pattern.

**Keep queue** allows the queue to be held without holding down the queue button the whole time. First, press the keep-queue key (defined in **File / Options / Keyboard / Keep queue**). Now, hitting any of the slot hot-keys, no matter how many, sets up the corresponding pattern slot to be queued. Also, in keep-queue mode, clicking on the pattern slot will queue the pattern. The keep-queue mode is disabled by hitting the "queue" key again (any currently active queues remain active until finished). There is also a "Q" button to toggle the keep-queue status.

Be sure to note the new option, **one-shot queue**, in the extended-keys section (section 3.2.1 "Pattern Slot" on page 40).

**4. Sequence toggle keys.** Each of these keys toggles the playing/muting of one of the 32 loop/pattern boxes. These keys are layed out logically on the keyboard, and can also be shown in each loop/pattern box. No need to list them all here! Please note that we often call them "shortcut keys" or "hot-keys" where the context makes it clear that they apply to the armed/unarmed state of a pattern.

**5. Mute-group slots.** There can be up to 32 mute-groups. When activated, a mute-group sets the muted/unmuted status of the current "playing set" to the pattern-muting statuses of the selected mute-group. Each of these keys operates on the mute-grouping of one of the 32 stored mute groups. These keys are layed out logically on the keyboard. No need to list them all here! Generally, they are the shifted versions of the keyboard keys used as hot-keys for the patterns. Note that a mute-group key will be memorized only when *Sequencer64* is in *group-learn* mode.

**6. Learn.**

To define the group of patterns for one mute group, press and hold the configured Learn key (the `Insert` key by default, the `Ctrl-L` key, or the "L" button in the user-interface. Simultaneously (not needed with the "L" button), press one of the mute group keys: *Sequencer64* will save the currently-playing pattern slots into the corresponding mute group. The default mute group keys must be the shifted version of the key, but one does not need the `Shift` key while pressing `Insert` to learn the group, only to trigger it. *Sequencer64* will automatically assign the corresponding key with `Shift` activated. Try pressing the `Shift` key in Learn mode and see what happens!

Group-mute can be globally enabled or disabled (with default keys apostrophe ' and igrave or grave `). So make sure it is enabled before trying to use it.

**7. Disable.** It is the **apostrophe** key by default. This key is the *group off* key.

**8. Enable.** It is the **igrave** (back-tick) key by default. This key is the *group on* key.

#### 2.2.8.4 Menu / File / Options / Ext Keys

`[extended-keys]`. A number of additional functions have been added to *Sequencer64*, and keystrokes have been provided for those new functions, in the **Ext Keys** page. This page is needed because the original page is completely filled. The new tab has its own section in the "rc" file.

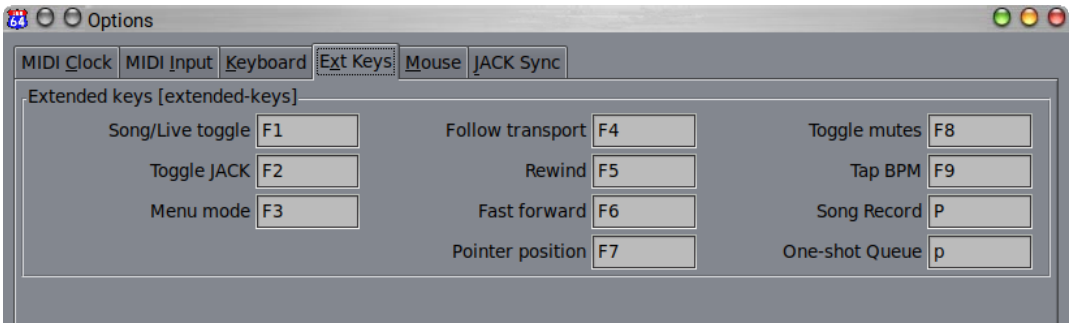



Figure 13: File / Options / Ext Keys (Condensed View)

**1. Ext Keys.** This block of fields in the **Options / Ext Keys** tab provides shortcut keys for more operations of *Sequencer64*, many of them ported from *Seq32* or *Kepler34*.

1. **Song/Live toggle**.
2. **Toggle JACK**.
3. **Menu mode**.
4. **Follow transport**.
5. **Fast forward**.
6. **Rewind**.
7. **Pointer Position**.
8. **Toggle mutes**.
9. **Tap BPM**.
10. **Song Record**.
11. **One-shot Queue**. This new feature allows one-shot queuing of a pattern.

Most of these extended keys implement operations performed with button presses. Some of the new keystrokes may not have a corresponding button. These values are saved as the `[extended-keys]` section of the "rc" configuration file.

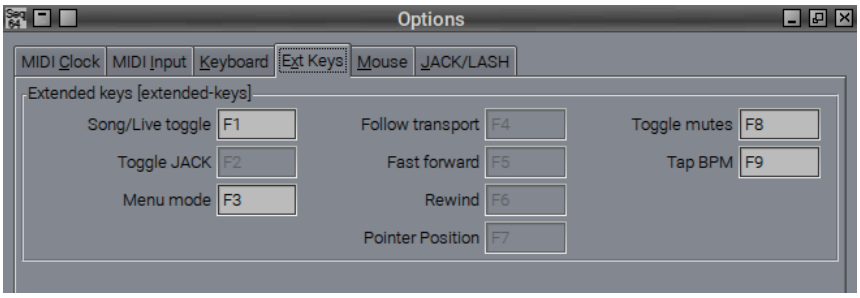



Figure 14: File / Options / Ext Keys (Disabled)

Note the **Song/Live toggle** key. The *song mode* normally is in effect only when playback is started from the **Song Editor**. Now this mode can be used from any window, if enabled by pressing this key.

There is also a button in the main window for this function, which shows the current state of this flag. Note that this flag is also stored in the "rc" configuration file, as well as this hot-key value, which defaults to `F1`.

The *JACK mode* is set via the **File / Options / JACK / JACK Connect** or **JACK Disconnect** buttons. This keystroke will toggle between JACK connect and JACK disconnect. The **Song Editor** will also have a **JACK** button. The hot-key for this function defaults to `F2`.

The *menu mode* indicates if the main menu of the main window is accessible or not. It is disabled during playback so that more hot-keys can be used without triggering menu functions. It can also be disabled by the user; the default hot-key is `F3`. This feature is needed because the original *Seq24* had numerous conflicts between the menu key bindings and the default key bindings for the main window.

*Follow transport* is a feature ported from *Seq32*. The default key is `F4`. It determines if *Sequencer64* follows JACK transport.

*Fast forward* is a feature ported from *Seq32*. The default key is `F5`. While this key is held, the song pointer will fast-forward through the song. This feature does not have a corresponding button. This feature requires that the *Seq32* transport option be enabled at build time.

*Rewind* is a feature ported from *Seq32*. The default key is `F6`. While this key is held, the song pointer will rewind. This feature does not have a corresponding button. This feature requires that the *Seq32* transport option be enabled at build time (and now that is the default). Be sure not to use the following keys, which are already hardwired for other functions in the Pattern Editor and Song Editor:

- `p`. Paint mode.
- `x`. Escape paint mode.

*Pointer position* is a feature ported from *Seq32*. The default key is `F7`. When this key is pressed, the song pointer will move to the current position of the mouse, snapped. This feature does not have a corresponding button.

*Toggle mutes* toggles the mute status of every pattern on every screen-set. It corresponds to the **Edit / Toggle mute all tracks** or the **Song / Toggle All Tracks** menu entries. There is also a button in the main window for this function, which shows the current state of this flag. Note that this hot-key value is stored in the "rc" configuration file, and defaults to `F8`.

*Tap bpm* allows the user to "tap" in time with some other music, and see the tap sequence translated into beats/minute (BPM). There is also a "0" button for this function. After 5 seconds, this feature resets automatically, so the user can try again if not satisfied. At least two taps are needed for the BPM to be registered.

#### 2.2.8.5 Menu / File / Options / Mouse

This item selects the mouse-interaction method.

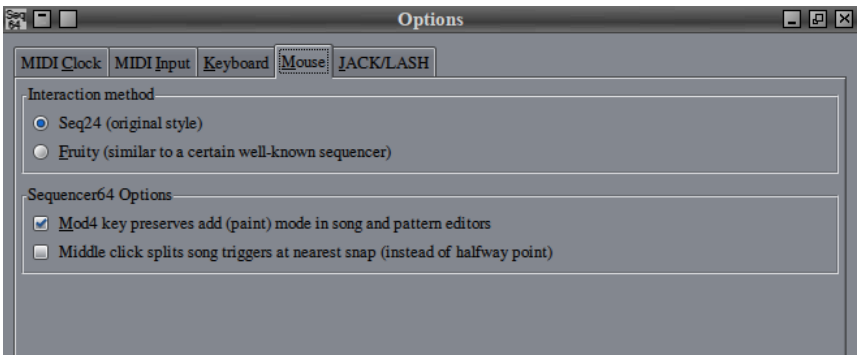



Figure 15: File / Options / Mouse (Condensed View)

**Interaction Method**

The default mouse interaction method is **Seq24 (original style)**. The alternate mouse interaction method is **Fruity (similar to a certain well known sequencer)**. The "Fruity" interaction method is currently only available in the Gtkmm-2.4 user-interface, and it is not comprehensive. For Qt 5, only the original style is available, at this time.

The alternate method is presumably that of the *Fruity Loops* (now *FL Studio*) sequencer. The fruity mode seems to involve the following rules:

- **Left-click left side**. Begin a grow/shrink operation for the left side. However, even in *Seq24*, this action is *broken*. It does allow one to move the note, however.
- **Left-click right side**. Begin a grow/shrink operation for the right side.
- **Left-click middle**. Move the object. To clarify, each note image has an invisible "handle" on the left or the right side, with the middle providing a third area of interaction in the "fruity" mode.
- **Left-click**. Add an event if nothing selected.
- **Middle-click**. Split the note?

There may be a few more rules to add, when time allows.

The *Seq24* note-editing style is as expected for basic actions such as selecting and moving notes using the left mouse button. Drawing a note or event is a bit different, in that one must first *click and hold* the right mouse button, and then *click and drag* the right mouse button to insert notes, Notes are inserted to be at the current length and grid-snap values for the sequence editor for as long as the buttons are pressed. Notes are inserted only up to the specified sequence length. Once notes are inserted, moving the mouse with the left button still held down moves the notes to the new note value of the mouse. If one releases the left button, then presses and holds it again, more notes will be added in the same way. This is unconventional, but a powerful way to layer notes in a short sequence. We call it the "draw mode" or "paint mode". Drawing/painting can also be done while the sequence is playing, and notes will be added to be played the next time the progress bar crosses them.

**Mod4 key preserves add (paint) mode in song and pattern editors**. In order to work with trackpads that don't permit simultaneous left and right clicks, the "Seq24" mode of mouse interaction can be modified in the Pattern or Song editors so that the Mod4 key (Super or Windows key) can be pressed when releasing the right mouse button. This keeps the mouse in note-add mode. Another right-click, without pressing Mod4, will exit this mode.

This option will not interfere with the Mod4 key being set in the **Keyboard** option tab, since the keys there mainly apply to the Patterns Panel (main window), not the pattern-editor window.

Middle click splits song triggers at nearest snap (instead of the halfway point).

Another way to turn on the paint mode has been added. To turn on the paint mode, press the `p` key while in the sequence editor. This is just like pressing the right mouse button, but the draw/paint mode stays on. To get out of the paint mode, press the `x` key while in the sequence editor. These keys, however, do not work while the sequence is playing.

Note that some *Sequencer64* windows can use the ctrl-left-click as a middle click.

#### 2.2.8.6 Menu / File / Options / Jack Sync

This tab sets up JACK transport, if *Sequencer64* was built with JACK support. This tab also sets up options for using LASH session management, *if Sequencer64* was built with LASH support, which is no longer the default, even though it is shown in the figure below.

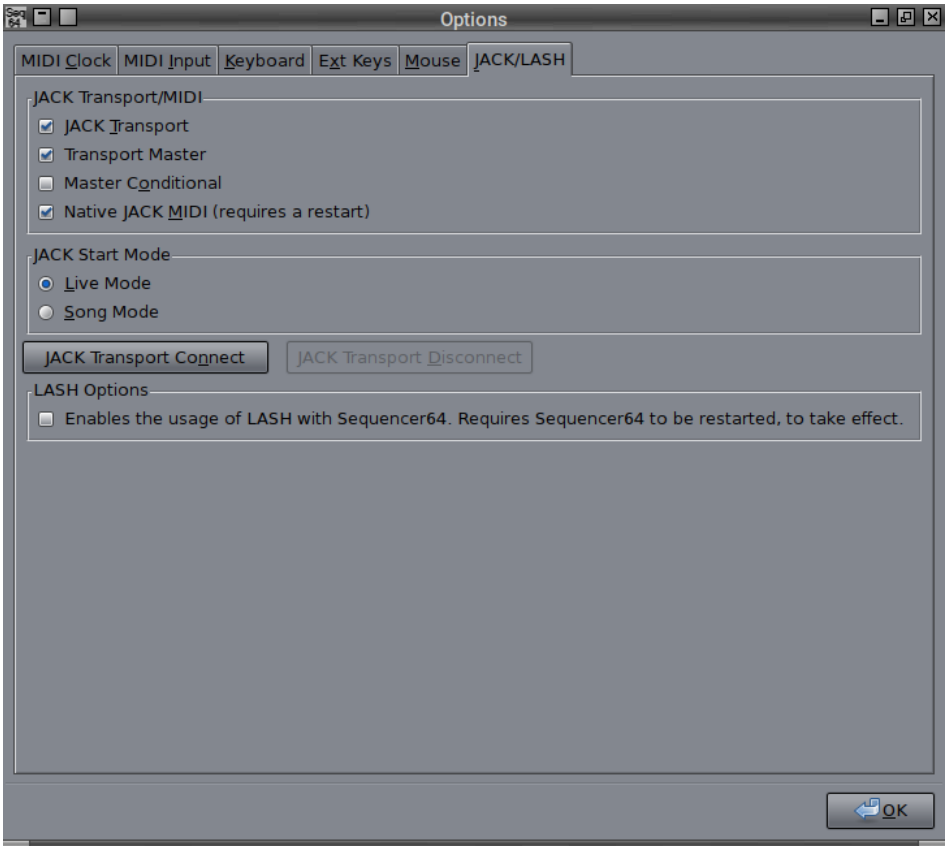



Figure 16: File / Options / JACK

The main sections in this dialog are:

1. **JACK Transport/MIDI**
2. **JACK Start Mode**
3. **JACK Transport Connect and Disconnect**
4. **LASH Options**

**1. Transport/MIDI.** These settings are stored in the "rc" file settings group `[jack-transport]`. See section 10.8 ""rc" File / JACK Transport" on page 119, which describes this configuration option. This items collects the following settings:

- **Jack Transport**. Enables slave synchronization with JACK Transport. The command-line option is `--jack-transport`. The behavior of this mode of operation is perhaps not quite correct. Even as a slave, *Sequencer64* can start and stop playback. Note that this option cannot be disabled via the mouse if the **Transport Master** option is enabled. Disable that one first.
- **Transport Master**. *Sequencer64* will attempt to serve as the JACK Master. The command-line option is `--jack-master`. **Tip**: *Sequencer64* generally works better as JACK Master. If this option is enabled the **JACK Transport** option is automatically enabled as well.
- **Master Conditional**. *Sequencer64* will fail to serve as the JACK Master if there is already a Master. The command-line option is `--jack-master-cond`. If this option is enabled the **JACK Transport** option is automatically enabled as well.
- **Native JACK MIDI**. This option is for the `seq64` version of *Sequencer64*. If set, MIDI input and output use native JACK MIDI, rather than ALSA. However, if JACK is not running on the system, then `seq64` will fall back to ALSA mode. The command-line option is `--jack-midi`.

If one makes a change in the JACK transport settings, it is best to then press the **JACK Transport Disconnect** button, then the **JACK Transport Connect** button. Another option is to restart *Sequencer64*... the settings are automatically saved when *Sequencer64* exits.

**2. JACK Start mode.** This item collects the following settings, also stored in the "rc" file settings group `[jack-transport]`.

- **Live Mode**. Playback will be in live mode. Use this option to allow muting and unmuting of patterns. This option might also be called "non-song mode". The command-line option is `--jack-start-mode 0`.
- **Song Mode**. Playback will use only the Song Editor's data. The command-line option is `--jack-start-mode 1`.

*Sequencer64* also selects the playback modes according to which window started the playback, reverting back to legacy *Seq24* behavior. *The main window*, or pattern window, causes playback to be in live mode. The user can arm and mute patterns in the main window by clicking on sequences, using their hot-keys, and by using the group-mode and learn-mode features. The song editor causes playback to be in performance mode, also known as "playback mode", or **Song** mode.

**3. Connect.** Connect to JACK Sync. This button is useful to restart JACK sync when making changes to it, or when *Sequencer64* was started in ALSA mode.

**4. Disconnect.** Disconnect from JACK Sync. This button is useful to stop JACK sync when making changes to it.

**5. LASH Options.** Currently contains only one item, which enables the usage of LASH session management. Currently, *Sequencer64* needs to be restarted to complete the enabling or disabling of LASH support. Like the rest of the options, this one is written to the "rc" configuration file. However, LASH is no longer supported in the default build.

Finally, there is a new button (labelled *Master* in the following figure) in the main window to bring up directly the **JACK** (or **JACK/LASH**) page.

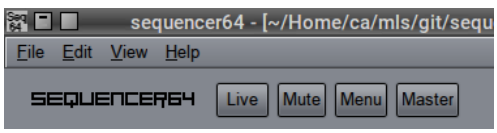



Figure 17: JACK Connection Button

This button not only brings up the JACK page, but also shows the current status of the MIDI connection: **Master** (JACK Transport Master), **Slave** (JACK Transport Slave), **JACK** (native JACK MIDI, overrides any transport label), and **ALSA** (overridden by any transport label). The `Ctrl-P` key will also bring up this page.

One thing to note is that, while playing, the JACK/ALSA button is disabled. However, one can still get to the JACK options via the main File menu. JACK connection and disconnection are disabled during playback, but the buttons don't yet reflect that status.

## 2.3 Menu / Edit

The **Edit** menu has undergone some expansion lately.

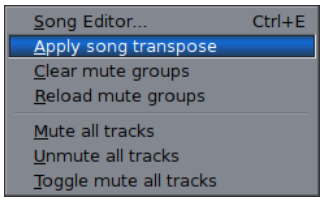



Figure 18: Edit Menu

1. **Preferences...** (only in the Qt user-interface)
2. **Song Editor...**
3. **Apply song transpose**
4. **Clear mute groups**
5. **Reload mute groups**
6. **Mute all tracks**
7. **Unute all tracks**
8. **Toggle mute all tracks**

**1. Preferences.** In the Qt user-interface, there is a **Preferences** menu entry, corresponding to the Gtkmm user-interface's **File / Options** menu entry.

**2. Song Editor.** This item is the same as the **View / Song Editor toggle** menu entry. It toggles the presence of the main song editor.

**3. Apply song transpose.** Selecting this item applies the song transposition value to **all** sequences/patterns that are marked as transposable. (Normally, drum tracks are *not* transposable). This actively changes the note/pitch value of all note and aftertouch events in the pattern. For the setting of song transpose, see section 5 "Song Editor" on page 77, for more information. Also note that transpose can be enabled in the patterns panel for each pattern (see section 3.2.2 "Pattern" on page 44) and in the sequence editor (see section 4 "Pattern Editor" on page 56).

**4. Clear mute groups.** A feature of *Seq24* and *Sequencer64* is that the mute groups are saved in both the "rc" file (see section 10.3 ""rc" File / Mute-Group Section" on page 115) *and* in the "MIDI" file (see section 18.2 "Legacy Proprietary Track Format" on page 173).

This menu entry clears them. If this resulted in any mute-group sequences status being set to false, then the user is prompted to save the MIDI file, so that it will no longer have any mute-group information. And then, if the application exits, the cleared mute-group information is also saved to the "rc" file.

**5. Reload mute groups.** This menu entry reloads the mute-groups from the "rc" file. So, if one loads a MIDI file that has its own mute groups that one does not like, this command will restore one's favorite mute-grouping from the "rc" file.

**6. Mute all tracks.** This menu entry, available only in **Live** mode, immediately mutes *all* patterns in the entire song.

**7. Unmute all tracks.** This menu entry, available only in **Live** mode, immediately unmutes *all* patterns in the entire song.

**8. Toggle mute all tracks.** This option toggles the mute/armed status of **all** tracks. It is only available in **Live** mode, which overrides **Song** mode even if the Song Editor is focussed. *Do not confuse it with the main **Mute** button, which toggles the status only of the tracks that are armed and remembers them.*

## 2.4 Menu / View

If the "allow two perfedits" option is turned off in the "user" configuration file, this menu item has only one entry, **Song Editor**, which is already covered by a button at the bottom of the Patterns window. Selecting this item bring up the Song Editor window. See Figure 67 "Song Editor Window" on page 78. The Song Editor window can also be brought up via the Ctrl-E key.

If the **allow two perfedits** option is turned on in the "user" configuration file, this menu item has two entries (Gtkmm only), as shown in the following figure:

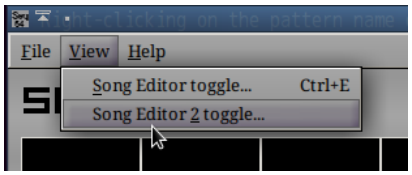



Figure 19: Dual Song Editor Entries in (Gtkmm) View Menu

Note that only the first Song Editor has a user-interface button and a hot-key. Also note that there can be issues bringing up the second song-editor with the hot-key. The menu entry will always work. If two song editors are up, they each track any changes made in the other song editor. But the main purpose of two song editors is to arrange two different parts of the performance at the same time when not all the patterns will fit in one window.

In the Qt user-interface, there is a **Song** tab in the main window. But there is also song-editor button in the main window and a menu entry in **Edit / Song Editor**, which bring up a separate window for the song-editor. Do not try to use both windows for song-editing at the same time; they are not synchronized.

## 2.5 Menu / Help / About...

This menu entry shows the "About" dialog.

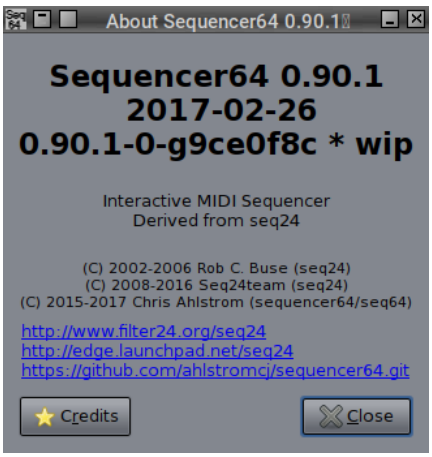



Figure 20: Help / About

The Qt version is slightly different.

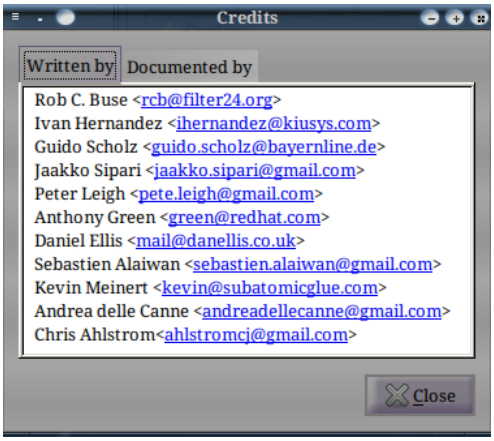



Figure 21: Help Credits

Shows who has worked on the program, with the original author at the top of the list.

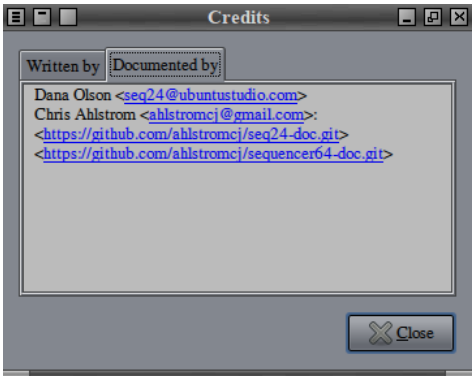



Figure 22: Help Documentation

Shows who has documented this project.

### 2.6 Menu / Help / Build Info...

This menu entry shows the "Build Info" dialog. This list of build options enabled in the current application is the same list that it generated via this command line:

```
$ seq64 --version
```

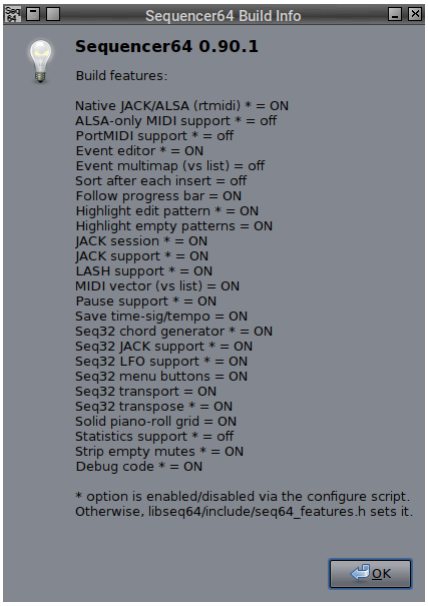



Figure 23: Help / Build Info

# 3 Patterns Panel

*Sequencer64* works with patterns (loops, tracks, or sequences) that are repeated throughout a song. One composes and edits small patterns, and combines them to create a full song. This is a powerful way to work, and makes one productive within an hour.

The *Sequencer64 Patterns Panel* is the **main window** of *Sequencer64*. See Figure 1 "Sequencer64 Main Screen, Linux ALSA MIDI" on page 12. It is called the "main window" or the "patterns window". It is here one creates a set of patterns (see section 16.1.17 "Concepts / Terms / screen set" on page 164), manages the configuration, controls the playback rate, adds tempo events, and opens the pattern, song, event, or playlist editors.

When the Patterns Panel has the application focus, and *Sequencer64* is *not* running in **Song** mode, it puts *Sequencer64* in **Live** mode. The musician can control the playback and muting/unmuting of each pattern in the song, while it is playing, from within this window.

If the song editor (see section 5 "Song Editor" on page 77) has the input focus, it controls the muting/unmuting of each pattern, and *Sequencer64* runs in **Song** mode. (There are ways to override this behavior.)

For exposition, we break the Patterns Panel into a menu bar, a top panel, a pattern panel, and a bottom panel. The *Sequencer64* menu bar is discussed in section 2 "Menu" on page 13. The Qt and

Gtkmm user-interfaces differ in the arrangement of buttons and panels; and the Qt interface uses tabs.

## 3.1 Patterns / Top Panel

The top panel of the Pattern window is simple, consisting of the name of the program and a few controls. The latest version adds more buttons.

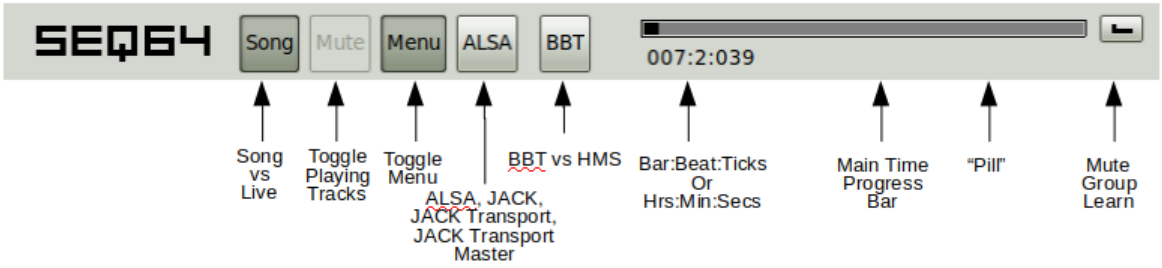



Figure 24: Patterns Panel, New Top Panel Items

This figure shows the appearance of the main window top panel. These items appear in the Gtkmm-2.5 user-interface. The Qt user-interface is arranged differently, but has roughly the same functionality. See Figure 2 "Sequencer64 Main Screen, Qt 5 Interface, With Colors" on page 13.

1. **Song/Live**
2. **Toggle Playing Tracks**
3. **Toggle Menu**
4. **ALSA/JACK Modes**
5. **Song Progress Bar**
6. **Time Display Selection**
7. **Mute Group Learn**

**1. Song/Live.** This new button allows the Song mode to be in force even if the **Song Editor** does not have the focus of the application. However, if the **Song Editor** does have focus, it overrides this button, to preserve expected behavior. There is also a configurable hot-key associated with it, which defaults to `F1`, and is configurable in the "rc" file and the **File / Options / Ext Keys** page.

**2. Toggle Tracks.** This button changes the status of all of the *playing* tracks, reversing the mute status of each pattern that is playing. The next click will then unmute only those tracks. Because it can be confusing, this button is disabled (not shown in the figure) in Song mode. There is also a configurable hot-key associated with it, called **Toggle mutes**, which defaults to `F8`, and is configurable in the "rc" file and the **File / Options / Ext Keys** page.

**3. Toggle Menu.** This button enables and disables the main menu. Disabling it allows additional hot-keys to be configured, without calling forth menu entries by accident. There is also a configurable hot-key associated with this toggle, called **Menu mode**, which defaults to `F3`, and is configurable in the "rc" file and the **File / Options / Ext Keys** page.

**4. ALSA/JACK Modes.** This button sets the ALSA versus JACK modes, including JACK transport and native JACK MIDI playback and recording. When clicked, it brings up the JACK connection page from the **File / Options** dialog. The hot-key for this button is `Ctrl-P`.

**5. Song Progress Bar.** The **Song Progress Bar** is also known as the "main time" bar. Also included is a numeric representation of the "BBT" (bar:beat:ticks). In the Gtkmm-2.4 user-interface,

this bar shows a number of small black cursors ("pills") that show the progress of the song through the various patterns. For short patterns, the progress is fast. For patterns that last longer, the progress is slow. The whole field flashes in time with the beat. This field shows that something is going on. It can also indicate the relative lengths of the various patterns. In the Qt user-interface, this bar is shown as four boxes which blink in time with playback.

Note that the individual pattern boxes in the main panel, for patterns that are not empty, have their own moving progress cursor, a vertical line in each box. By default, this progress line is black, but it can be changed to other colors via a "user" configuration file entry in the `[user-interface-settings]` section. Set the `progress_bar_colored` item to a value ranging from 0 (black) to 6 (dark cyan). There is also a `progress_bar_thick` item to enable a thicker progress bar, for better visibility.

**6. Time Display Selection.** The **Time Display Selection** is also known as the "main time" bar. This button toggles between the states of **BBT** (bar:beats:ticks) and **HMS** (hours:minutes:seconds).

**7. Mute Group Learn.** This button is also known as the "L" button. Click this button, and then press a mute-group key to store the mute-state (whether armed or unarmed) of all the patterns with in that key. In addition to this button, one can also press the `Ctrl-L` key. When in group-learn mode, the `Shift` key cannot be hit, so the group-learn mode automatically converts the keys to their shifted versions. This feature known as *shift-lock* or *auto-shift*.

See the **File / Options / Keyboard** menu entry for the dialog showing the available mute-group keys and the corresponding hot-key for the "L" button. This key is usually the `Insert` key. Unlike the button or the `Ctrl-L` key, this key must be held down while pressing the desired mute-group key. This is a clumsy thing on our main keyboard... one must press and hold `Shift-Fn-Insert` to get the `Insert` key, and then also press the desired mute-group key. A real knuckle-buster! We have our own alternate setup (stored in the file `sequencer64.rc.example`. And we've also added the "shift-lock" feature for this reason.

Group-learn is a modifier key to be pressed just before pressing the desired group key, and the group on/off keys are there to enable each group, *not* to toggle group states.

To set up the mute groups, press the 'L' button, and then press a key on the keyboard to 'learn' or 'save' the preset. Looking at the list of keys assigned for these mute groups (in **File / Options / Keyboard**), the first bank of keys are "!", "", "?", etc., and the second bank are "Q", "W" "E", etc. If the key is valid, then the following prompt occurs:

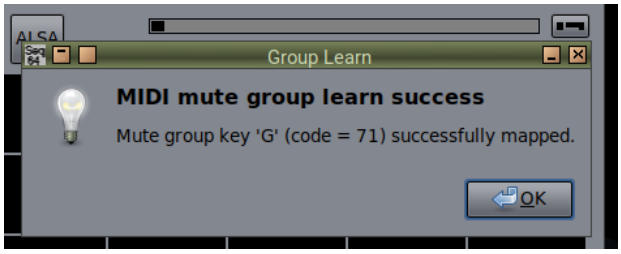



Figure 25: Group Learn Confirmation Prompt

When you ask the program to 'learn' the key, one cannot use the Shift key, so one normally could not use the "!" or other symbol keys. In fact, the process stops as soon as the Shift key is pressed:

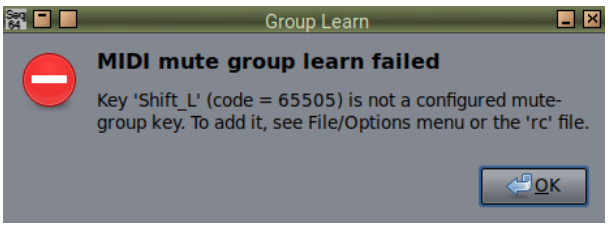



Figure 26: Group Learn Failure Prompt (Shift Key)

To avoid this issue with the `Shift` key, *Sequencer64* now "shift-locks" the keys while group-learn is in progress, so that none of the keys, whether letters or the punctuation characters above the numbers, need the `Shift` key to be held, to learn them. Thus, there is now no need to use the **Caps Lock** is *On* before starting the "learn" process. We tend to map the `Caps Lock` key as the *true* Control key, so that `Caps Lock` is no longer available anyway (it is one of the most useless keys *ever*).

Remember, though, that once learned, the `Shift` is then needed to call up the mute-group that was learned.

There is another way for mute-group learn to fail. With the recent ability of *Sequencer64* to increase the set size via the `--option sets=8x12` (for example), there can be fewer than 32 mute-groups available. This can be seen by viewing **File / Options / Keyboard** when larger set-sizes are in force:

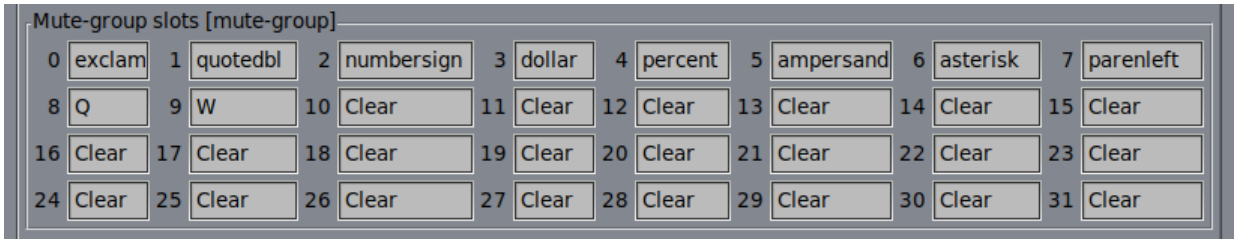



Figure 27: Group Learn Keys With Larger Set Size

Note that only 10 mute-groups (0 to 9) are available with an 8x12 set size. Groups 10 and above are not available, and this is indicated by the word "Clear". Don't worry, though, all of the mute-group key settings still exist, and are still present in the "rc" configuration file in the `[keyboard-group]` section.

But when one attempts a group-learn using one of the unavailable keys, an error occurs, as shown in the following prompt.

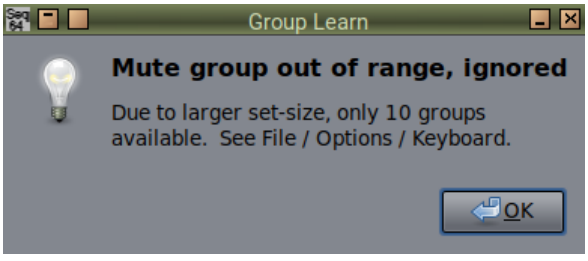



Figure 28: Group Learn Limit Prompt

This prompt also appears if one tries to use the shifted version of an unavailable mute-group key to toggle patterns; this makes it obvious what has gone wrong.

*At some point in the future, we may allow a much larger number of sets, so that all 32 mute-groups are always available. We have to think hard about that enhancement, however.*

One can configure the MIDI settings in similar ways by assigning MIDI commands to arm or toggle loops, using the 'on' option in the "rc" file. See section 10.2 ""rc" File / MIDI Control" on page 103.

Remember that groups work with the playing ("in-view") screen-set. One must change the screenset and give it the command to make it the playing one. By default, the `Home` key is configured for this purpose.

So, for example, if one sets `A` to turn on the patterns in slots 1 and 2 in set 0 (the first set), then pressing `A` in another set will arm the patterns in the same relative location in the current set. This setup is flexible, but takes some thought. One can set up a number of mute-groups, and decide to use them for all sets, or mentally allocate one mute-group per set. Please note that a mute-group key does not *toggle* the saved armed patterns... it can only turn them on.

Let's go through an example using the `Home` key (or whatever key is configured as the **Set Playing Screenset** key.)

1. Load a song with more than one screen-set.
2. Unmute the pattern(s) in the first set and start playback.
3. Use the "`]`" (**Screenset Up**) key to move to the next set. Note that the first set is still playing. Also note that the now-current set is *not* playing.
4. Press the `Home` key. Note that the first set turns off, and the current set turns on. These steps can be repeated at will.
5. Finally, hit the `F8` (**Toggle Mutes**) key. Note that all tracks on all sets toggle muting each time this key is pressed.

## 3.2 Patterns / Main Panel

The main panel of the Patterns window provides a grid of empty boxes, each box delimited by brace-like lines at left and right. Each filled box represents a loop or pattern. One sees only 32 loops at a time in the main panel (but many more than 32 loops can be supported by *Sequencer64*). This group of 32 loops is called a "screen-set", as discussed in section 16.1.17 "Concepts / Terms / screen set" on page 164. One can switch between sets by using the "`[`" and "`]`" keys on the keyboard, or by using the spin-widget-driven, labelled **Set** interface item, or by hitting the (default) `Home` key to make it the playing screenset, or by hitting `Page-Up` or `Page-Down` with the pattern window in keyboard focus. There are a total of 32 sets, for a total of 1024 loops/patterns. Only one screen-set can be controlled at a time, in general. But any number of screensets can be playing at the same time.

Note that the `Page Up` and `Page Down` keystrokes, and their counterparts in **File / Options / Keyboard / Control Keys / Screenset Up** and **Screenset Down**, can be used in lieu of the **Set** spin-button.

It is important to note that keystroke control of the screen-set will wrap-around in screen-set values (i.e. screen-set down at 0 results in screen-set 31, and screen-set up at 31 results in screen-set 0). However, the spinbuttons will stop up at 31 and stop down at 0. We consider this a feature rather than a bug, at this time.

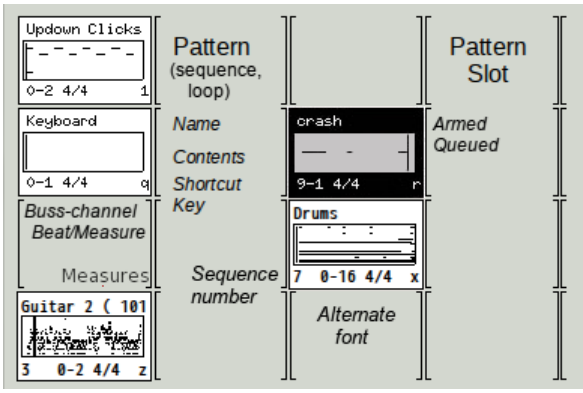



Figure 29: Patterns Panel, Main Panel Items

The individual items annoted in this figure are described in section 3.2.2 "Pattern" on page 44, in more detail. The slot at the bottom left of this figure shows some new features:

- The sequence number appears at the bottom left of the slot.
- The buss number (re 0) and the channel number (re 1) appears to the right of the sequence number, in the format "0-1".
- To the right of that, the time signature ("4/4") appears, at the bottom.
- The hot-key for muting/unmuting the pattern appears next, at the bottom right of the slot.
- The title of the sequence appears at the top left of the pattern slot.
- The length of the sequence, in number of measures (bars), appears at the upper right of the slot.
- Notice the default alternate font, which has a little more body than the *Seq24* font.

Observe that feature in the first figure of the next section. The two main items are the empty *pattern slot*, and the slot filled with a MIDI *pattern*:

1. **Pattern Slot**
2. **Pattern**

### 3.2.1 Pattern Slot

An empty box is a slot for a pattern. If a pattern is present in the slot, the top line will show the title of the pattern, and the number of measures in the pattern. The latter is not shown on some of the figures in this manual, a lack we will ave to rectify someday. Also, these colors are not yet supported in the Qt user-interface.

A pattern can show a number of different statuses based on the coloring of elements in the pattern slot.

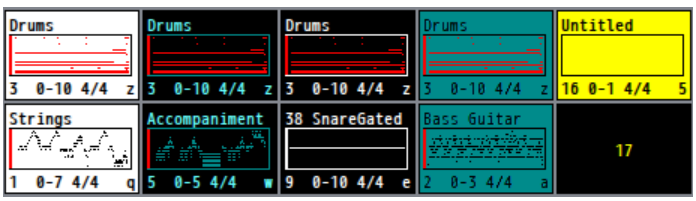



Figure 30: Various Status of Pattern Slots

Not shown in the figure are the gray pattern colors resulting from queuing and one-shot queuing, nor is the number-of-measures number shown. The colors have meaning (in the Gtkmm user-interface only):

- **Empty background**. Whether the classic gray pattern of *Seq24*, or the many patterns of *Sequencer64*, including all black with yellow sequence numbers, this slot coloring indicates that the slot is unused.
- **White background**. Unarmed (muted) patterns show black text on a white background.
- **Black background**. Armed (unmuted) pattern. If the text is yellow, it is a pattern with no MIDI events, but is armed. Note that armed/unmuted patterns can be exported if they have a layout in the Song Editor.
- **Yellow background**. A pattern with no MIDI events, just textual MIDI information. If armed (uselessly), it is yellow text on a black background (not shown).
- **Cyan background, black text**. An unarmed pattern currently being edited in a **Pattern Editor** or event editor. Or, if an SMF 0 MIDI file was just opened or imported, this color combination indicates the SMF 0 format track with all of the data in the song, which only occurs in slot 16 (unless the user then dragged it to another slot).
- **Black background, cyan text**. An armed pattern currently being edited in a **Pattern Editor** or event editor. Or an armed SMF 0 format MIDI sequence.
- **Red events**. Indicates a pattern for which the new transpose feature is disabled. The white, black, and cyan background have the same meanings as in the other items for statuses of unarmed, armed, and currently being edited.

As of version 0.95, the user can also apply coloring to each sequence. This feature was adopted from *Kepler34* ([9]). Here is the new pattern menu for sequence color:

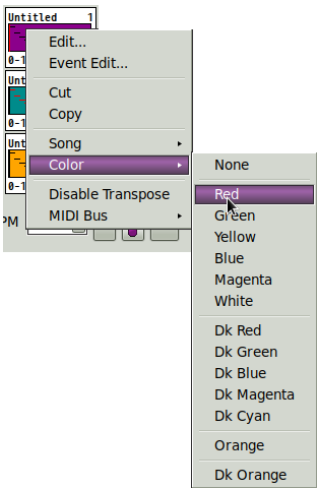



Figure 31: Sequence/Pattern Color Menu

Here is a sample of the coloration as it appears in the patterns panel and in the song editor:

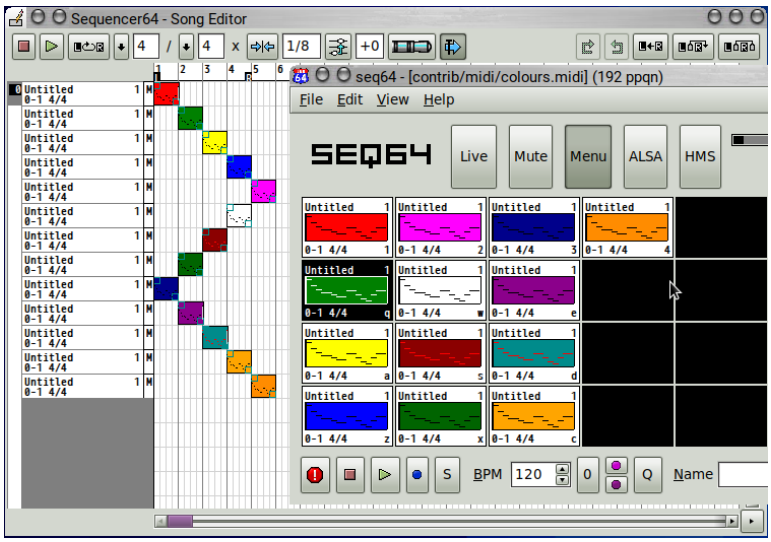



Figure 32: Sequence/Pattern Coloration

Right-click on an empty box one brings up a menu to create a new loop, as well as some other operations:


Figure 33: Empty Pattern, Right-Click Menu

1. **New**
2. **Paste**
3. **Song**
   - Mute All Tracks
   - Unmute All Tracks
   - Toggle All Tracks
   - Toggle Live Tracks

This menu entry is quite different for the Qt user-interface. See section 13.1.1 "Qt 5 Live Slot Menu" on page 147.

**1. New.** Creates a new loop or pattern. Clicking this menu entry fills in the empty box with an untitled pattern, and brings up the Pattern Editor so that one can fill in the new pattern.

In addition to right-click and select **New**, the user can double-click on the empty slot, to bring up a new instance of the sequence editor. For the double-click, the effect can be a bit confusing at first, because it also toggles the arming/mute status of the slot quickly twice (leaving it as it was). It takes some getting used to, but we miss it when using *Seq24*.

A nice feature is hitting the equals ("=") key, then hitting a pattern shortcut key (hot-key), to bring up a new sequence or edit an existing one in a **Pattern Editor** . Another feature is hitting the minus ("-") key, then the hot-key, to bring up the **Event Editor**. The configuration file settings for the the '=' and '-' keys can be altering in the **File / Options / Keyboard** tab.

When an unarmed (muted) pattern is first brough up for sequence editing (or event editing), the slot in the main window is now highlighted (Gtkmm only), using black text on a cyan background, as being the "currently-edited" slot. (This is the same background used to indicate the original track in an SMF 0 to SMF 1 conversion.)

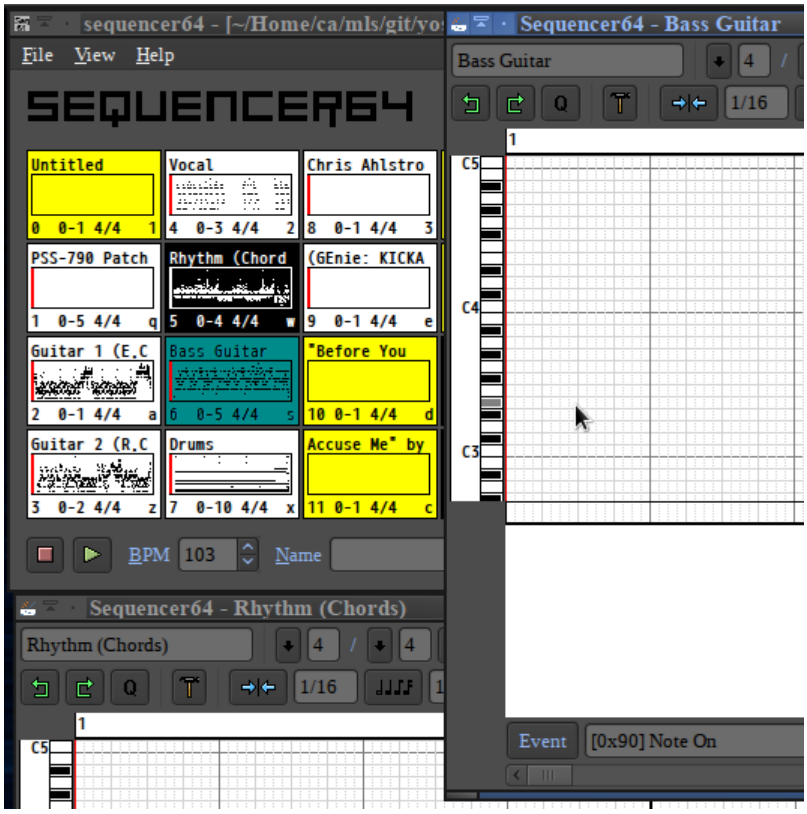



Figure 34: Currently-Edited Pattern, Unarmed

If the currently-edited sequence is armed (unmuted), then the highlighting is reversed (cyan text on a black background), and resembles the highlighting for an armed sequence (which is white text on a black background).

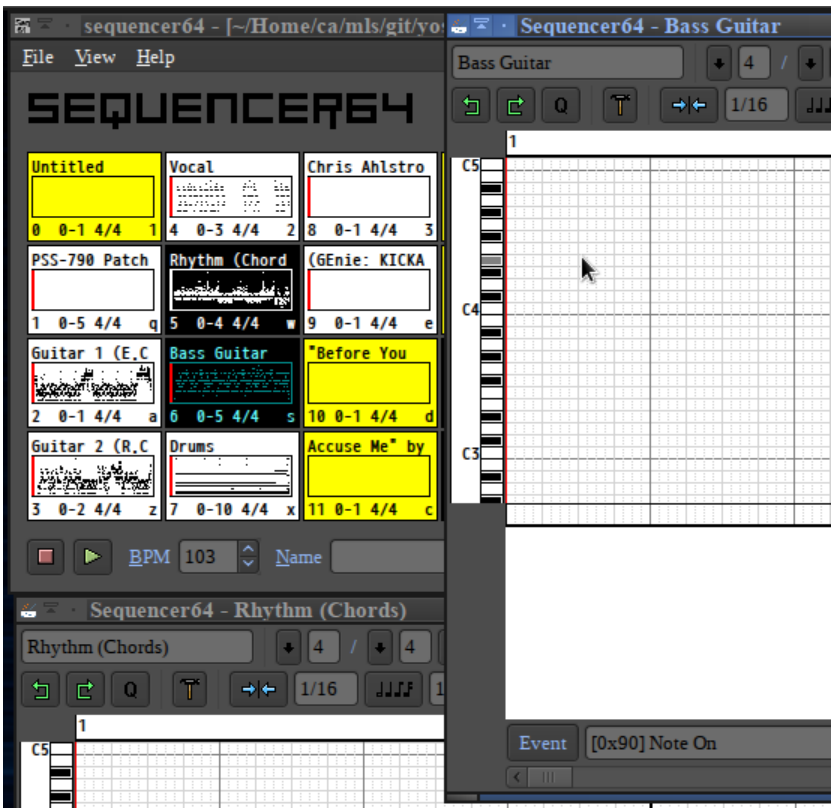



Figure 35: Currently-Edited Pattern, Armed

If more than one sequence or **Event Editor** is brought up, only the slot for the last one to have focus is hightlighted. Note that this highlighting also applies to the **Event Editor**.

**2. Paste.** Pastes a loop or pattern that was previously copied. Also note that there is no `Ctrl-V` key for this operation in the main window.

**3. Song.** The **Song** items are described later, in reference to Figure 38 "Existing Pattern, Right-Click Menu, Song" on page 48.

### 3.2.2 Pattern

A filled pattern slot is referred to as a *pattern* (or *loop*, or *sequence*). A pattern is shown in the Pattern window as a filled box with the following items of information in it. Examine Figure 29 "Patterns Panel, Main Panel Items" on page 40; it shows these items annotated for clarity.

- **Name**. This line contains the name or title of the pattern, to help reference it when juggling a number of patters.
- **Pattern Length**. If the option to show pattern hot-key is enabled, the length of the pattern, in measures, is shown in the upper right corner of the pattern slot. This feature is useful when recording tempo events that will increase the length of the tempo track.
- **Contents**. The contents of the pattern provide a fairly detailed and distinguishable representation of the notes or events in the pattern. Also, when the song is playing, a vertical bar cursor tracks the position of the playback of the pattern or loop; it returns to the beginning of the box every

time that pattern starts over again. With *Sequencer64*, an imported empty pattern will no longer needlessly scroll. However, if a pattern has even a single event (say, a program change), it will scroll.

- **Sequence Number**. If the option to show the sequencer number is set in the **File / Options / Keyboard** section (see section 2.2.8.3 "Menu / File / Options / Keyboard" on page 23, the this number is shown at the bottom left of the pattern slot.
- **Bus-Channel**. This pair of numbers shows the the MIDI buss number, a dash, and the MIDI channel number. For example, "0-2" means MIDI buss 0, channel 2.
- **Beat**. This pair of numbers is the standard time-signature of the pattern, such as "4/4" or "3/4". The first number is the beats-per-measure, and the second is the size of the beat, here, a quarter note.
- **Shortcut Key**. If the display of hot-keys is enabled (see section 2.2.8.3 "Menu / File / Options / Keyboard" on page 23), then the key noted in the lower-right corner of the pattern can be pressed to toggle the mute/unmute status of that pattern. This action is an alternative to left-click on the pattern.
- **Progress Cursor**. At the left of each box is a vertical line, waiting for playback to start so that it can move through the pattern, again and again.
- **Armed**. See Figure 29 "Patterns Panel, Main Panel Items" on page 40; it shows a black and white pattern. The black color indicates that the pattern is armed (unmuted), and will play if playback is initiated in the pattern window in live mode. An item is armed/disarmed by a left-click on it. If the Shift key is held during a left-click on a pattern, then the armed/unarmed state of every other active pattern is toggled. This feature is useful for isolating a single track or pattern.
- **Queued**. That same pattern also shows that it is queued, which means that it will toggle its playing status when the pattern next begins again.
- **Alternate font**. Later builds of *Sequencer64* are now built with a new font. See Figure 29 "Patterns Panel, Main Panel Items" on page 40. It shows the new font. The old font can be selected in the "user" configuration file, and is also selected automatically if *Sequencer64* is run in the *legacy* mode.
- **Sequence number**. Later builds of *Sequencer64* are now built with the option to also show the sequence number in the pattern box, if the "show sequence numbers" option is on. This option can be set in the "user" configuration file. See Figure 29 "Patterns Panel, Main Panel Items" on page 40. It shows an example of the sequence number, using the new font.

Left-click on an filled pattern box will toggle the status of the pattern between muted (white background) and unmuted (black background). If the song is playing via the main window, toggling this status makes the pattern stop playing or start playing. The armed status can also be toggled using hot-keys.

If the **Song Editor** is the active window and was used to start the playback, the pattern boxes will toggle between the muted/unmuted states as the music plays, and the pattern is active or inactive at the point of playback. (The **Song Editor** acts as a list of triggers).

A right-click on an already-filled box brings up a menu to allow one to edit it, or perform a few other actions specified in the context menu. Here is that menu:

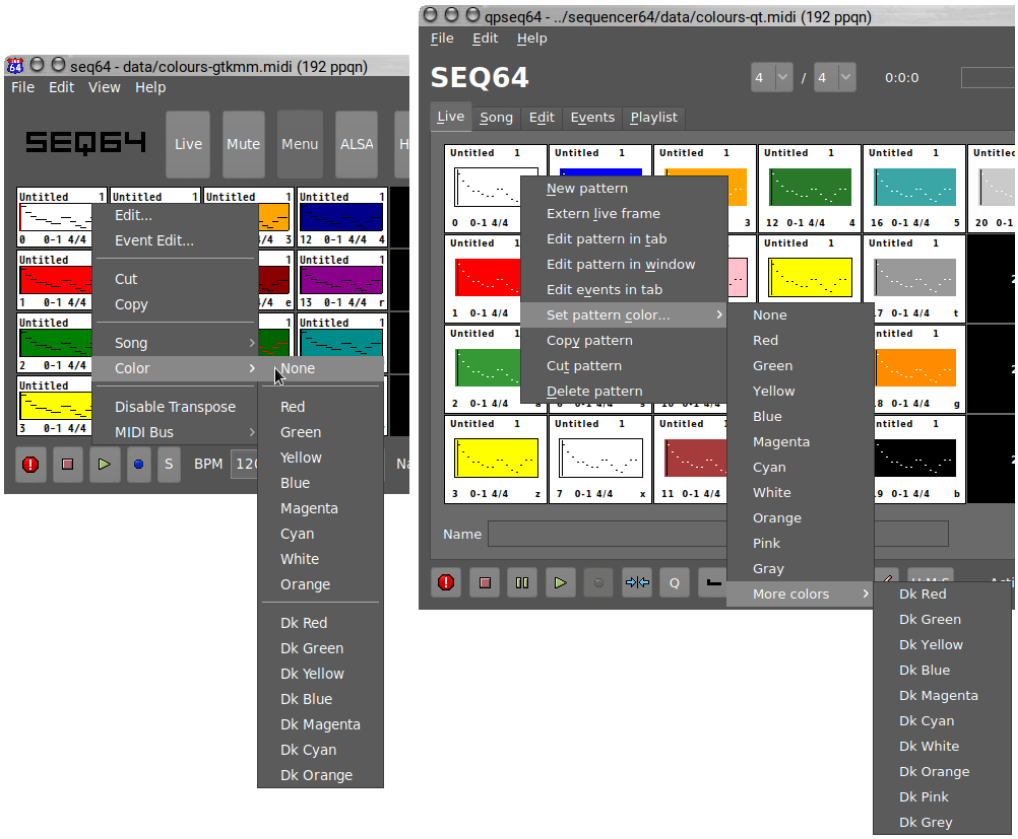



Figure 36: Existing Pattern, Right-Click Menus, Gtkmm and Qt Versions

Here one can choose to edit the pattern, cut and copy the pattern, set the MIDI bus/channel, and more. One can also clear all performance (song) data for the pattern. Here are the Gtkmm menu entries:

1. **Edit...**
2. **Event Edit...**
3. **Cut**
4. **Copy**
5. **Song**
6. **Color**
7. **Disable/Enable Transpose**
8. **MIDI Bus**

The Qt menu entries are different and more extensive:

1. **New pattern**
2. **Extern live frame**
3. **Edit pattern in tab**
4. **Edit pattern in window**
5. **Edit events in tab**
6. **Set pattern color**
7. **Copy pattern**
8. **Cut pattern**
9. **Delete pattern**

See section 13.1.1 "Qt 5 Live Slot Menu" on page 147. It describes these additional items and how the Qt user-interface works. The next sections describe the Gtkmm-accessible functions.

**1. Edit....** Edits an existing loop or pattern. Clicking this menu entry brings up the **Pattern Editor** so that one can modify the existing pattern by click-dragging new notes in a piano roll user-interface. See Figure 48 "Pattern Edit Window" on page 57. Also known as the "sequence editor".

In addition to right-click and selecting **Edit...**, the user can double-click on the slot, to bring up the **Pattern Editor**.

Another way to bring up a pattern in the **Pattern Editor** is to click the **equal** key and then the pattern's hot-key. For example, "**=q**" will open up the editor for the pattern with the hot-key **q**. The Equals key (**=**) is the default key that does this action. This key can be changed by modifying the **File / Options / Keyboard / Control keys / Pattern Edit** item.

**2. Event Edit....** Edits an existing loop or pattern, but using a detailed **Event Editor** that shows events as text and numbers, and allows editing them as text and numbers. See Figure 48 "Pattern Edit Window" on page 57.

Another way to bring up the **Event Editor** is to click the **minus** key and then the pattern's hot-key. For example, "**-q**" will open up the **Event Editor** for that pattern. The Minus key (**-**) is the default key that does this action. This key can be changed by modifying the **File / Options / Keyboard / Control keys / Event Edit** item.

The **Event Editor** is not the same as the **event** pane in the pattern editor; the **Event Editor** shows all events at once, and shows them only in text/list format. This editor is basic, meant for viewing MIDI events and making some minor edits or deletes. The **Event Editor** is most useful when trying to find events that are screwing up the performance of that pattern. See section 6 "Event Editor" on page 85, for more information.

To simplify the application and avoid editing a pattern in two different dialogs, if either the **Pattern Editor** or the **Event Editor** is active for a given sequence, the right-click sequence-slot menu leaves out the **Edit...** and **Event Edit...** menu entries. This trimmed menu looks like this:

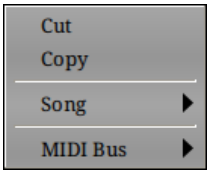



Figure 37: Existing Pattern, Right-Click Menu Without Edit Entries

The old functionality was to have the **Edit...** menu entry simply raise the existing **Pattern Editor** to the top of the windows.

**3. Cut.** Deletes and copies an existing loop or pattern. One can also drag-and-drop a pattern into another cell (there is no outline box during the drag, sadly). Note that there is no `Ctrl-X` key for this operation in the main window.

**4. Copy.** Copies an existing loop or pattern. The pattern can then be pasted elsewhere in the Patterns panel. One can also drag-and-drop a pattern into another cell (there is no outline box during the drag). See section 3.2.1 "Pattern Slot" on page 40. Note that there is no `Ctrl-C` key for this operation in the live (main) window.

**5. Song.** Clicking this menu entry brings up a small popup menu:

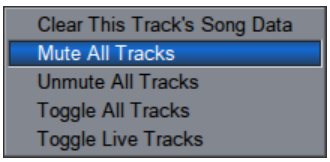



Figure 38: Existing Pattern, Right-Click Menu, Song

1. **Clear This Track's Song Data**
2. **Mute All Tracks**
3. **Unmute All Tracks**
4. **Toggle All Tracks**
5. **Toggle Live Tracks**

**Clear This Track's Song Data** This item is not available if the pattern is empty. Selecting this filled-box right-click menu item causes that box's loop/pattern to be removed from the song editor. The triggers disappear from the Song Editor window, and so will not be played when the song plays in Song mode.

**Song / Mute All Tracks** Selecting this filled-box right-click menu item causes the tracks in the Song Editor to be muted. Sometime it takes a few seconds for the user-interfaces to show this big change. This item mutes all tracks (or loops/patterns). It works when one has opened the Song Editor window and started playing in playback mode by starting play using that window.

So, let us assume the song is running in live (playback) mode. The patterns that are active (unmuted) in the live window are shown with a black background in the main patterns window. If one right clicks on a pattern cell and selects **Song / Mute All Tracks**, all those patterns will become white and be silenced. Eventually, the Song Editor window catches up and shows the "M" activated for all tracks.

**Unmute All Tracks** Provides the opposite functionality, making all tracks armed and audible. Selecting this filled-box right-click menu item causes the tracks in the song to be unmuted.

**Toggle All Tracks** Toggles the armed/mute status of all tracks. It doesn't matter if Live or Song Mode is in force. By default, the `F8` key will also toggle all tracks.

Note that there is also a feature where a `Shift-Left-Click` on a pattern slot toggles the mute status of the *other tracks*.

**Toggle Live Tracks** Toggles the mute status of only the armed/unmuted tracks when in Live mode. Works only in Live mode. This operation unmutes all tracks that are currently unmuted. The statuses of these armed tracks are saved; when this operation is performed again, those tracks are unmuted, turned back on. This menu entry provides the same function as the **Mute** button in the main window.

**6. Color.** This menu item allows for specifying colors for the patterns. Colors can make it easier to find a pattern while running live. Note that there are some minor issues with colors, and that this feature is still in flux.

**7. Enable/Disable Transpose.** This menu entry changes depending upon whether the new transpose feature is enabled or disabled for the sequence/pattern. Note that, if the events shown in the slot are red, this denotes that transpose is currently *disabled* for that pattern, which might be a drum pattern.

**8. MIDI Bus.** Selecting this filled-box right-click menu item brings up a list of the up to 16 MIDI output busses that *Sequencer64* supports. For each of these buss items, another pop-up menu allows one to specify the MIDI output channel for that buss.

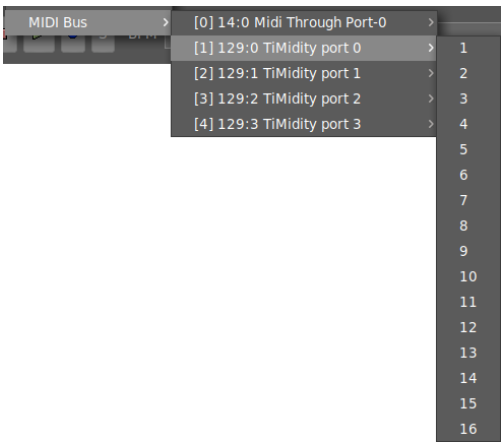



Figure 39: Existing Pattern Right-Click Menu, MIDI Output Busses/Channels

Another way to specify busses is the `--buss n` command-line option. It causes *every* pattern in the MIDI file to be directed to that buss number, and when a new sequence/pattern is created. This option is only for convenience in testing. Save the file, and it will have that buss number as part of each track's data, which makes the song file less portable, so be careful.

### 3.2.3 Pattern Keys and Click

This section recapitulates all the clicks and keys that perform actions in the Pattern windows. Some additional clicks and keys are noted here as well.

#### 3.2.3.1 Pattern Keys

Each pattern in the patterns panel can have a hot-key or shortcut-key associated with it.

For each pattern, hitting its assigned hot-key will also toggle its status between muted/unmuted (armed/unarmed). Below is the default grid that is mapped to the loops/patterns on the screen-set. This grid can be changed in the **File / Options / Keyboard** tab, and is saved in the *keyboard-control* section of the "rc" file.

```
[ 1 ][ 2 ][ 3 ][ 4 ][ 5 ][ 6 ][ 7 ][ 8 ]
[ q ][ w ][ e ][ r ][ t ][ y ][ u ][ i ]
[ a ][ s ][ d ][ f ][ g ][ h ][ j ][ k ]
[ z ][ x ][ c ][ v ][ b ][ n ][ m ][ , ]
```

These characters are shown in the lower right corner of each pattern, as an aid to memory.

A "shift" functionality is available for the mute/unmute hot-keys when a set is larger than 32 patterns. Normally, pressing the `1` key will toggle sequence 0. If preceded by one slash key (`/`), then sequence 32 will be toggled. If preceded by two slash keys, then sequence 64 will be toggled. This features supports using set sizes of 32, 64, and 96 patterns.


Figure 40: Patterns Panel, Shift-Key Pattern Toggle

This figure shows how one presses `y`, `/y`, and `//y` to arm three patterns in this 96-pattern set.

The "`[`" and "`]`" keys on the keyboard decrement or increment the set number.

The left and right `Alt` keys are, by default, set up in the **File / Options / Keyboard / Snapshot 1** and **Snapshot 2** fields to be used as "snapshot" keys. Our preference is to use something that does not trigger desktop commands, perhaps "`F11`" or "`F12`", or one of the keys in the keypad.

When a snapshot key is pressed, the state of the patterns (armed versus unarmed) is saved. While the snapshot key is held, one can then change the state of the patterns (using the keyboard, *not* the mouse) to change how the song plays. When the snapshot key is released, the original saved state of the patterns is restored.

Holding the "queue" key and then hitting a pattern hot-key will queue an on/off toggle for a pattern when the end of the loop is reached. This is the "queue" functionality. This means that the change in state of the pattern will not take hold immediately, but will kick in when the pattern restarts. This pending state is indicated by coloring the central box of the pattern grey, as shown in the figure below. Please note the "keep queue" functionality and the "one-shot queue" functionality described below.

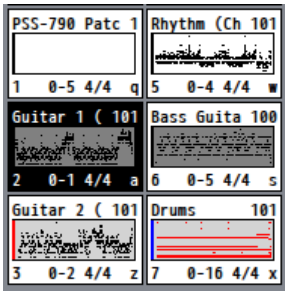



Figure 41: Pattern Coloration when Queued

This figure shows the coloring for queuing in the top two patterns with dark grey event backgrounds. At the end of the pattern, the left top pattern will turn off, and the right top pattern will turn on. The bottom two patterns show the light-grey coloring used to show a "one-shot" queue. The one-shot queue can only turn a pattern on, and it will force the pattern off after one play. Queue also works for mute/unmute pattern sets ("groups"); in this case, every sequence will toggle its status after its individual loop ends.

We do **not** recommend using `Ctrl` or `Alt` keys for pattern control. They conflict with application or desktop settings. However, if one insists on such hot-key combinations, use the **Menu** button in the main window to disable the menu. One can also use normal keys to enable queuing. For example, the minus key or the keypad's slash key can be used.

Pressing the "keep queue" hot-key assigned in the "rc" file activates a "sticky" queue mode. In this mode, pressing a pattern key immediately turns on queuing, instead of mute/unmute. And multiple patterns can be handled in this way at the same time. Keep-queue persists until one clicks the normal queue function hot-key, or changes the active (viewed) screen-set. "Keep queue" mode is cancelled by pressing the normal queue hot-key. This hot-key can be changed in the **File / Options / Keyboard /**

**Keep queue** field. There is also a **Q** button for the same purpose. Also note the "queued replace/solo" functionality, described a bit later.

Thanks to *Kepler34*, we have "one-shot queue" functionality. This one-shot setup queues a pattern up for unmuting only, and, once the pattern has played, it is automatically muted. This process is easier than having to unqueue the pattern manually before the next playback. This hot-key can be changed in the **File / Options / Ext Keys / One-shot queue** field.

The "replace" hot-key (the left `Ctrl` key by default, which should be changed to something better), sets a form of muting/unmuting. When the "replace" hot-key is pressed and held while clicking a pattern or pressing that pattern's hot-key, that sequence is unmuted, and all of the other sequences are muted. "Replace" is a form of "solo". "Replace" is also implemented via MIDI control, where the MIDI control can be activated, but then the user has to select the desired sequence.

*Sequencer64* provides an extension to the replace/solo functionality that is called "queued-replace" or "queued-solo". In this feature, when the "keep queue" function is activated, the replace function is queued so that it does not occur until the next time the patterns loop. And queued-replace provides a form of snapshot, limited to the *current* screen-set. Here are the steps:

1. Start playback with some patterns on.
2. Press and release the "keep queue" hot-key. This puts the application into "queue" mode. It is indicated via a "**Q**" button.
3. Press and hold the "replace" hot-key.
4. Click the desired pattern hot-key. Observe that it arms or stays on, and that the other playing patterns show the "queued" color (grey). At the end of the loop, they turn off, and the "replace" pattern is now solo.
5. Click the same pattern hot-key again. Observe that the other patterns that were toggled off are now queued to be toggled on at the next loop. Steps 4 and 5 can be repeated endlessly.
6. To end the "queued-replace" mode, click the normal "queue" hot-key. Also, changing the active screen-set ends "queue-replace" mode. It does *not* end normal queue mode, to preserve the behavior found in *Seq24*. One needs to clear the queue mode in order to select another pattern to solo.

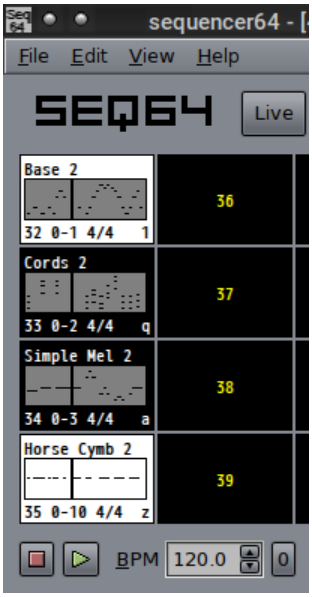



Figure 42: Queued-Replace (Queued-Solo) In Action

Before pressing the "keep queue" key, patterns 33 ("**q**") and 34 ("**a**") are unmuted, while the desired replace pattern, 32 ("**1**") is off. Then the user presses (and holds) the "replace" key, then clicks the "**1**" key. This puts all unmuted patterns, plus the muted replace pattern as well, into queue mode, as shown by the grey panels. When the progress bar reaches the end of the pattern, pattern 32 will go on, and patterns 33 and 34 will go off. If the replace-pattern is already on, it is not queued, as there's no need to turn it on.

If, while in queue mode, the replace key is held and "**1**" is pressed again, the other patterns will be queued, and will turn on again. Thus, the solo status of the replace pattern can be toggled at will, until queue mode is exited by pressing and releasing the normal "queue" key. If the replace key is *not* held down, and another pattern's replace hot-key is pressed, that pattern will be queued normally. If one wants to change the solo functionality to a different pattern, simple hold the replace key and click on a different pattern. The new arrangement of soloing is memorized. One can clear the queue mode by pressing the normal queue key.

There are more keys defined in the **Keyboard** dialog, and it is worth figuring out what they do, if not documented here. For a couple of short, but good, video tutorials about using arming, queuing, and snapshots, see references [35] and [36].

There is a truer "Solo" functionality in the Patterns Panel and the Song Editor. To "solo" a pattern, move the mouse cursor over the pattern, hold the `Shift` key, and left-click the pattern. This will turn off all the other patterns, so that the selected pattern ins the only one playing. Holding the `Shift` key and clicking the same pattern again will unmute all of the other patterns.

#### 3.2.3.2 Pattern Clicks

Left-click on a pattern-filled box will change its state from muted (white background) to playing (black background), whether the sequencer is playing or not.

Left-click-hold-drag on a pattern, drags it to a different pattern on the grid. The box disappears while dragged, and reappears in the new location when dropped. However, a pattern *cannot* be dragged if its **Pattern Editor** window is open.

Right-click on a pattern brings up the appropriate context menus, as discussed earlier, depending on whether the pattern box is empty or filled.

Middle-click does nothing when the mouse rests inside a pattern box.

### 3.3 Patterns / Bottom Panel

The bottom panel of the Patterns window provides way to control the overall playback of the song. It has changed quite a bit over the last few versions of *Sequencer64*, and we have not yet caught up with the diagrams. And the Qt user-interface adds more changes. Refer to the diagram of the whole window, for now.

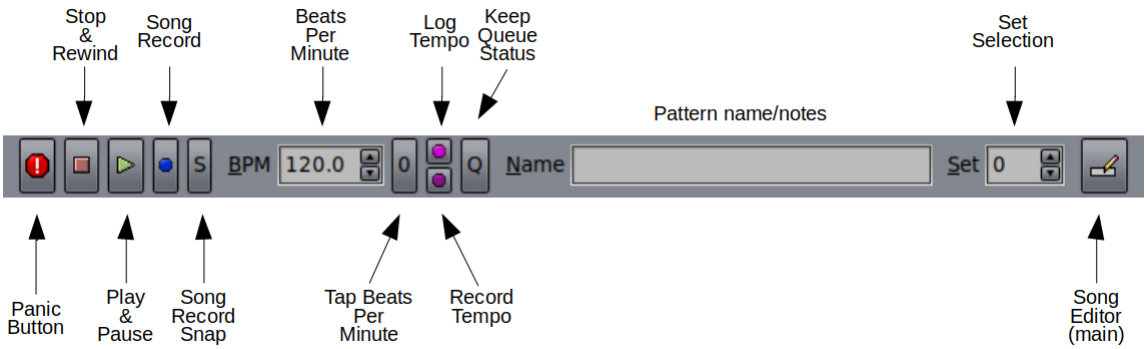



Figure 43: Patterns Panel, Bottom Panel Items

This figure shows a number of new items.

1. **Panic!**
2. **Stop**
3. **Play and Pause**
4. **Song Record**
5. **Song Record Snap**
6. **BPM**
7. **Tap Tempo**
8. **Log Tempo**
9. **Record Tempo**
10. **Keep-Queue Status**
11. **Name**
12. **Set**
13. **Toggle Song Editor**

**1. Panic!.** This new button stops the song and sends MIDI Off messages on all notes.

**2. Stop.** The red square button stops the playback of the song and all its patterns. The keystroke for stopping playback is the `Escape` character. It can be changed to `Space`, so that the space-bar then becomes effectively a playback toggle key.

**3. Play and Pause.** The green triangular button starts the playback of the whole song. The keystroke for starting playback is the `Space` character by default.

The Play button can be used as a Pause button. When the Play button is clicked, the button icon changes to a Pause icon:

Figure 44: Patterns Panel, Pause Button

A Pause key (by default, the period) is also defined.

**4. Song Record.** Song-recording in *Sequencer64* is adopted from the *Kepler34* project. This feature takes live muting changes and records them as triggers in the **Song Editor**. The default hot-key for this function is `P`. This feature does not honor queuing... rather than waiting until the end of the pattern when the queuing takes effect, the trigger recording starts immediately.

**5. Song Record Snap.** This button toggles snapping the beginning and end of a recorded trigger to the nearest beat. There is no hot-key for this button at this time.

**6. BPM.** The spin widget adjusts the "beats per minute" (BPM) value. The range of this field is from 1 bpm to 600 bpm, with a default value of 120 bpm. Although this field looks editable, it is not. Most keystrokes that are entered actually toggle one of the pattern boxes. However, the following keys can also modify the BPM in small increments: The `semicolon` reduces the BPM; The `apostrophe` increases the BPM. Also, if one right-clicks on the Up button, the BPM advances to its largest supported value, and if one right-clicks on the Down button, the BPM advances to its lowest value. MIDI control for this value is also available.

The precision of the BPM value can be set to 0, 1, or 2 decimal places, and the increment values for the step size (small) or page size (large) of the BPM spinner can be configured in the "usr" file. See section 11.4 ""usr" File / User MIDI Settings" on page 137; it describes the precision and increment options more fully. The following figure shows the appearance of the BPM field with different precision values:

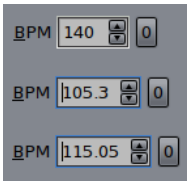



Figure 45: Patterns Panel, BPM Precisions

**7. Tap Tempo.** This control is clicked in time with a tune, to set the tempo based on the tempo of the clicks. Once clicked, the label of this button increments with every click, and the **BPM** field updates to display the calculated tempo. If the user stops tapping for 5 seconds, the label reverts to 0, the BPM value keeps its final value, and the user can try tapping the tempo again, or accept the current value. Tapping can also be done using the keystroke defined in **File / Options / Ext Keys / Tap BPM**. It defaults to the "`F9`" key.

**8. Log Tempo.** This light-magenta button (Gtkmm only), logs the current tempo at the current playback spot as a Set Tempo meta-event, in the first track (pattern slot #0) or the alternate tempo track if defined, and only if it is active. According to the MIDI standard, such events should be present *only* in the first track, and so *Sequencer64* follows this rule, and also makes tempo events officially supported. They can be edited in the **Pattern Editor** or in the **Event Editor**. The main/global tempo is not written to the tempo track; it is stored in a SeqSpec section. See section 9 "Sequencer64 Meta Event / SysEx Support" on page 99, for more information.

**9. Record Tempo.** This dark-magenta button (Gtkmm only) becomes light-magenta when activated, and turns on the recording of any tempo changes made in the BPM spinner. If the spinner is held down, a ramping stream of tempo events is created. If the time exceeds the current length of the tempo track, then the length of the track is automatically increased. These tempo events will not affect playback speed unless the tempo track is unmuted.

**10. Keep-Queue Status.** This item is the **Q** button. It provides a visual way to know the current state of keep-queue, and is activated either by clicking on it or by pressing the assigned keep-queue key.

**11. Name.** Each of the 32 available screen-sets can be given a name by entering it into this field. This name is saved with the MIDI file.

**12. Set.** This spin widget selects the current screen-set. The values in this field range from 0 to 31 (less if the set-size is a larger value), and default to 0.

**13. Toggle Song Editor.** Pressing this button toggles the presence on-screen of the **Song Editor**. The `Ctrl-E` keystroke can also be used.

## 3.4 Patterns / Multiple Panels

*Sequencer64* has a "multiwid" option, which allows for multiple live panels showing more than one set at a time. The default is to show the usual single "mainwid". In the Qt user-interface, multiple live panels in separate windows can be opened.

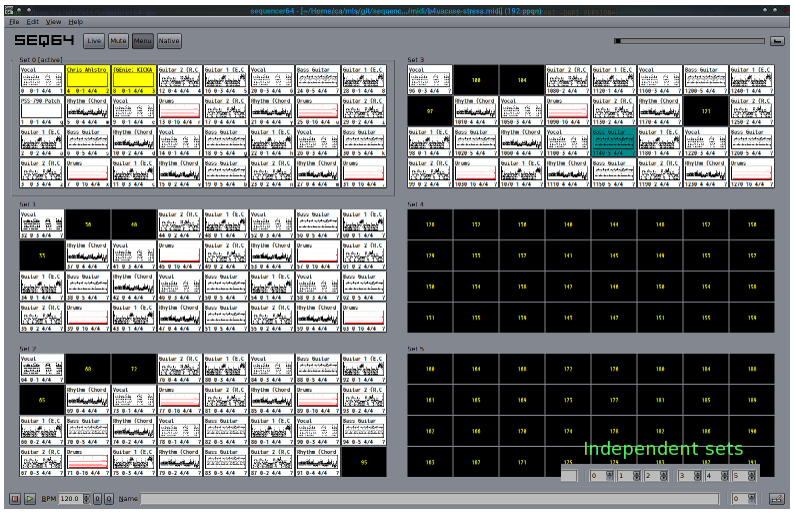



Figure 46: Patterns Panel, with Multiple Panels

In the Gtkmm user-interface, either one spinner for all, or one spinner for each, is available. The latter is the "independent" mode. Note that it is possible, in this mode, to show the same set in two different "mainwids", but this is not recommended, as there may be minor unavoidable issues with that.

Note that Page Up and Page Down keystrokes, as well as their configurable counterparts in **File / Options / Keyboard / Control Keys / Screenset Up** and **Screenset Down**, apply only to the top leftmost "mainwid". Of course, if the "mainwids" are synced, then all are affected by these keystrokes.

In multi-wid mode, each "mainwid" frame shows the corresponding set number and, if present, the set notepad text for each "mainwid" set.

See section 14 "Sequencer64 Man Page" on page 155, for how the `-o wid=3x2,i` option can be used to set this mode, and section 11.3 ""usr" File / User Interface Settings" on page 132, for how these settings can be made permanent in the "usr" file. In that file, the options modified are `block_rows` and `block_columns`.

## 3.5 Patterns / Variable Set Size

This option, informally known as "variset", allow some changes in the set size and layout from the default 4x8 = 32 sets layout. The row count can be set from 4 to 8, and the column count can be set to 8 to 12. Note that the set size can only be *increased* by these settings.

**Warning:** *seq24* was fairly hardwired for supporting 32 patterns per set, and there are still places where that is true. Thus, consider this option to be experimental.

Also see section 14 "Sequencer64 Man Page" on page 155, for how the `-o sets=8x8` option can be used to set this mode, and section 11.3 ""usr" File / User Interface Settings" on page 132, for how these settings can be made permanent in the "usr" file. In that file, the options modified are `mainwnd_rows` and `mainwnd_cols`.

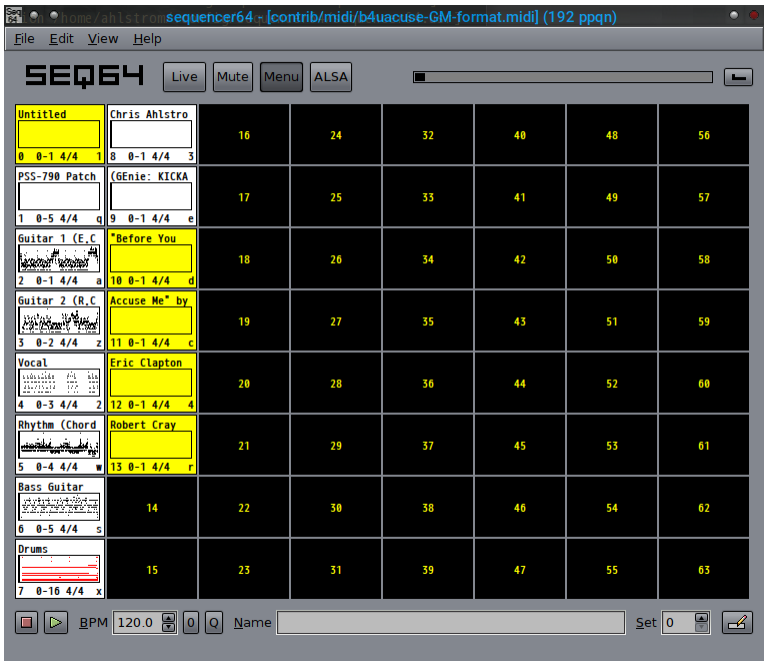



Figure 47: Patterns Panel, 8 x 8 Layout

Generally, it is recommend to stick with the 4x8 (32 patterns/set), 8x8 (64 patterns/set), and 8x12 (96 patterns/set). This works best with the existing set of 32 hot-keys.

Also note that the Qt 5 user-interface also supports "variset", whether in the main window or in the external live-frame. In addition, both Qt windows can be resized and still show good renditions of the pattern-slots.

# 4 Pattern Editor

The *Sequencer64 Pattern Editor* is used to edit and preview a pattern, as well as to configure its buss and channel settings. Please note that the **Qt 5** user-interface (which will eventually supercede the Gtkmm-2.4 user-interface) provides an **Edit Tab**, which has limited functionality, and an external window interface, which is on par with the Gtkmm-2.4 version.

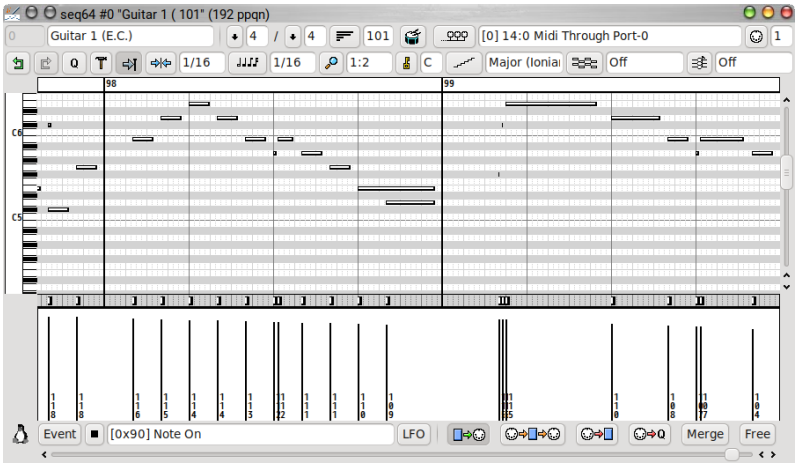



Figure 48: Pattern Edit Window

This dialog is complex. For exposition, we break it into some common actions, a first panel, a second panel, a bottom panel, and a piano-roll/events section.

1. **First Panel**
2. **Second Panel**
3. **Piano-Roll/Events Panel**
4. **Bottom Panel**
5. **Common Actions**

Before we describe this window, there are some things to recognize. First, if the pattern is empty when play is started, the progress bar will still move, that the user can key in new notes. Second, to add a note, one must press the *right* mouse button (the pointer changes to a pencil) and, *while holding it*, press the left mouse button. Or click in the pattern editor, press the `p` key to select the "pencil" or "paint" mode, then left-click to add a note or left-click-drag to add multiple notes as the mouse moves. Press or release the right mouse button, or press `x` to "eXit" or "eXscape" from paint mode. Third, notes are drawn only with the length selected by the "notes" button near the top of the pattern window. There are tricks to modifying the new notes that are described later.

*Sequencer64* automatically scrolls horizontally through the sequence/pattern editor window when playback moves the progress bar outside of the current frame of data. This feature makes it easier to follow patterns that are longer than a measure or two.

*Sequencer64* also provides a way to restart the progress bar within the pattern without resetting it to the beginning of the pattern. This "pause" feature is accessed by the pause key (which defaults to the period character) or by the pause button, which appears when playback is underway.

## 4.1 Pattern Editor / First Panel

The top bar (horizontal panel) of the Pattern (sequence) Editor lets one change the name of the pattern, the time signature of the piece, how long the loop is, and some other configuration items.

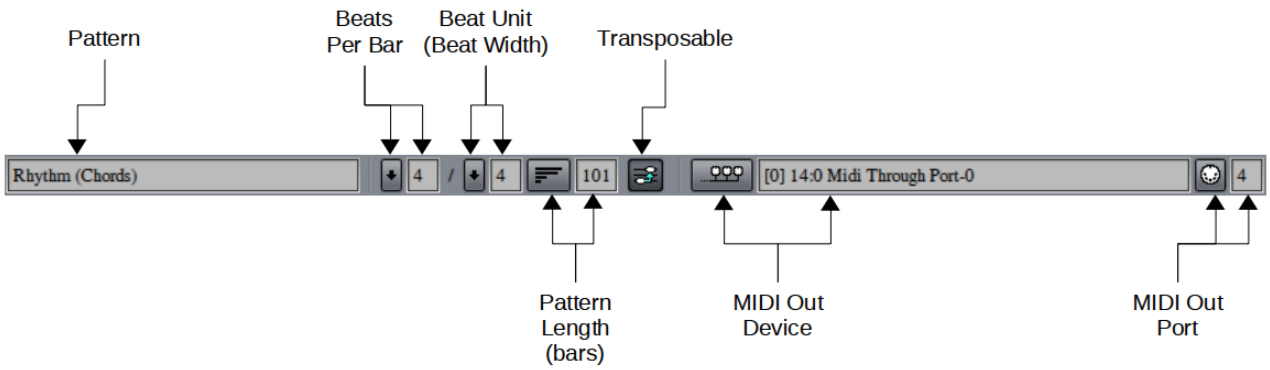



Figure 49: Pattern Editor, First Panel Items

In recent versions of *Sequencer64*, a "sequence-is-transposable" feature has been added, as shown above.

1. **Pattern Number** (not shown)
2. **Pattern Name**
3. **Beats Per Bar**
4. **Beat Unit (Beat Width)**
5. **Pattern Length**
6. **Transposable (toggle)**
7. **MIDI Out Device**
8. **MIDI Out Port**

**1. Pattern Number.** This item shows the sequence/track/pattern/loop number, to make it easier to pick it out when a lot of patterns are being edited at once.

**2. Pattern Name.** Provides the name of the pattern. This name should be short and memorable. It is displayed in the Patterns window, on the top line of its pattern box/slot. There is no edit cursor in this field (not sure why yet), so one has to edit blind.

**3. Beats Per Bar.** Specifies the number of beat units per bar in the time signature. The possible values range from 1 to 16, if the drop-down menu is used. The numeric value can be directly edited, to achieve non-standard values, thanks to an update from Jean-Emmanuel. But be careful!

**4. Beat Unit (Beat Width).** Specifies the size of the bottom beat unit of the time signature: 1 for whole notes; 2 for half notes; 4 for quarter notes; 8 for eight notes; 16 for sixteenth notes; and 32 for thirty-second notes. The whole time signature is display at the bottom center of a pattern slot.

**5. Pattern Length.** Sets the length of the current pattern, in measures. The possible values range from 1 to 64. *However*, when opening or importing a non-*Sequencer64* MIDI tune, the length of each track will be used, and so other values are possible; they just cannot be set via the user-interface. The numeric value can be directly edited, to achieve non-standard values, thanks again to Jean-Emmanuel.

Bringing up a short pattern (one less than one measure or bar in length) in the pattern editor will adjust the pattern to pad it to the length of one measure. *Sequencer64* will, when it reads such a short pattern from a MIDI file, pre-pad it to the length of a measure, so that it will always show smooth progress. For example, a pattern containing just one program change will be padded to the size of a full measure.

It would be nice to have a value that represents "indefinite", so that the loop or pattern would be more like a track, and not be repeatable. We have part of that functionality: a feature from user Stazed

allows the pattern to expand indefinitely while the user inputs MIDI from a controller. See section 4.4 "Pattern Editor / Bottom Panel" on page 73below. (Also nice would be a "one-shot" pattern, useful for live intro patterns, for example.)

**6. Transpose Toggle.** This setting allows the sequence to be transposed by the global transpose selection made in the song editor. If transpose is enabled for that pattern, the button will be highlighted as per the current desktop theme. Patterns for drums should, in general, not be transposable.

**7. MIDI Out Device (Buss).** This setting specifies one of the 16 MIDI output busses provided by *Sequencer64*, or one of the MIDI devices set up in the computer. The settings look a lot like Figure 39 "Existing Pattern Right-Click Menu, MIDI Output Busses/Channels" on page 49.

**8. MIDI Out Port (Channel).** This settings select the MIDI output channel, or port. The possible values range from 1 to 16. If instruments are defined in the "user" configuration file to that device and channel, their names will be shown.

## 4.2 Pattern Editor / Second Panel

The second horizontal panel of the Pattern Editor provides a number of additional settings.

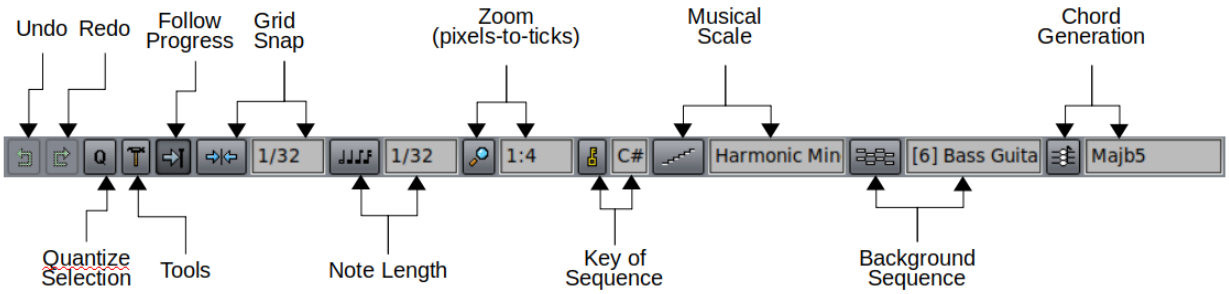



Figure 50: Pattern Editor, Second Panel Items

There is a new button to the right of the **Tools** ("hammer" button). This button toggles the "follow progress" feature; it is the active button in the diagram above.

*Sequencer64* includes a chord-generation option.

1. **Undo**
2. **Redo**
3. **Quantize Selection**
4. **Tools**
5. **Follow Progress** (new)
6. **Grid Snap**
7. **Note Length**
8. **Zoom**
9. **Key of Sequence**
10. **Musical Scale**
11. **Background Sequence**
12. **Chord Generation**

**1. Undo.** The Undo button rolls back any changes to the pattern from this session. It will roll back one change each time pressed. It is not certain what the undo limit (if any) is, however. Pressing `Ctrl-Z` is the same as using the **Undo** button.

**2. Redo.** The Redo button will restore any undone changes to the pattern from this session. It will restore one change each time it is pressed. It is not certain what the redo limit is, however.

**3. Quantize Selection.** This button quantizes the selected events as per the **Grid Snap** setting.

**4. Tools.** This button brings up a nested menu of tools for modifying selected events and notes.

**5. Follow Progress.** This button toggles whether or not the progress bar follows progress in long patterns. Turning off this feature is useful when one wants to concentrate on the current measure without the paging to subsequent measures that occurs with the "follow progess" feature.


Figure 51: Tools, Context Menu

1. **Select**
2. **Modify Time**
3. **Modify Pitch**

**Select** provides two sets of selections for notes:

- **All Notes**, which selects all notes in the pattern; Note that `Ctrl-A` will also select all of the events in the pattern editor.
- **Inverse Notes**, which inverts the selection of notes.

Other event-selection actions are provided using the mouse in the piano roll:

- **Left Click**. Pressing the left button on a note or a event deselects all other notes or events, and selects the item clicked on.
- **Ctrl Left Click**. Pressing the `Ctrl` key and the left button on a note or an unselected event *adds* that event to the selection.
- **Left Click Drag**. Pressing the left mouse button and dragging also lets one select ("lasso") multiple events and notes.
- **Ctrl Left Click Drag**.
  - Pressing the `Ctrl` while left-click-dragging *on unselected events* lets one make additional selections of multiple events and notes.
  - Pressing the `Ctrl` while left-click-dragging *on an already-selected event* lets one stretch or compress the lengths of multiple notes in the selection.

There are many things that can be done with selected notes:

• **Modify Event Data** offers a way to alter the event data values in the lower pane of the pattern editor, the "data pane". By left-dragging the mouse in the data pane across the value lines that are shown, the values are chopped or set to the height of the mouse pointer at each event. When notes are selected, and the mouse is used to change the values (heights) of the lines in the event-data area, *only the events that are selected* are changed. The data-values of *unselected* events are left unchanged. A cool feature from *Seq24*.

• **Modify Time** offers two ways to tweak the timing of the selected note: **Quantize Selected Notes**, which quantizes the selected notes, the same way as the **Quantize** ("**Q**") button; **Tighten Selected Notes**, which is merely a less strict form of quantization.

• **Modify Pitch** has only one entry by default, **Transpose Selected** (not shown). Selecting the **Transpose Selected** entry brings up the following sub-menu:

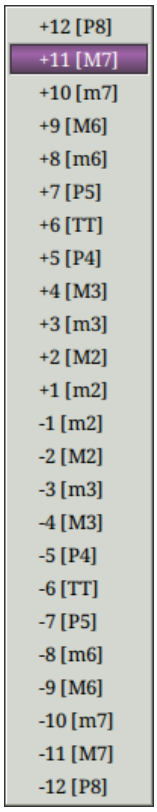



Figure 52: Tools, Transpose Selected Values

• If the user has selected a **Musical Scale** setting other than **Off**, then **Modify Pitch** has two entries: **Transpose Selected**, discussed above, plus another sub-menu, **Harmonic Transpose Selected**, which makes sure that all transpositions stay on the selected scale.

Note that one can also modify the pitch of selected notes by the left-click-drag action, or by moving the selection using the up-arrow or down-arrow keys.

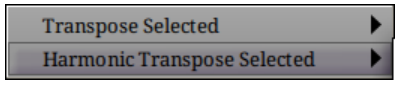



Figure 53: Tools, Two "Transpose" Menus

Remember that only the **Transpose Selected** entry is shown if the **Musical Scale** setting is **Off**. Selecting the **Harmonic Transpose Selected** entry brings up the following sub-menu:

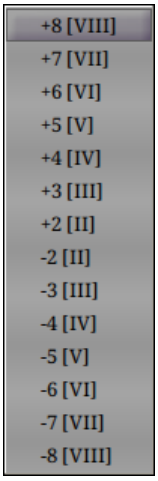



Figure 54: Tools, Harmonic Transpose Selected Values

Again, the harmonic-transpose option will not be available unless a scale has been selected.

**6. Grid Snap.** Grid snap selects where the notes will be drawn. The following values are supported: 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, and 1/128. Additional values are also supported: 1/3, 1/6, 1/12/, 1/24, 1/48, 1/96, and 1/192.

**7. Note Length.** Note length determines the duration of the inserted notes. Like the **Grid Snap** values, the following values are supported: 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, and 1/128. Additional values are also supported: 1/3, 1/6, 1/12/, 1/24, 1/48, 1/96, and 1/192.

**8. Zoom.** Horizontal zoom is the ratio between MIDI pixels and ticks, written as "pixels:ticks", where "ticks" is the "pulses" in "PPQN". For example, 1:4 = 4 ticks per pixel. Supported values are 1:1, 1:2, 1:4, 1:8, 1:16, and 1:32, along with more new values to support higher PPQN tunes: 1:64, 1:128, 1:256, and 1:512. The default zoom is 2 for the standard PPQN value, 192, but it increases for higher PPQN values, so that the zoom looks sensible. As the right number (ticks) goes higher, the effect is to zoom out, and show more of the pattern. In fact, it might be better to list them as ticks (pulses) per pixel. Legacy stuff.

After a left-click in the piano roll, the **z**, **Z**, and **0** can be used to zoom the piano-roll view horizontally. The **z** key zooms out (smaller), the **Z** key zooms in (larger), and the **0** key resets the zoom to the default value. The horizontal zoom feature also affects the time-line (measures indicator) and the data area.

(Note that the Song Editor, which now has zoom functionality, has a default resolution of 32 pulses per pixel, so, by default, it has 16 times the resolution of the Pattern Editor.)

**9. Key of Sequence.** Selects the desired musical key for the pattern. The following keys are supported: C, C#, D, D#, E, F, F#, G, G#, A, A#, and B. Changing the **Key** will also shifts the marked note-rows for the **Musical Scale** setting.

The musical key that a sequence/pattern is set to is saved in the MIDI file along with the rest of the data for the sequence. **However**, it turns out that a change made to the key, scale, or background sequence in the pattern editor is saved in that editor, so that opening another sequence will apply the same settings to that sequence. This is an optional feature, now more rigorously supported, as noted below.

If the global-sequence feature is enabled, and the user selects a different key, scale, or background sequence in the pattern editor, then *all* patterns share the selected key, scale, or background sequence.

Furthermore, these settings are saved in the "proprietary" section of the MIDI file, where they are available for all patterns.

If the global-sequence feature is *not* enabled, and the user selects a different key, scale, or background sequence in the pattern editor, then only that pattern will use the selected key, scale, or background. The key, scale, or background sequence change will be saved in the MIDI file only for that pattern, as a SeqSpec meta event. The global-sequence feature setting can be made in the "user" configuration file.

**10. Musical Scale.** Selects the desired scale for the pattern. When a scale is selected, the following features are supported:

- The notes that are *not* in the scale are shown as grey in the piano roll, to make it easier to key all notes in-scale.
- For harmonic transposition, the notes are shifted so that they remain in the selected scale.
- The exact notes that are considered "in-scale" depend also on the exact value of the **Key of Sequence** setting.

The following musical scales are supported:

- **Off (Chromatic)**
- **Major (Ionian)**
- **Minor (Aeolian)**
- **Harmonic Minor**
- **Melodic Minor**
- **Whole Tone**
- **Blues**
- **Major Pentatonic**
- **Minor Pentatonic**

Please let us know of any mistakes in the new scales. Please note that the **Melodic Minor** scale is supposed to descend in the same was as the natural **Minor** scale, but there is no way to support that trick in *Sequencer64*.

One can select which **Musical Scale** and **Key** the piece is in, and *Sequencer64* will grey those keys on the piano-roll that are *not* in the selected scale for the selected key. This is purely visual; a user can still add off-key notes. This effect is shown for the C Major scale in the following figure:

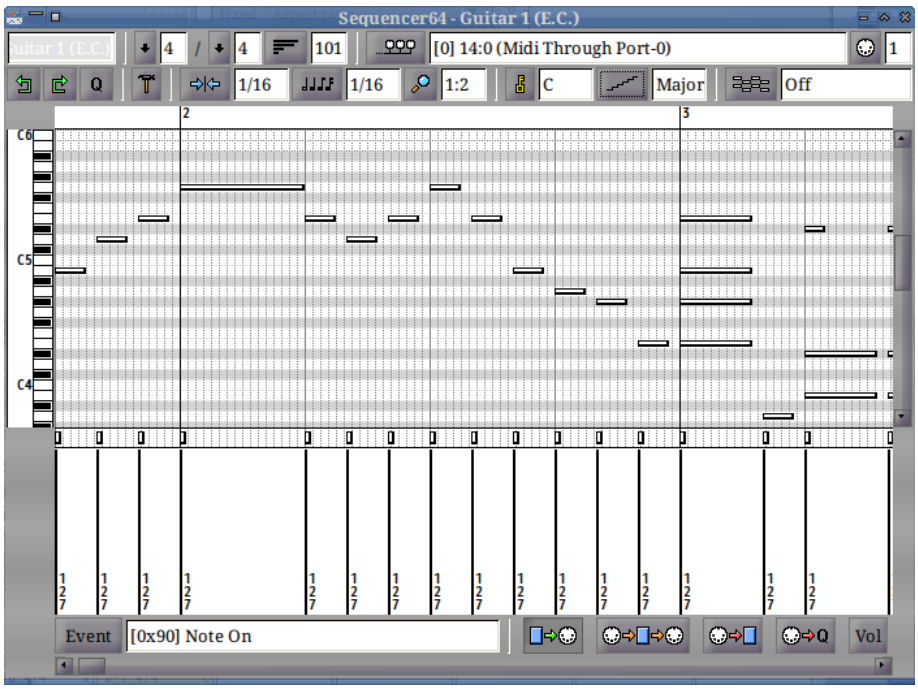



Figure 55: C Major Scale Masking

This feature makes it easier to stay in key while playing and recording. Note that the scale will shift when a different **Key** is selected.

The scale that a pattern is set to is now saved in the MIDI file along with the rest of the data for the pattern. **However**, a change made to the key, scale, or background pattern in the pattern editor is saved in the editor, so that opening another pattern will apply the same settings to that pattern. This is a feature. The feature had some quirks, which are fixed, and it is now optional. The user has the option of applying the key/scale/background-sequence either globally (all patterns) or locally, per-pattern, with each pattern holding its key, scale, and background-sequence settings in SeqSpec meta events.

**11. Background Sequence.** One can select another pattern to draw on the background to help with writing corresponding parts. The button brings up a small menu with values of **Off** and [**0**]. The 0 is a set number. Sets are numbered from 0 to 31. Additional set numbers appear in the menu for each set that has data in it. Under the **0** entry, a menu like the following appears:

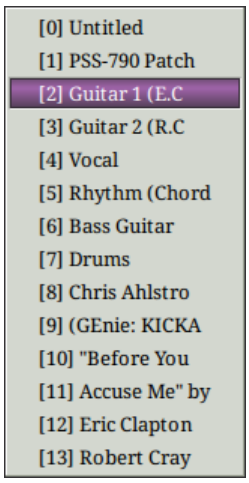



Figure 56: Sample Background Sequence Values

Once the desired pattern is selected from that list, it appears as dark cyan note bars, along with the normal notes that are part of the pattern. (Also note the orange selected notes and events in the following figure.)

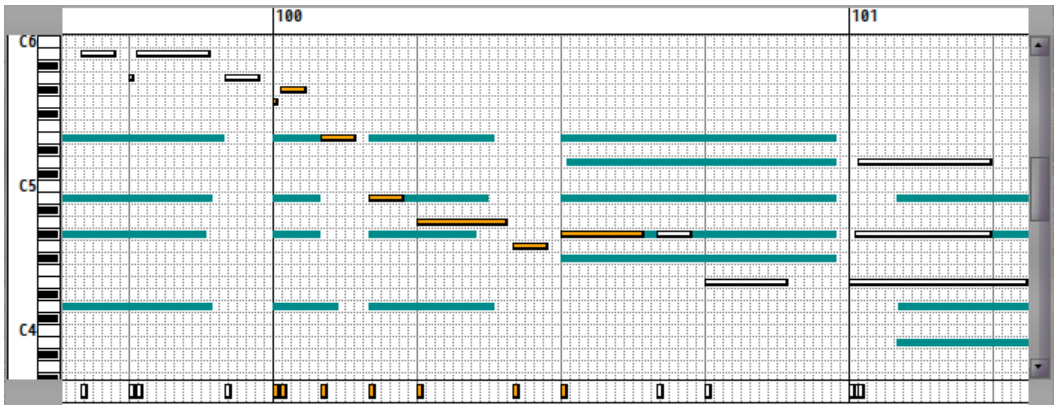



Figure 57: Background Sequence Notes

The dark cyan notes shown represent the rhythm pattern that was selected as the background pattern.

The background sequence that shows is saved in the MIDI file along with the rest of the data for the sequence/pattern. A change made to the key, scale, or background sequence in the pattern editor is saved in the editor, so that opening another sequence will apply the same settings to that sequence. This is an optional feature, now supported, as noted earlier.

**12. Chord Generation.** The ability to insert chords with one click has been added. This feature comes from user "stazed" and his *Seq32* project ([26]).

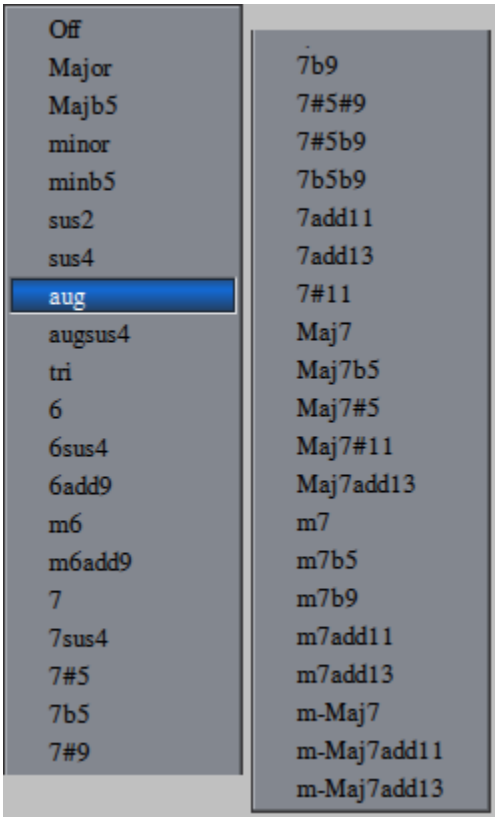



Figure 58: Chord Generation Menu

The figure shows the menu broken into two pieces. Once a value other than **Off** is selected, a left-click in drawing mode will add multiple notes representing the chord created, with the clicked note value as the base of the chord.

## 4.3 Pattern Editor / Piano Roll

The piano roll is the center of the pattern/loop/track/sequence editor. It is accompanied by a thin "event bar" ("event area", "event strip") just below it, and a taller "data bar" or "data area" just below that. While the pattern editor is very similar to note editors in other sequencers, it is a bit different in feel. A good mouse with 3 or more buttons is practically a necessity for editing (though *Sequencer64* is usable with some crummy trackpads common on modern laptops, and with keystrokes.) We tend to like the Logitech Marble Mouse, an ambidextrous USB trackball. It has four buttons, and we use the `contrib/scripts/marblemouse` script to set up the left small button as a middle button. The script merely makes the following call:

```
xmodmap -e "pointer = 1 8 3 4 9 6 7 2 5 10 11"
```

One can page vertically in the piano roll using the Page Up and Page Down keys. One can go to the top using the Home key, and to the bottom using the End key. One can page left and right in the piano roll using the Shift Page Up and Shift Page Down keys. One can go to the leftmost position using the

Home key, and to the rightmost position using the End key, There are more keystrokes to be described later.

Do not forget the note-step option. If one paints notes with the mouse, the note position advances with each click. If one paints notes via an external MIDI keyboard, the notes are painted and advanced, but not previewed. To preview them, click the **pass MIDI in to output** button to activate so that they will be passed to the sound generator or software synthesizer.

### 4.3.1 Pattern Editor / Piano Roll Items

The center of the pattern editor consists of a time panel at the top, a virtual keyboard at the left, a note grid, a vertical scrollbar, an event panel, and a data panel at the bottom.

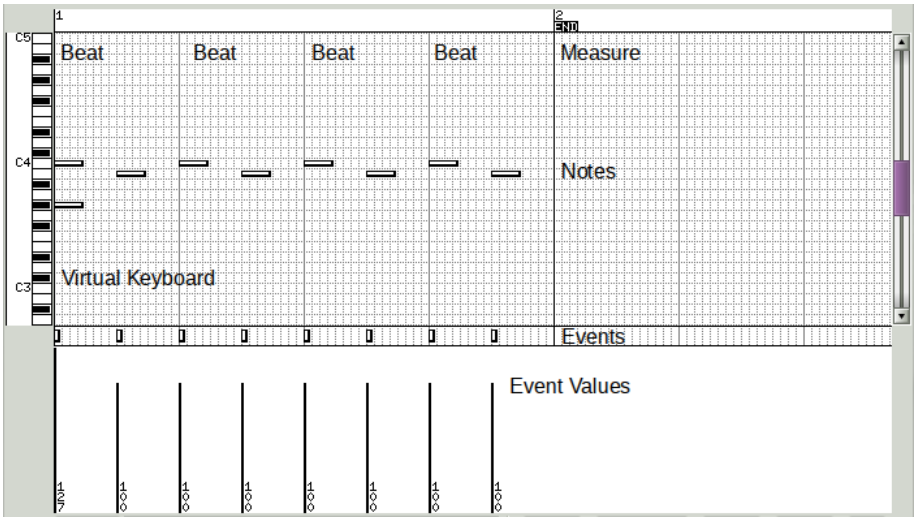



Figure 59: Pattern Editor, Piano Roll Items

1. **Beat**
2. **Measure**
3. **Virtual Piano Keyboard**
4. **Notes**
5. **Events**
6. **Event Values**

**1. Beat.** The light vertical lines represent the beats defined by the configuration for the pattern. The even lighter lines between the beats are useful for snapping notes.

**2. Measure.** The heavy vertical lines represent the measures defined by the configuration for the pattern. Also note that the end of the pattern occurs at the end of a measure, and is marked by a blocky **END** marker.

**3. Virtual Piano Keyboard.** The virtual keyboard is a fairly powerful interface. It shows, by shadowing, which note on the keyboard will be drawn. It can be played with a mouse, using left-clicks, to preview a short motif. It can show marks to indicate off-scale notes, to make them easy to avoid. Every octave, a note letter and octave number are shown, as in "C4". If there is a difference scale in force, then the letter changes to match, as in "F#5".

A right-click anywhere in the virtual keyboard area toggles the display between the octave note letters and the MIDI note numbers (only every other one is displayed due to space, to avoid cramped numbering). The following figure shows both views, superimposed for comparison.

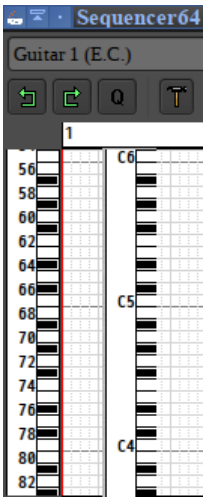



Figure 60: Pattern Editor, Virtual Keyboard Number and Note Views

**4. Notes.** Musical notes are indicated by thick horizontal bars with white centers. Each bar provides a visual representation of the pitch of a note and the length of a note.

**5. Events.** Also known as the "events pane" or "events panel". The narrow (a few pixels high) events strip shows discrete events, such as Note On and Note Off We recommend not editing or selecting events in that pane (feel free to disobey), but it is a good way to add events. Either left-click (to add one event), or left-click-drag horizontally (to add a series of events at the current note resolution.) One can also left-click in that section, then hit the `p` key to go into "paint" mode, and hit the `x` key to escape that mode.

**6. Event Values.** Also known as the "data pane" or "data panel".

The events values for the currently selected category of events are shown in this window as vertical lines of a height proportional to the value. These values can be easily modified by left-click-dragging the mouse past each line, to chop it off at the given value. Easier to try it than explain it. Right-click-drag also works the same.

### 4.3.2 Pattern Editor / Event Editing

When we say "editing" in the context of the piano roll, in part we mean that we will "draw" or "paint" notes. Drawing, modifying, copying, and deleting notes is actually very elegant in *Sequencer64* and in *Seq24*.

Note that, if only a middle-button is needed, `Ctrl-left` will simulate that button.

There is a feature to allow the Mod4 (the Super or Windows) key to keep the right-click in force even after it is released. See 2.2.8.5. Pressing `Mod4` before releasing the right-click that allows note-adding, keeps note-adding in force after the right-click is released. Now notes can be added at will with the left mouse button. Right-click again to leave the note-adding mode.

**New:** Another way to turn on the paint mode has been added. To turn on the paint mode, first make sure that the piano roll has the keyboard focus by left-clicking in it, then press the `p` key while in the pattern editor. This is just like pressing the right mouse button, but the draw/paint mode sticks (as if the `Mod4` mode were in force). To get out of paint mode, press the `x` key while in the pattern editor, to "x-scape" (get it? get it?) from the paint mode. These keys also work while the sequence is playing.

The `p` and `x` keys also works in the small event strip just above the white data area. The Mod4-right-click feature does not yet work in that area of the user-interface, but the `p` key does.

#### 4.3.2.1 Editing Note Events

The piano roll provides for a quite sophisticated set of note-editing actions. Not only is there a native mouse-interaction mode, but there is a "fruity" mouse-interaction mode that works more like the application *Fruity Loops*, its follow-on *FL Studio*, and its Linux look-alike, *LMMS*.

This option is currently *not available* in the **Qt 5** version of *Sequencer64*. We really want to refactor the code to be able to completely reconfigure the mouse interface according to any user's preferences, but that is currently a low priority.

**1. Fruity Mode.** At some point, we will add a section detailing the usage of the "fruity" mode of mouse-interaction. For now, section 2.2.8.5 "Menu / File / Options / Mouse" on page 28, such as it is, will have to do. Study the following paragraphs carefully, ideally while trying them out in *Sequencer64*.

**2. Enter Draw Mode.** In the note (grid/roll) panel, **holding** down the **right** mouse button will change the cursor to a pencil and put the editor into "draw" mode, also known as "note-adding" or "paint" mode. To exit the draw mode, release the right mouse button, and the cursor will turn back into an arrow. Another way to enter paint mode is to make sure the piano roll has focus (left click in it), and then press the `p` key. To exit the draw mode, press the `x` key.

**3. Add Notes.** If using the mouse, while **holding** the **right** mouse button, click the **left** mouse button to **insert** new notes. Many people find this combination strange at first, but once one gets used to it, it becomes a very fast method of note manipulation. Holding the `Mod4` key while releasing the right button keeps the mouse in draw mode. Note that this click will add a single note, and the length of the note will be that specified in the note-length setting (e.g. "1/16").

To increase the number of notes, keep dragging the mouse (with both buttons held). It can be dragged rightward, leftward, upward, and downward. Dragging left or right adds new notes, while dragging upward or downward moves the current note to a different pitch. This is the "auto-note" feature. Please note that the auto-note feature does *not* work with chord-generation. The draw mode has the following features:

- Notes are continually added as the mouse is dragged ("auto-notes").
- Notes cannot be added past the "END" marker of the pattern, which marks the **Sequence Length in bars** setting.
- As the mouse is dragged while the left button is held in draw mode, notes are either added, or, if already present at that note-on time, are moved up and down.
- If the draw mode is exited, and entered again, then the original notes will not be altered. Instead, new ones will be added.
- Notes can be added while the pattern is playing, and will be heard the next time the progress bar passes over them.

Thus, one can, with some care, draw a nice chorded sequence. Adjustments to it can be made afterward.

**4. Select Notes.** Adjustments can be made to one or more notes by selecting one or more notes, and then applying one or more special "selection actions" to the selection.

To select a single note, simply **left click** on it. The selected note will turn orange.

To select multiple notes, perform a **left click drag** to form a selection box that intersects (even partly) the desired notes. Once the mouse is released, all of the desired notes should be orange.

To add more notes to a selection of notes, move to an unselected note and perform a **ctrl left click drag** to form a selection box that intersects (even partly) the desired notes. Once the mouse is released, all of the desired notes should be orange. Be careful! If you ctrl-left-click-drag on an already-selected note, the drag will change the length of all of the notes in the selection.

Pressing the `Ctrl-A` key will select all of the events in the pattern editor.

The **Tools** button described in section 4.2 "Pattern Editor / Second Panel" on page 59can also be used to modify selections.

Once one or more notes are selected, they can be modified in time, pitch, or length.

**5. Deselect Notes.** To deselect the notes, click somewhere else in the piano roll, and the notes should change back to white. There is no way to deselect a single note, with, say, a shift-click or ctrl-click action.

**6. Move Notes in Pitch.** To move notes in pitch, once selected, grab one of the notes in the selection and drag it upward or downward. Also, since a selection is in force, the Up and Down arrow keys can also be used to change the pitch of every note in the selection. The smallest unit of pitch change is one MIDI note value.

**Warning:** If one moves the selection too low or too high in pitch, whether with the mouse or the arrow keys, any notes that go below the lowest MIDI pitch or above the highest MIDI pitch **will be lost**! If done using the mouse, the undo feature (Ctrl-z) will work. If done using the arrow keys, the undo feature does not work! Be careful, especially if you have a fast keyboard repeat rate!

**7. Move Notes in Time.** To move notes in time, once selected, grab one of the notes in the selection and drag it leftward or rightward. **New:** Also, since a selection is in force, the Left and Right arrow keys can also be used to change the time of every note in the selection. The smallest unit of time change is the **Grid snap** value, which might be a 16th note, for example.

Note that there is no possibility of note loss with a change in time. When a note disappears at one end of the pattern boundary, it wraps around to the other end. Cool.

**8. Change Note Length.** Pressing the **middle** mouse button ***or*** pressing the **ctrl left** mouse button in tandem, while the pointer is hovered over a selected note, will let one change the length of a selected note. If more than one note is selected, then the length of all selected notes is changed.

Once a selection of notes is made, one can use the shift-middle-click-drag or ctrl-left-click-drag sequence over the selected notes to draw a box beyond the extent of the notes. When the mouse is released, each of the events is moved and lengthed to be proportionally longer to fit exactly within the box one drew. This feature is called *event stretch*. If the box that was drawn was shorter than the original extent of the notes, then the notes move and shrink proportionally to occupy the smaller box. This feature is called *event compression*.

**Warning**: Reducing or increasing the length of a note selection by too much causes the note or notes to "wrap-around" to the end of the pattern boundary and grow more from the beginning of the sequence. It is not clear if this new note has an ending time that is less than its beginning time. If it happens, one probably ought to undo it.

**9. Copy/Paste.** Copying, cutting, and pasting is supported by selecting a number of events or notes, and using the Cut (`Ctrl-X`), Copy (`Ctrl-C`), Paste (`Ctrl-V`), and "drop" (`Enter`) keys. When the notes are selected, one can delete them with the `Delete` or `Backspace` key. If the events are *cut*, using the `Ctrl-X` key, then they can be pasted, using the `Ctrl-V` key. However, once `Ctrl-V` is struck, then one must *move* the mouse pointer to see where to paste the events, or move it with the arrow keys. An orange box representing the selected-and-copied notes appears, and the user should move the box (note) to the desired location and then left-click.

One can move the orange box using the arrow keys, to the desired location, and then hit the `Enter` key to drop the notes at that location.

Selected notes that are cut or copied can then be pasted into *other* pattern editor dialogs; that is, they can be pasted into other sequences.

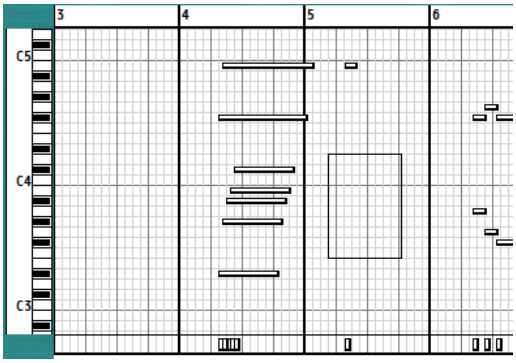



Figure 61: Piano Roll, Paste-Box for Cut Notes

Move this box to where pasting is desired, and left-click. The moved notes appear, still selected, and they can then be moved further, if desired, by using the arrow keys, or cut and move them again.

For the appearance of selected events (orange), see Figure 62 "Piano Roll, Selected Notes and Events" on page 72.

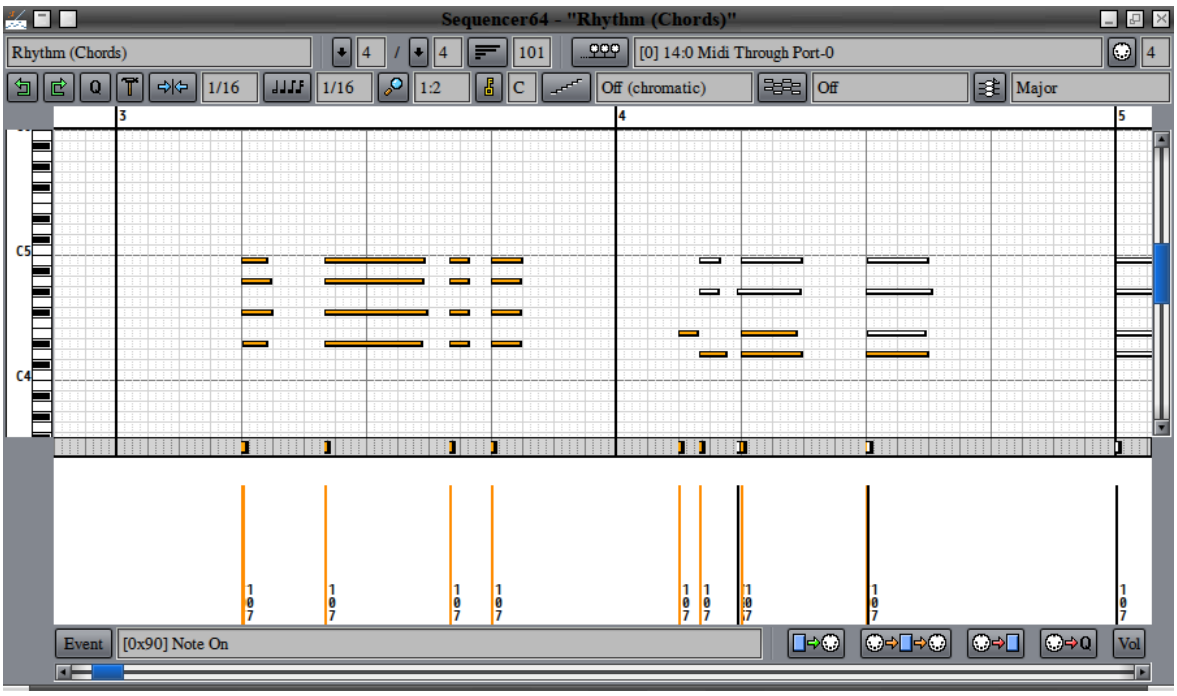



Figure 62: Piano Roll, Selected Notes and Events

The selection, shown in Figure 62 "Piano Roll, Selected Notes and Events" on page 72, illustrates the style of event selection, which colors the data bars as well as the event and note bars. This selection was made by first selecting one set of events by the left-click-drag action, then selecting more events by holding the `Ctrl` key for the next left-click drag action. The second selection left out some events, which are thus still shown as black bars in the data area.

#### 4.3.2.2 Event and Data Panels / Editing Other Events

*Note On* and *Note Off* events (and other events) can appear as small squares in the event strip, along with a black vertical bar with a height proportional to the velocity of the note event, plus a numeric representation of that value. Note events do not need to be inserted in the event strip. *They can be inserted there, but they end up as short events of the lowest possible note, 0 or C1, and they don't have a Note Off event. Don't do that!*)

Other event types can be inserted via the event strip. To do that, first select the kind of event to insert using the **Event** button in the bottom panel. The place the mouse cursor in the event strip. Right-click to make the drawing pencil appear at the exact spot where the event must go. While holding the right button, click the left button. A small square for the event should appear.

Should one want more of the same event, continue to hold both buttons and drag the mouse. One event should appear at each beat position (e.g. at each 16th note position) that is crossed.

To move the event(s) to a different spot, select it or them via the left button. Then drag it or them to where one wants them. it is currently not possible to move them to positions smaller than the beat size. The work-around is to temporarily reduce the beat size, but this requires caution.

Once the event positions are set, the next step is to modify the data values of the events. The event value (data) editor (directly under the event strip) is used to change note velocities, channel pressure,

control codes, patch select, etc. Just left-click+drag the mouse across the window to draw a line. The values will match that line. middle-click+drag and right-click+drag also draw the value line.

**Bug:** Sometimes the editing of event values in the event data section will not work. The workaround is to do a `Ctrl-A`, and the click in the roll to deselect the selection; that makes the event value editing work again.

Any events that are selected in the piano roll or event strip can have their values modified with the mouse wheel.

#### 4.3.2.3 Editing Note Events the "Fruity Way"

This mode is a lot different, and we have yet to do the exhaustive testing needed to understand how this mode works. Input from actual users of this mode would be welcome. For now, see section 2.2.8.5 "Menu / File / Options / Mouse" on page 28.

## 4.4 Pattern Editor / Bottom Panel

The bottom horizontal panel of the Pattern Editor provides for selecting events for viewing and editing, MIDI playback, pass-through, and recording.

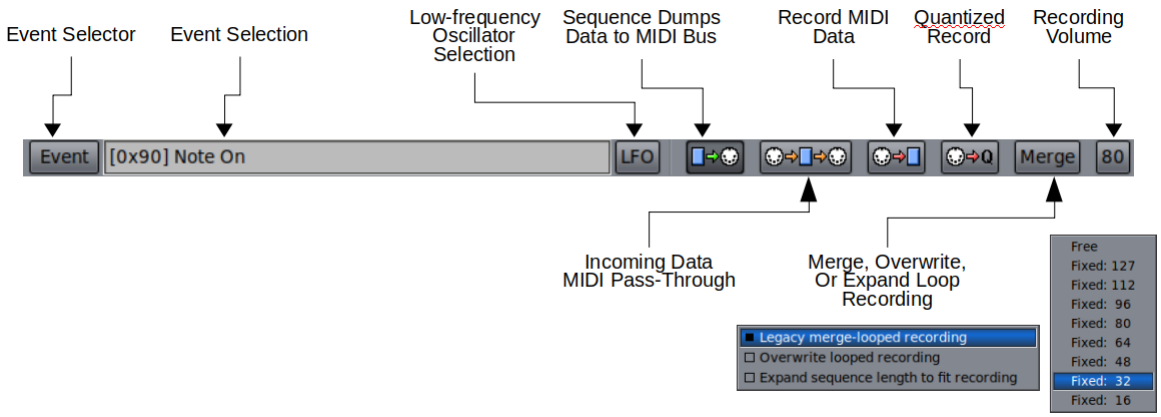



Figure 63: Pattern Editor, Bottom Panel Items

Missing from this diagram is the "Existing Event Selector" which zeroes in only on events already present in the pattern.

1. **Event Selector**
2. **Existing Event Selector**
3. **Event Selection**
4. **LFO**
5. **Data To MIDI Buss**
6. **MIDI Data Pass-Through**
7. **Record MIDI Data**
8. **Quantized Record**
9. **Recording Type** (Merge, Replace, Expand)
10. **Select Recording Volume**

**1. Event Selector.** This button brings up the following context menu, so that the user can select the category of events to view and edit.

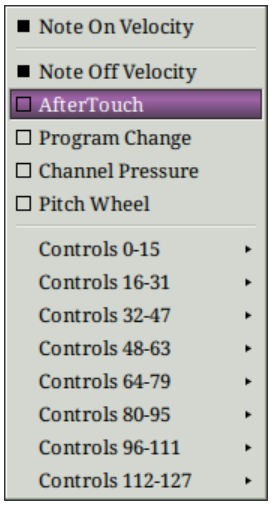



Figure 64: Pattern Editor, Event Button Context Menu

Note the squares. Some of filled (black), others are empty. The filled squares indicate that the sequence does indeed have some events of that type. Otherwise, there are no such events in the sequence. Useful in deciding if it is worth selecting the event.

The sub-menus of this context menu show 128 controller messages, so we won't try to show all of them here. They also use the squares to indicate if there are any events of the type shown in the menu. These sub-menus can be modified by editing the file

```
$HOME/.config/sequencer64/sequencer64.usr
```

to make it match one's instrument. See section 11 "Sequencer64 "usr" Configuration File" on page 124.

**2. Existing Event Selector.** Not shown in the figure above is a small button that will contain either an empty square (no editable events) or a black square (at least one editable event). Unlike the event-selector described above, this one shows only the actual events existing in the track, for quicker selection.

**3. Event Selection.** Shows the selection event, with its number shown in hexadecimal notation, and the name of the event shown.

**4. LFO.** A low-frequency oscillator allows data events can be modulated by some rudimentary wave functions. By clicking on the **LFO** button or using the `Ctrl-L` key, the following window appears, shown as the set of 5 vertical sliders:

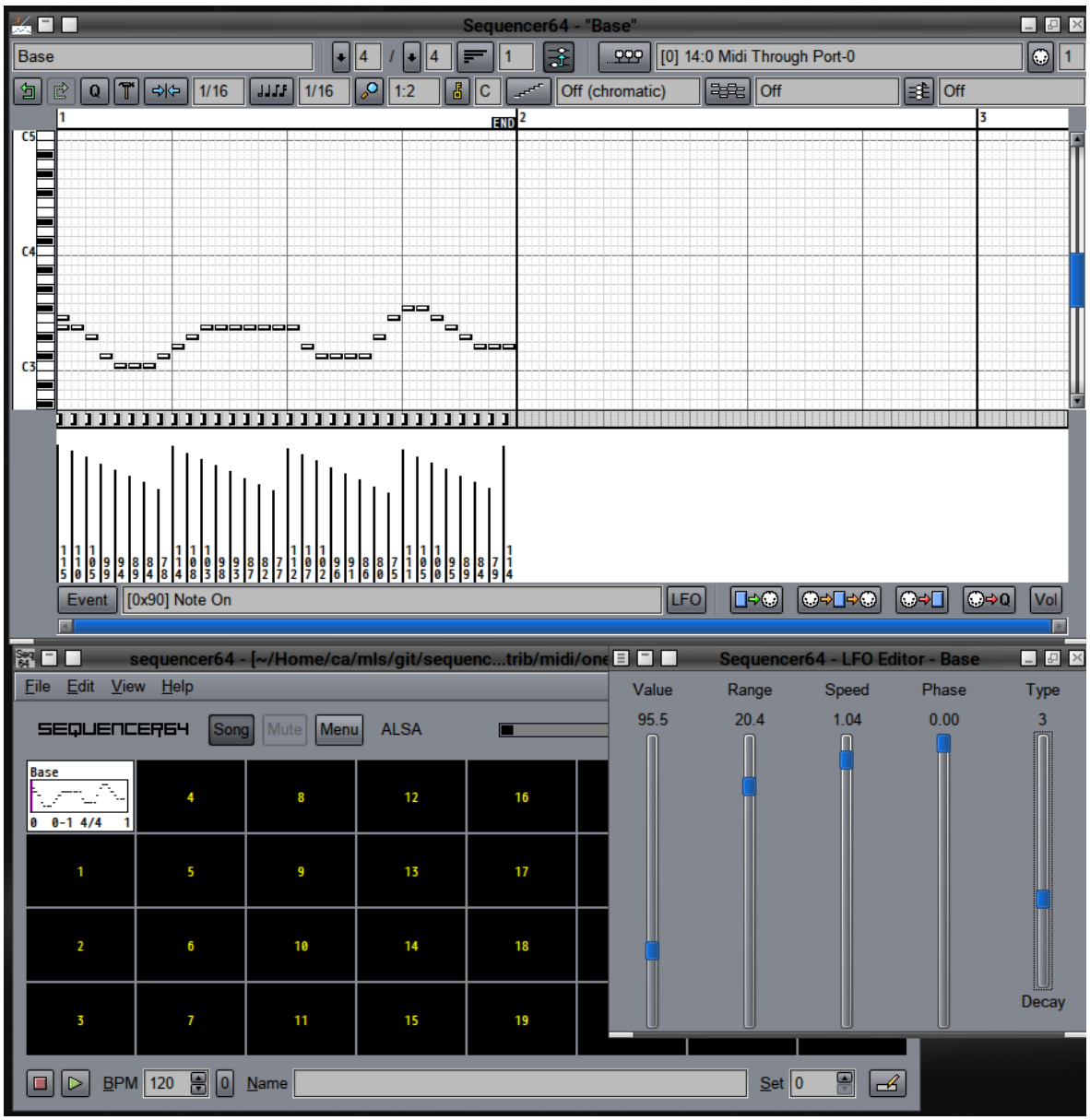



Figure 65: Pattern Editor, LFO Support

1. **Value**: Provides a kind of DC offset for the data value. Starts at 64, and ranges from 1 to 127.
2. **Range**: Controls the depth of modulation. Starts at 64, and ranges from 1 to 127.
3. **Speed**: Indicates the number of periods per pattern (divided by beat width, normally 4). For long patterns, this parameter needs to be set high, to even show an effect. It is also subject to an 'anti-aliasing' effect in some parts of the range, especially for short patterns. Try it! For short patterns, try a value of 1 at first. For a pattern of one measure in length, this will create four periods of the wave.
4. **Phase**: Provides the phase shift within a period of the LFO wave. A value of 1 is a phase shift of 360 degrees (or maybe it is one radian?).
5. **Wave Type**: Selects the kind of wave to use for the LFO:
    1. **Sine wave**.
    2. **Ramp (up) sawtooth**.

3. **Decay (down) sawtooth**.
4. **Triangle wave**.

We may have more to explain about this dialog at some point. For now, try it out on the file `one-measure.midi`, and be sure to hover over each control to see the tooltips. Note that it works best with short patterns.

**5. Time Scroll.** Allows one to pan through the whole pattern, if it is too long to fit in the window horizontally.

**6. Data To MIDI Buss.** This button causes the pattern to be output to the selected MIDI output buss, which will normally be connected to a software or hardware synthesizer, to be heard. Generally, this control should always be activated.

**7. MIDI Data Pass-Through.** This button routes incoming MIDI data through *Sequencer64*, which then writes it to the MIDI output buss.

**8. Record MIDI Data.** This button routes incoming MIDI data into *Sequencer64*, which then saves the data to its buffer, and also displays the new information (notes) in the piano roll view.

**9. Quantized Record.** This button will causes MIDI data to be recorded, but be quantized on the fly before recording it. The quantization is to the current snap value.

**10. Recording Type.** In *Seq24*, the pattern recording worked by merging new notes played as the pattern to be recorded was looped. This method allows a loop to be built up bit-by-bit. *Sequencer64* adds two more methods from Stazed's *Seq32* project. The three methods are:

1. **Merge**. This is the "legacy" style of recording loops, where notes can accumulate.
2. **Replace.** When the loop starts over, and a note is pressed, then the existing notes in that loop are erased, and the new note is added. This provides a good way of correcting major mistakes, live. It will not work if adding notes while not recording. This mode can cause incomplete notes if one holds the note and releases it in the next iteration, leaving a partially-drawn note behind. The workaround is to try again.
3. **Expand**. Once the end of the loop is near, whether or not any notes are being input, another measure is added to the length of the loop. This continues indefinitely, whether or not any notes are being played/recorded.

**11. Vol.** This button allows setting the volume of the recording. The velocity of the notes will be set to the selected value upon recording. If the **Free** item is selected, then the incoming note velocity is preserved.

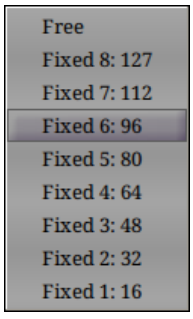



Figure 66: Pattern Recording Volume Menu

The velocity values are shown at the right side of each menu entry. These values correspond to MIDI volume levels from 127 down to 16, as shown in the figure.

## 4.5 Pattern Editor / Common Actions

This section is a catch-all for actions not described above.

### 4.5.1 Pattern Editor / Common Actions / Scrolling

Let us describe the actions that can be performed with a scroll wheel, or with the scrolling features of multi-touch touchpads. There are three major scrolling actions available when using mouse scrolling, with the mouse hovering in the piano-roll area:

- **Vertical Panning (Notes Panning)** Using the vertical scroll action of a mouse or touchpad moves the view of the sequence/pattern notes up and down. One can also click in the piano roll, and then use the `Page-Up` and `Page-Down` keys to move the view up and down in pitch.
- **Horizontal Panning (Timeline Panning)** Holding the Shift key, and then using the vertical scroll action of a mouse or touchpad moves the view of the sequence/pattern time forward and backward. One can also click in the piano roll, and then use the `Shift Page-Up` and `Shift Page-Down` keys to move the view left and right in time.
- **Horizontal Zoom (Timeline Zoom)** Holding the Ctrl key, and then using the vertical scroll action of a mouse or touchpad zooms the view of the sequence/pattern time to compress it or expand it. One can also click in the piano roll, and then use the `z` , `Z` , and `0` keys to change the timeline zoom.

The actions of this scrolling are smooth and fast. If an event is selected in the piano-roll area or the (thin) event area, then the scrolling increases or decreases the value of the event. In the case of a note, this increases or decreases the velocity of the note. For all events, this increases or decreases the length of the vertical line that represents the value of the event.

### 4.5.2 Pattern Editor / Common Actions / Close

There is no **Close** button in the pattern editor. One can use window-manager actions, such as clicking on the X button of the window frame, or pressing the exit key defined in the window manager. *Sequencer64* also provides the Ctrl-W key to close the pattern editor window. However, be aware that this convention does not apply to the other application windows of *Sequencer64*.

# 5 Song Editor

The *Sequencer64 Song Editor* combines all patterns into a complete tune with specified repetitions of each pattern. It shows one row per pattern/loop/sequence in numbered columns, with the placement of each pattern at various time locations in the song. In *Sequencer64* parlance, the song editor creates a *performance*, and the performance is implemented by a set of triggers (see section 16.1.22 "Concepts / Terms / trigger" on page 164).

As an option in the `[user-interface]` section of the "user" configuration file, two song editor windows can be brought onscreen, as a convenience for arranging projects with a large number of sequences/patterns. (In the Qt user-interface, a Song tab and a Song window can be shown at the same time.) The Song window, when in focus, also activates the **Song** mode of *Sequencert64*, as opposed to the **Live** mode activated by the Patterns panel when it is in focus.

When the song editor has the focus of the application, it takes over control from the patterns panel. The song editor then controls playback. Once playback is started in the song editor, some actions in the patterns panel no longer have effect, effectively disabling live mode. The song editor takes over the arming/unarming (unmuting/muting) shown in the patterns panel. The highlighting of armed/unarmed patterns changes according to whether the pattern is playing in the song editor, or not. If one tries to change the muting using a hot-key (or even a click) in the patterns panel, the song editor immediately returns the pattern to the state it has in the song editor. The only way to manually change the muting then is to click the pattern's label in the song editor. Both the song editor and the patterns panel both reflect the change in muting in the user-interface, though with *opposite colors*.

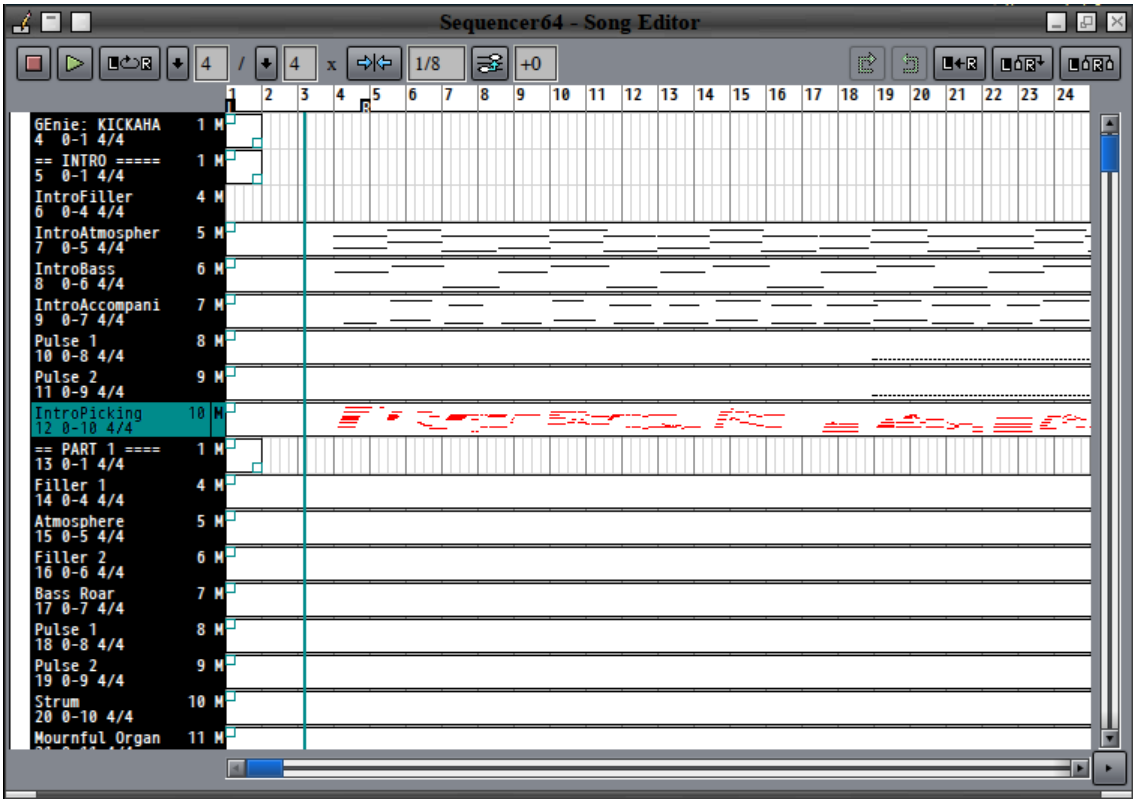



Figure 67: Song Editor Window

Here are some features for the song editor, as seen above and in the following figure:

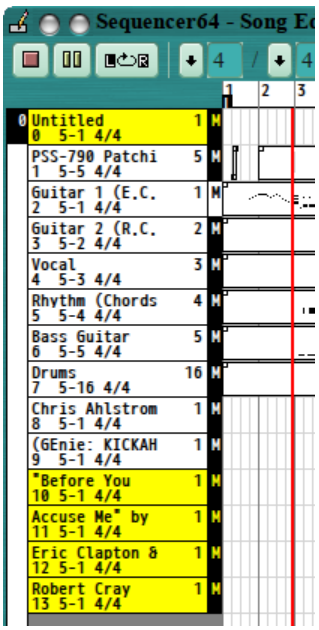



Figure 68: Song Editor Window, Features

- Toggling of the mute state of multiple patterns by holding the Shift key while left-clicking on the M or a pattern name.
- Pause button functionality.
- Optional coloring (selected in the Patterns panel) and thickening of the progress bar.
- A Redo button (not shown).
- A Transpose button (not shown).
- Red coloring of events for patterns that are not transposable, such as drum tracks.

This window (in *Sequencer64*) shows any empty patterns highlighted in yellow (Gtkmm only). An empty pattern is one that exists, but contains only meta information, and contains no MIDI events that can be played. For example, some tracks just serve as name tracks or information tracks.

The song editor is not too complex, but for exposition, we break it into the top panel, the bottom panel, and the rest of the window.

There is a button that allows the whole song (except for exempt, non-transposable sequences) to be transposed up or down by up to an octave in either direction.

## 5.1 Song Editor / Top Panel

The top panel provides quick access to song-playback actions and configuration.

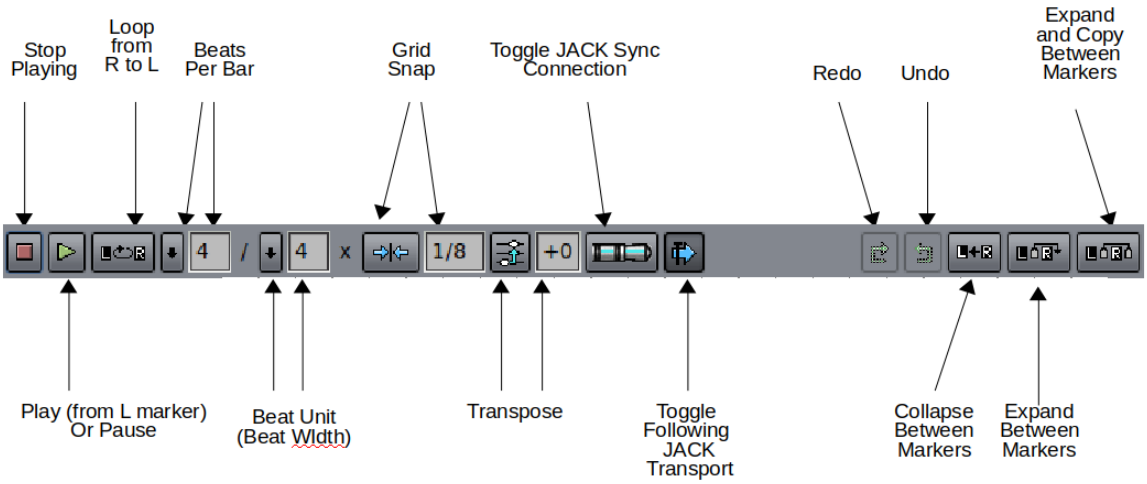



Figure 69: Song Editor / Top Panel Items

1. **Stop**
2. **Play**
3. **Play Loop**
4. **Beats Per Bar**
5. **Beat Unit**
6. **Grid Snap**
7. **Transpose**
8. **Toggle JACK Sync**
9. **Toggle Following JACK Transport**
10. **Redo**
11. **Undo**
12. **Collapse**
13. **Expand**
14. **Expand and copy**

**1. Stop.** Stops the playback of the song. The keystroke for stopping playback is the `Escape` character. It can be configured to be another character (such as `Space`, which would make the space-bar toggle the playback status).

**2. Play.** Starts the playback of the song, starting at the **L marker**. The **L marker** serves as the start position for playback in the song editor. One can change the start position only when the performance is not playing. The default keystroke for starting playback is the `Space` character. The default keystroke for stopping playback is the `Escape` character. The default keystroke for pausing playback is the `Period` character.

**3. Play Loop.** Activates loop mode. When Play is activated, play the song and loop between the **L marker** and the **R marker**. This button is a state button, and its appearance indicates when it is depressed, and thus active. If this button is deactivated during playback, then playback continues past the **R marker**.

**4. Beats Per Bar.** Part of the time signature, and specifies the number of beat units per bar. The possible values range from 1 to 16.

**5. Beat Unit.** Also called "beat width". Part of the time signature, and specifies the size of the beat unit: 1 for whole notes; 2 for half notes; 4 for quarter notes; 8 for eight notes; and 16 for sixteenth notes.

**6. Grid Snap.** Grid snap selects where the patterns are drawn. Unlike the **Grid Snap** of the pattern editor, the units of the song editor snap value are in fractions of a measure length. The following values are supported: 1, 1/2, 1/4, 1/8, 1/16, 1/32, and 1/3, 1/6, 1/12, and 1/24.

**7. Redo.** The Redo button reapplies the last change undone by the Undo button. It is inactive if there is nothing to redo.

**8. Undo.** The Undo button rolls back the last change in the layout of a pattern. Each time it is clicked, the most recent change is undone. It rolls back one change each time pressed. It is inactive if there is nothing to undo.

**9. Collapse.** This button collapses the song between the **L marker** and the **R marker**. What this means is that, if there is song material (patterns) before the **L marker** and after the **R marker**, and the **Collapse** button is pressed, any song material between the L and R markers is erased, and the song material after the **R marker** is moved leftward to the **L marker**. Collapsing occurs in all tracks present in the song editor.

**10. Expand.** This button expands the song between the **L marker** and the **R marker**. It inserts blank space between these markers, moving the song material that is after the **R marker** to the right by the duration of the blank space. Expansion occurs in all tracks present in the song editor.

**11. Expand and copy.** This button expands the song between the **L marker** and the **R marker** much like the **Expand** button. However, it also copies the original data that is present after the **R marker**, and pastes it into the newly-available space between the L and R markers.

## 5.2 Song Editor / Arrangement Panel

The arrangement panel is the middle section shown in Figure 67 "Song Editor Window" on page 78. It is also known as the "piano roll" of the song editor. Here, we zero in on its many features.

Keystrokes and additional mouse configuration have been added to make editing easier even without a good mouse. For example, one can page up and down vertically in the arrangement panel using the Page Up and Page Down keys. One can go to the top using the Home key, to the bottom using the End key. One can page left and right horizontally in the arrangement panel using the Shift Page Up and Shift Page Down keys. One can go to the leftmost position using the Home key, and to the rightmost position using the End key,

The following figure is taken from a conventional MIDI file, imported, with a few long tracks, rather than a large number of smaller patterns. In other words, the patterns used here are very long, and used only once in the song.

If playback is started with the song editor as the active window, then the pattern boxes in the patterns panel show as armed/unarmed (unmuted/muted) depending upon whether or not the pattern is shown as playing (or not) at the current playback position in the song editor piano roll.

The following figure highlights the main features of the center panel of the song editor.

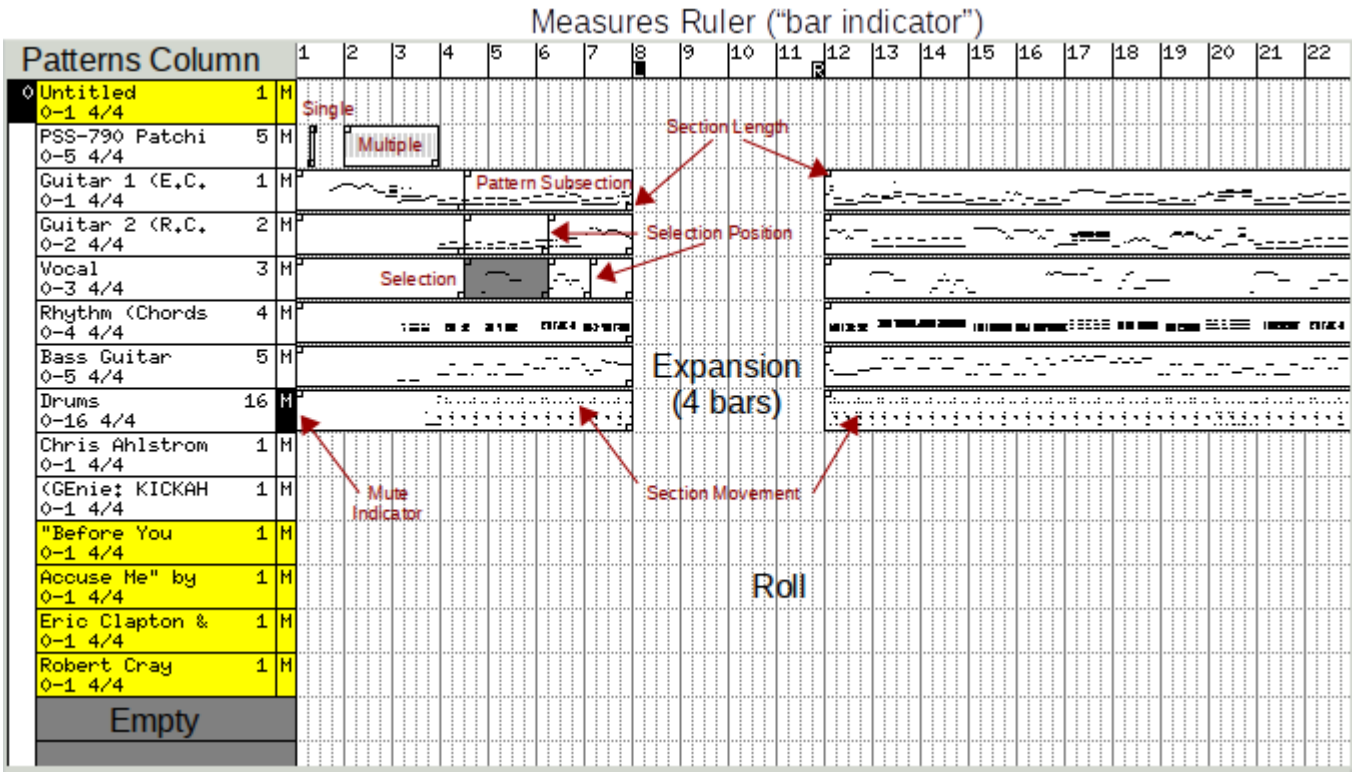



Figure 70: Song Editor Arrangement Panel, Annotated

One new feature of the song editor is that, if the new Transpose feature is built into *Sequencer64*, any patterns that are marked as exempt from transposition (common with drum tracks) have their events shown in red instead of black.


Figure 71: Song Editor for Non-Transposable Patterns

The measures ruler (measures strip) consists of a *measures ruler* (bar indicator) at the top, a numbered patterns column at the left with a muting indicator, and the grid or roll section. There are a lot of hidden details in the arrangement panel, as the figure shows. Here are the main sections:

1. **Patterns Column**
2. **Piano Roll**
3. **Measures Ruler**

These items are discussed in the following sections.

### 5.2.1 Song Editor / Arrangement Panel / Patterns Column

Here are the items to note in the patterns column:

1. **Number**. Not yet sure what the number on the left means. The number of the screen set?

2. **Title**. The title is the name of the pattern, for easy reference.
3. **Channel**. The channel number appears (redundantly) at the right of the title.
4. **Buss-Channel**. This pair of numbers shows the MIDI buss number used in the pattern and the channel used for the pattern.
5. **Beat/Measure**. This pair of numbers is the standard time-signature of the pattern.
6. **Mute Indicator**. The letter M is in a black box if the track/pattern is muted, and a white box if it is unmuted. Left-clicking on the "M" (or the name of the pattern) mutes/unmutes the pattern. If the Shift key is held while left-clicking on the M or the pattern name, then the mute/unmute state of every other active pattern is toggled. This feature is useful for isolating a single track or pattern.
7. **Empty Track**. Completely empty tracks (no track events or meta events) are indicated by a dark-gray filling in the pattern column. Tracks that have only meta information, but no playable event, are indicated by a yellow filling in the pattern column.

The patterns column shows a list of all of the patterns that have been created in the current song. Each pattern in this list has a track of pattern layouts associated with it in the piano roll section.

Left-clicking on the pattern name or the "M" button toggles the muting (arming) status of the track. It does the same thing if the `Ctrl` key is held at the same time.

Shift-left-clicking on the pattern name or the "M" button toggles the muting (arming) status of *all other tracks* except the track that was selected. This action is useful for quickly listening to a single sequence in isoloation.

Right-clicking on the pattern name or the "M" button brings up the same pattern editing menu as discussed in section 3.2.2 "Pattern" on page 44. Recall that this context menu has the following entries: **Edit...**, **Event Edit...**, **Cut**, **Copy**, **Song**, **Disable Transpose**, and **MIDI Bus**.

### 5.2.2 Song Editor / Arrangement Panel / Piano Roll

The "Piano Roll" section of the arrangement panel is where patterns or subsections are inserted, deleted, shrunk, lengthened, or moved. Here are features to note in the annotated piano roll area shown in Figure 70 "Song Editor Arrangement Panel, Annotated" on page 82:

1. **Single**. In the diagram, under the word "Single", is a very small pattern. It is small because it consists only of some MIDI Program Change messages meant to set the programs on a Yamaha PSS-790 keyboard.
2. **Multiple**. This item is the same pattern as in "Single", but dragged out for multiple repetitions, simply to show how even the shortest patterns can be replicated easily.
3. **Pattern Subsection**. Middle-clicking inside a pattern inserts a selection position marker in it, breaking the pattern into two equal pieces. We call each piece a *pattern subsection*. This division can be done over and over. In the song editor, a middle-click *cannot* be simulated by ctrl-left-click.)
4. **Selection Position**. A selection position is a marker that divides a pattern into two pieces, called *pattern subsections*. This makes it easy to select smaller portions of a pattern for editing or deleting. It is especially useful for making holes in a pattern.
5. **Selection**. Clicking inside a pattern or a pattern subsection darkens it (gray) to denote that it is selected. A pattern subsection can be deleted by the Delete key, copied by the `Ctrl-C` key, and then inserted (one or more times) by the `Ctrl-V` key. When inserted, each insert goes immediately after the current item or the previous insertion. The same can be done for whole patterns.
6. **Section Length ("handle")**. Looking closely at the diagram where the arrows point, small squares in two corners of the patterns can be seen. Call them "handles". By grabbing a handle with a left-click, the handle can be moved horizontally to either lengthen or shorted the pattern or

pattern subsection, if there is room to move in the desired direction. It doesn't matter if the item is selected or not.

7. **Section Movement**. If, instead of grabbing the section length handle, one grabs inside the pattern or pattern subsection, that item can be moved horizontally, as long as there is room. A left-click inside the item shows it as selected. One can also highlight a pattern section (making it gray), then click the `p` key to enter "paint" mode, and move the pattern left or right with the arrow keys.
8. **Expansion**. Originally, all the long patterns of this sample song were continuous. But, by setting the L and R markers, and using the **Expand** button, we opened up some silent space in the song, just to be able to show it off.

A useful feature is to split a pattern section in the song editor, either in half, or at the nearest snap point to the mouse pointer. The location of the split is determined by the setting of the **File / Options / Mouse / Sequencer64 Options / Middle-click splits song trigger at nearest snap (instead of halfway point)** setting. To split a pattern, left-click it to highlight it, move the mouse (if not splitting in half) to the desired pointer, and ctrl-left-click with the mouse.

The *Seq24* help files refer to work in the song editor as the "Performance Editor" or "Performance Mode". Adding a pattern in this window is a bit like adding a note in the pattern editor. One clicks, holds, and drags the mouse to insert a copy or copies of the pattern associated with the row in which one is dragging. The longer one drags, the more copies of the pattern that are inserted.

Right-click on the arrangement panel (roll) to enter draw mode, and hold the button. Just like the patterns panel, there is a feature to allow the Mod4 (the Super or Windows) key to keep the right-click in force even after it is released. See 2.2.8.5. Pressing Mod4 before releasing the right-click that allows pattern-adding ("painting"), keeps pattern-adding in force after the right-click is released. Now pattern can be entered at will with the left mouse button. Right-click again to leave the pattern-adding mode.

Another way to turn on painting: make sure that the performance editor piano roll has the keyboard focus by left-clicking in it, then press the `p` key. While in the paint mode, one can add pattern clips with the left mouse button, via click or drag, and one can highlight a pattern clip and move it with the left and right arrow keys.

To get out of the paint mode, press the `x` key while in the sequence editor, to "x-scape" from the paint mode. These keys, however, do not work while the sequence is playing.

The song editor supports zoom in the piano roll. This feature is not accessible via a button or a menu entry – it is accessible only via keystrokes. After one has left-clicked in the piano roll, the `z`, `Z`, and `0` can be used to zoom the piano-roll view. The `z` key zooms out, the `Z` key zooms in, and the `0` key resets the zoom to the default value. The zoom feature also modifies the time-line.

A left-click with a simultaneous right-click-hold inserts one copy of the pattern. The inserted pattern shows up as a box with a tiny representation of the notes visible inside. Some patterns can be less than a measure in length, resulting in a tiny box. To keep adding more copies of the pattern, continue to hold both buttons and drag the mouse rightward.

Middle-click on a pattern to drop a new selection position into the pattern, which breaks the pattern into two equal *pattern subsections*. Each middle-click on the pattern adds a new selection position, halving the size of the subsections as more pattern subsections are added.

When a pattern or a pattern subsection is left-clicked in the piano roll, it is marked with a dark gray filling. When a right-left-hold-drag action is done in this gray area, the result is to *delete* that pattern section or subsection. One can also hit the Delete key to *delete* that pattern section or subsection.

### 5.2.3 Song Editor / Arrangement Panel / Measures Ruler

The *measures ruler* is the ruled and numbered section at the top of the arrangement panel. It provides a place to put the left and right markers. In the *Seq24* documentation, it is called the "bar indicator".

Left-click in the measures ruler to move and drop an **L marker** (**L anchor**) on the measures ruler. Right-click in the measures ruler to drop an **L marker** (**R anchor**) on the measures ruler.

Once these anchors are in place, one can then use the *Collapse* and *Expand* buttons to modify the placement of the pattern events.

Note that the **L marker** serves as the start position for playback in the song editor. One can change the start position only when the performance is not playing.

Another way to move the L and R markers, a so-called "movement mode" has been added. To turn on the movement mode, first make sure that the piano roll (not the bar indicator!) has the keyboard focus by left-clicking in it at a blank spot, then press the `l` key or `r` key or while in the sequence editor. *There is no visual feedback that one is in the movement mode.* Then press the left or right arrow key to move the "L" or "R" (depending whether `l` or `r` was used to enter the movement mode) markers by one snap value at a time. To get out of the movement mode, press the `x` key while in the performance editor, to "x-scape".

## 5.3 Song Editor / Bottom Panel

The bottom panel is simple, consisting of a stock horizontal scroll bar and a small button, called the **Grow** button, labelled with a ">".

The **Grow** button adds to the number of measures that exist in the song editor. The visual effect is very subtle, resulting only in a small change in the thumb of the horizontal scroll-bar, unless one is at the right end of the piano roll. Then, one can see the added measures. Usually about 128 at a time are added, but this depends on the value of PPQN in force.

# 6 Event Editor

The *Sequencer64 Event Editor* is used to view and edit, in detail, the events present in a sequence/pattern/track. This editor is not very sophisticated. It is a basic editor for simple edits and viewing. Viewing and scrolling generally work; editing, deleting, and inserting events work. But there are many possible interactions between event links (Note Off events linked to Note On events), performance triggers, and the pattern, performance, and event editor dialogs. Surely some bugs still lurk. If anything bad happens, do *not* press the **Save to Sequence** button! If the application aborts, let us know!

Here are the major "issues":

1. It requires the user to know the details about MIDI events and data values.
2. It does not present handy dropdown lists for various items.
3. It does not detect any changes made to the sequence in the pattern editor, and so both editors cannot be brought on screen at the same time for the same pattern.
4. It does not have an undo function.
5. It cannot mark more than one event for deletion or modification.
6. There is no support for dragging and dropping of events.

The event editor is a good way to see the events in a sequence, and to delete or modify problematic events. Additionally, it can be used to add **Set Tempo** meta events. If an event is added that has a time-stamp beyond the current length of the sequence, then the length of the sequence is extended. Unlike the event pane in the pattern editor, the event-editor dialog shows all types of events at once.

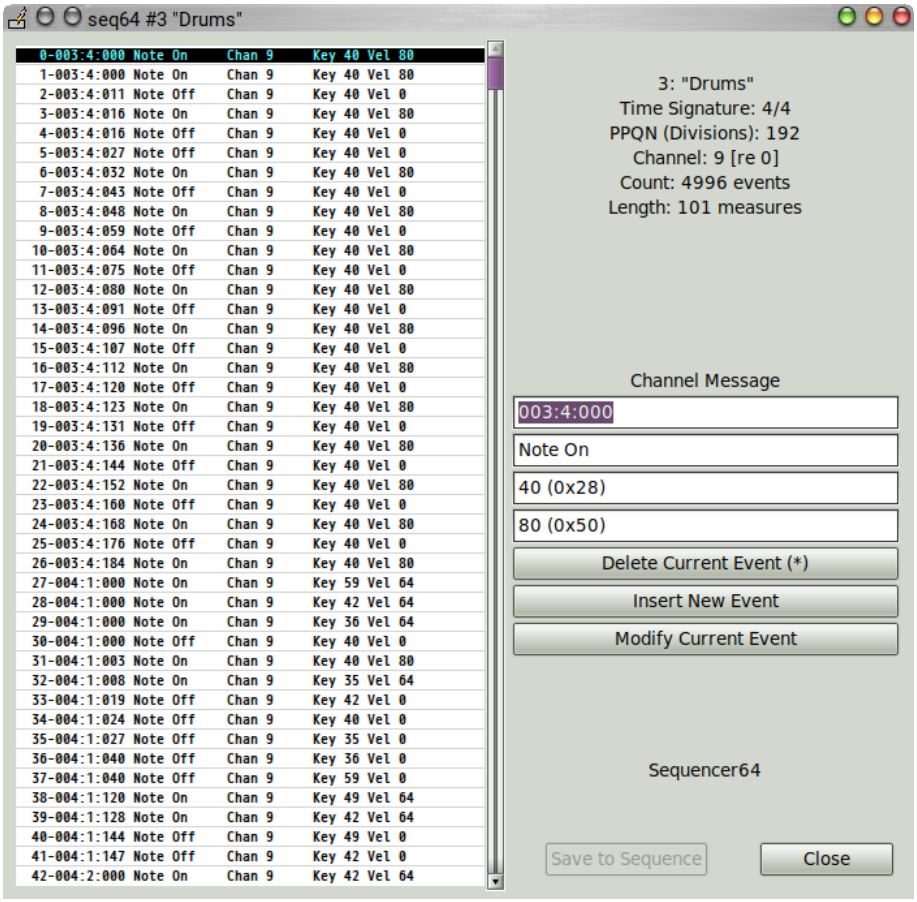



Figure 72: Event Editor Window

The event-editor dialog is fairly complex. For exposition, we break it down into a few sections:

1. **Event Frame**
2. **Info Panel**
3. **Edit Fields**
4. **Bottom Buttons**

The event frame is a list of events, which can be traversed, and edited. The fields in the right panel show the name of the pattern containing the events and other information about the pattern. The edit fields provide text fields for viewing and entering information about the current event, and buttons to delete, insert, and modify events. The bottom buttons allow changes to be saved and the editor to be closed. The following sections described these items in detail.

## 6.1 Event Editor / Event Frame

The event frame is the event-list shown on the left side of the event editor. It is accompanied by a vertical scroll-bar, for moving one line or one page at a time. Mouse or touchpad scrolling can be used to move

up and down in the event list. This movement is even easier than reaching for the scrollbars.

### 6.1.1 Event Frame / Data Items

The event frame shows a list of numbered events, one per line. The currently-selected event is highlighted in cyan text on a black background. Here is an example of the data line for a MIDI event:

```
17-003:3:128 Note On   Chan 3    Key 66 Vel 107
```

This line consists of the following parts:

1. **Index Number**
2. **Time Stamp**
3. **Event Name**
4. **Channel Number**
5. **Data Bytes**

**1. Index Number.** Displays the index number of the event. This number is purely for reference, and is not part of the event. Events in the pattern are numbered from 0 to the number of events in the pattern.

**2. Time Stamp.** Displays the time stamp of the event, which indicates the cumulative time of the event in the pattern. It is displayed in the format of "measure:beat:divisions". The measure values start from 1, and range up to the number of measures in the pattern. The beat values start from 1, and range up to the number of beats in the measure. The division values range from 0 up to one less than the PPQN (pulses per quarter note) value for the whole song. As a shortcut, one can use the dollar sign ("$") to represent PPQN-1.

**3. Event Name.** Displays the name of the event. The event name indicates what kind of MIDI event it is. The following event names are supported:

1. **Note Off**
2. **Note On**
3. **Aftertouch**
4. **Control Change**
5. **Program Change**
6. **Channel Pressure**
7. **Pitch Wheel**
8. **Tempo**

**4. Channel Number.** Shows the channel number (for channel-events only) re 0, not 1. For the user, of course, MIDI channels always range from 1 to 16. Internally, they range from 0 to 15.

**5. Data Bytes.** Shows the one or two data bytes for the event.

Note Off, Note On, and Aftertouch events requires a byte for the key (0 to 127) and a byte for the velocity (also 0 to 127). Control Change events require a control code and a value for that control code. Pitch wheel events require two bytes to encode the full range of pitch changes. Program change events require only a byte value to pick the patch or program (instrument) to be used for the sequence. The Channel Pressure event requires only a one-byte value. Tempo requires a number (e.g. "120.3") to be typed in.

### 6.1.2 Event Frame / Navigation

Moving about in the event frame is straightforward, but has some wrinkles to note. Navigation with the mouse is done by moving to the desired event and clicking on it. The event becomes highlighted, and its data items are shown in the "info panel". There is no support for dragging and dropping events in the event frame.

The scrollbar can be used to move within the frame, either by one line at a time, or by a page at a time. A page is defined as one frame's worth of lines, minus 5 lines, for some overlap in paging.

Navigation with keystrokes is also supported, for the Up and Down arrows and the Page-Up and Page-Down keys. Note that using the Up and Down arrows by holding them down for awhile causes autorepeat to kick in, and the updates become very erratic and annoying. Use the scrollbar or page keys to move through multiple pages. Home and End also work.

## 6.2 Event Editor / Info Panel

The "info panel" is simply a read-only list of properties on the top right of the event editor. It serves to remind the used of the pattern being edited and some characteristics of the pattern and the whole song. Five items are shown:

1. **Sequence Number and Name**. A bit redundant, as the window caption for the event editor also shows the pattern name. It can be set in the pattern editor.
2. **Time Signature**. A pattern property, shown only as a reminder. It can be set in the pattern editor.
3. **PPQN** Shows the "parts per quarter note", or resolution of the whole song. The default PPQN of *Sequencer64* is 192.
4. **Sequence Channel** In *Sequencer64*, the channel number is a property of the pattern. All channel events in the pattern get routed to the same channel, even if somehow the event itself specifies a different channel.
5. **Sequence Count** Displays the current number of events in the pattern. This number changes as events are inserted or deleted.

## 6.3 Event Editor / Edit Fields

The edit fields show the values of the currently-selected event. They allow changing an event, adding a new event, or deleting the currently-selected event.

1. **Event Category** (read-only)
2. **Event Timestamp**
3. **Event Name**
4. **Data Byte 1**
5. **Data Byte 2**
6. **Delete Current Event**
7. **Insert New Event**
8. **Modify Current Event**

**Important**: changes made in the event editor are *not* written to the sequence until the **Save to Sequence** button is clicked. If one messes up an edit field, just click on the event again; all the fields will be filled in again. That's as much "undo" as the event-editor offers at this time, other than closing without saving.

**1. Event Category.** Displays the event category of the event. Currently, only channel events can be handled, but someday we hope to handle the wide array of system events, and perhap even system-exclusive events.

**2. Event Timestamp.** Displays the timestamp of the event. Currently only the "measure:beat:division" format is fully supported. We allow editing (but not display) of the timestamp in pulse (divisions) format and "hour:minute:second.fraction" format, but there are bugs to work out.

If one wants to delete or modify an event, this field does not need to be modified. If this field is modified, and the **Modify Current Event** button is pressed, then the event will be moved. This field can locate a new event at a specific time. If the time is not in the current frame, the frame will move to the location of the new event and make it the current event.

**3. Event Name.** Displays the name of the event, and allows entry of an event name. The event name indicates what kind of MIDI event it is. The following event names are supported:

1. **Note Off**
2. **Note On**
3. **Aftertouch**
4. **Control Change**
5. **Program Change**
6. **Channel Pressure**
7. **Pitch Wheel**
8. **Tempo**

Typing in one of these names changes the kind of event if the event is modified. Abbreviations and case-insensitivity can be used to reduce the effort of typing. This handling of the editing of the event name is still a bit clumsy. It would be better to provide a drop-down list for more painless selection of events. Some day.

**4. Data Byte 1.** Allows modification of the first data byte of the event. One must know what one is doing. The scanning of the digits is very simple: start with the first digit, and convert until a non-digit is encountered. The data-byte value can be entered in decimal notation, or, if prepended with "0x", in hexadecimal notation.

**5. Data Byte 2.** Allows modification of the second data byte of the event (if applicable to the event). One must know what one is doing. The scanning of the digits is as noted above.

**6. Delete Current Event.** Causes the selected event to be deleted. The frame display is updated to move following events upward.

*Sequencer64* would support using the Delete and Insert keys to supplement the buttons, but the Delete key is needed for editing the event data fields. The current structure of the dialog prevents using it for both the frame and the edit fields. Therefore, *Sequencer64* allows the usage of the **asterisk** keys (regular and keypad) for deletion.

**7. Insert New Event.** Inserts a new event, described by the **Event Timestamp**, **Event Name**, **Data Byte 1**, and **Data Byte 2** fields. The new event is placed in the appropriate location for the given timestamp. If the timestamp is at a time that is not visible in the frame, the frame moves to show the new event, so be careful.

**8. Modify Current Event.** Deletes the current event, and inserts the modified event, which is placed in the appropriate location for the given timestamp.

### 6.4 Event Editor / Bottom Buttons

The buttons at the bottom of the event editor round out the functionality of this dialog.

1. **Save to Sequence**
2. **Close**

**1. Save to sequence.** Saves the event container back to the sequence from whence the events came. This button does not close the dialog; further editing can be performed. The Save button is enabled only if some unsaved changes to the events exist.

Any sequence/pattern editor that is open should be reflected in the pattern editor once this button is pressed. However, at present, simultaneous use of the pattern editor and event editor has been disabled. If both the event editor and the pattern editor are open for a sequence (currently disabled), and some events are deleted in the event editor, and the **Save to Sequence** button is pressed, the pattern editor would crash and takes down *Sequencer64* with it. Therefore, when either editor is open for a given sequence, the right-click menu entries that bring them up are hidden.

**2. Close.** Closes the event editor. Any unsaved event changes are discarded. There is a "modification indicator" to show that the events have been modified.

Again, good luck with the dialog. Bug reports are appreciated.

## 7 Import/Export

This section explains the details of the MIDI import and export functionality, accessed by the main menu as noted in sections 2.2.5, 2.2.6, and 2.2.7, on page 17.

### 7.1 Import MIDI

The **Import** menu entry imports an SMF 0 or SMF 1 MIDI file as one or more patterns, one pattern per track. Even long tracks, that aren't short loops, are imported. The difference from **File / Open** is that the destination screen-set (bank) for the import can be specified, and the existing data in the already-loaded MIDI file is preserved. If the imported file is a *Sequencer64* MIDI file, it's proprietary sections will *not* be imported, in order to preserve the performance setup.

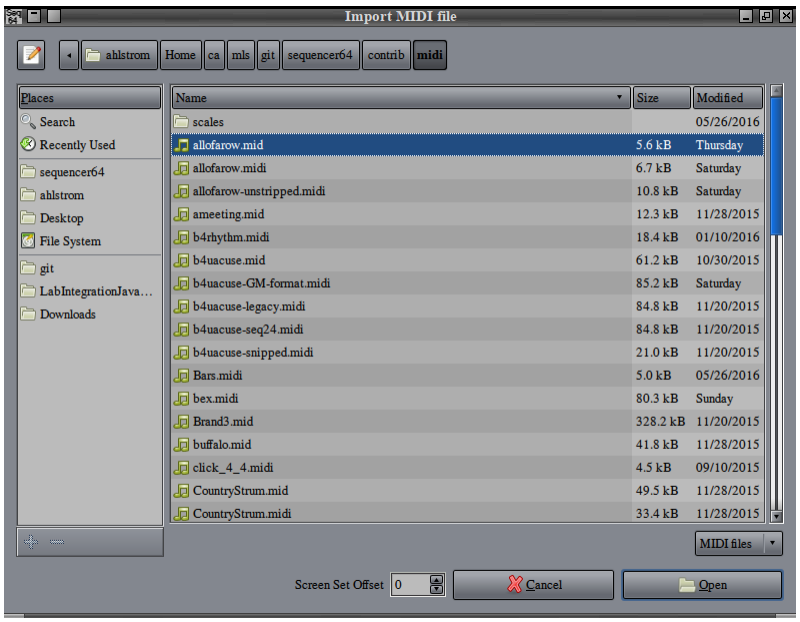



Figure 73: Import MIDI

When imported, each track, whether music or information, is entered into its own loop/pattern box (slot). The import operation can handle reasonably complex files.

Note the additional file-dialog field, **Select Screen Offset**. This setting is actually a set offset. It lets one place the imported data into a screen-set other than the first screen-set (0). This field is not editable. It requires using the scroll button to move the screen set offset up or down in value, from 0 to 31. This field is present only in Gtkmm, but not Qt; in Qt, the current selected screen-set/bank is where the import is loaded, and the patterns are appended after any existing patterns.

When the file is imported, the sequence number for each track is adjusted to put the track into the desired screen-set. The import can place the imported data into any of the 32 available screen-sets. Quite large songs can be built by importing patterns.

Import also handles SMF 0 MIDI files. It parcels out the SMF 0 data into sequences/patterns for each of the 16 MIDI channels. It also puts all of the MIDI data into the 17th pattern (pattern 16), in case it is needed. Note that this slot is used no matter which screen-set one imports the file into. Bug, or feature?

## 7.2 Export Song as MIDI

Thanks to the *Seq32* project, the ability to export songs to MIDI format has been added. "But wait!", you say, "Sequencer64 already saves to a MIDI-compatible format. Why the need for an Export operation?" Well, the **Export Song as MIDI** operation modifies the song in the following ways:

- Only tracks (sequences, loops, or patterns) that are "exportable" are written. To be exportable, a track must be unmuted, and it must have triggers (see section 16.1.22 "Concepts / Terms / trigger" on page 164). That is, the track must have a layout entry in the **Song Editor**.
- All triggers for a given track are consolidated. Each trigger generates the events, including repeats and offset-play of the events. If there is a gap in the layout (e.g. due to an **Expand** operation

in the Song Editor), then the corresponding gap in the events is exported. The result is a track that reconstructs the original playback/performance layout of that pattern. The events themselves are sufficient to play the performance exactly. The triggers are useful for further editing of the song/performance, so they are preserved (in a **SeqSpec** section).

- Empty pattern slots between tracks are removed.
- No matter what set the original track was in, it ends up in the first set.
- Other additions, such as time signature and tempo meta events, are written in the same manner as for a normal **File / Save** operation.

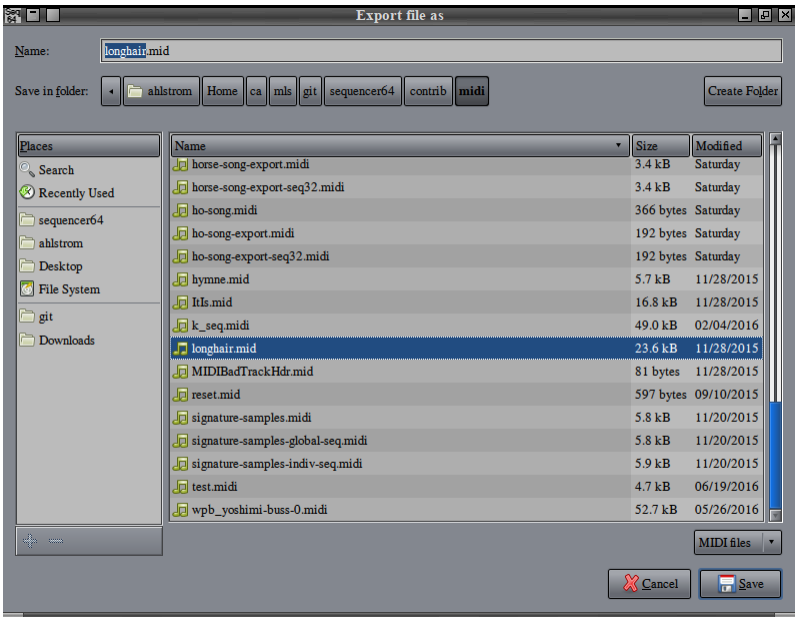



Figure 74: File / Export Song as MIDI

If there are no exportable tracks, the following message is shown:

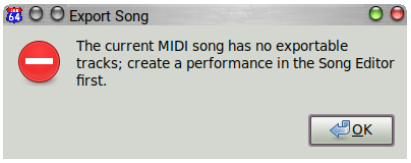



Figure 75: Unexportable MIDI File

Once the file is exported, reopen it to see the results of the export. Both the main window and the song performance window show the results of the export. Here is a short song, in two sets, shown in a composite view of four windows, showing each set and the performance layout of the tracks in the sets. (Note the second Song Editor window.)

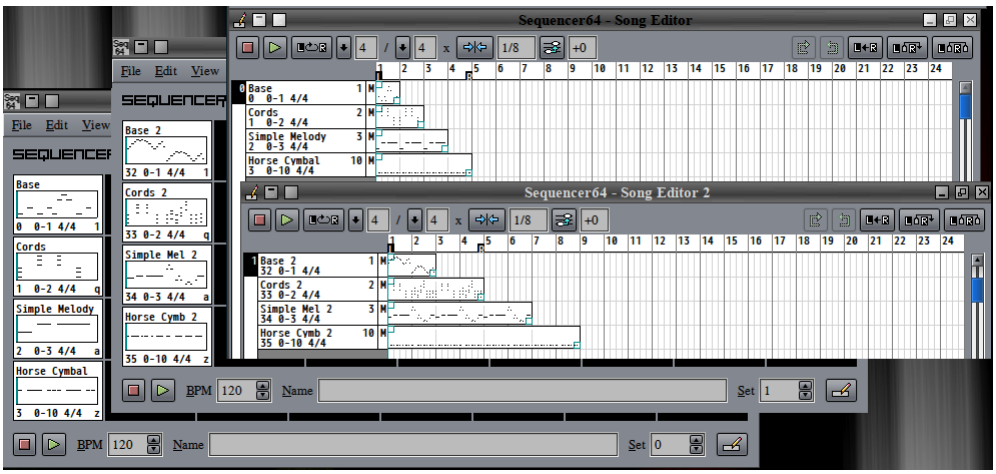



Figure 76: Composite View of an Exportable Song

The left column shows the four tracks of the first set (Set 0), the next column shows the four tracks of the second set (Set 1), and the layouts of the two sets are shown in the remainder of the diagram. Export this song, and see the result:

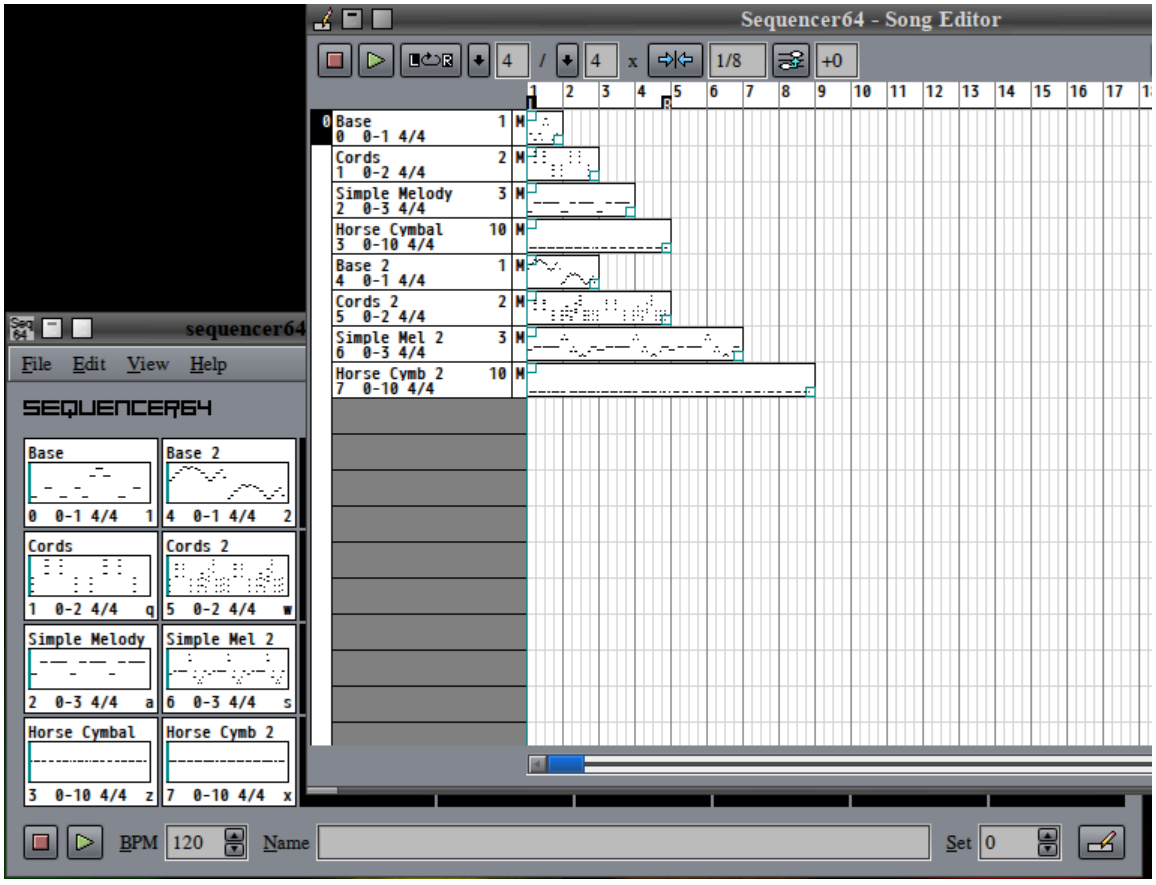



Figure 77: Composite View of Exported Song

The two sets are combined into the first set, and all of the track layouts (triggers) have been exported.

Had there been gaps in layouts or repeats of layouts in the song/performance data, these would have been reflected in the triggers. Much more complex examples are possible.

### 7.3 Export MIDI Only

Sometimes it might be useful to export only the non-sequencer-specific (non-SeqSpec) data from a *Sequencer64* song. For example, some buggy sequencers (hello *Windows Media Player*) might balk at some SeqSpec item in the song, and refuse to load the MIDI file. For such cases, the **Export MIDI Only** menu item writes a file that does not contain the SeqSpec data for each track, and does not include all the SeqSpec data (such as mute groups) that is normally written to the end of the *Sequencer64* MIDI file. Note that the meta data for track name is still written.

## 8 Sequencer64 Keyboard and Mouse Actions

This section presents some tables summarizing keyboard and mouse actions available in *Sequencer64*. It does not cover mute keys and group keys, which are well described in the keyboard options for the main window. See section 2.2.8.3 "Menu / File / Options / Keyboard" on page 23). It does not cover the "fruity" mouse actions, though they are touched on in section 2.2.8.5 "Menu / File / Options / Mouse" on page 28.

This section describes the keystrokes that are currently hardwired in *Sequencer64*. This description only includes items not defined in the **File / Options** dialog. That is, hardwired values. "KP" stands for "keypad". The effect that keystrokes have depends upon which window has the keyboard/mouse focus. It must be noted that the Qt 5 user-interface does not (yet) support the full set of keystrokes supported by the legacy Gtkmm-2.4 user-interface.

### 8.1 Main Window

The main window keystrokes are all defined via the options dialog and "rc" configuration file, or are stock Gtk window-management keystrokes. The main window has a very complete setup for live control of the MIDI tune via keystrokes. These actions are not included in table 1 "Main Window Support" on page 94.

Table 1: Main Window Support

| **Action** | **Normal** | **Double** | **Shift** | **Ctrl** | **Mod4** |
|---|---|---|---|---|---|
| **e** | — | — | — | Open song editor | — |
| **l** (el) | — | — | — | Enter Learn mode | — |
| Left-click slot | Mute/Unmute | New/Edit | Toggle other slots | — | — |
| Right-click slot | Edit menu | — | Edit menu | Edit Menu | — |

The new mouse features of this window for *Sequencer64*, as noted in section 3 "Patterns Panel" on page 35, are:

- *Shift-left-click*: Over one pattern slot, this action toggles the mute/unmute (armed/unarmed) status of all other patterns (even the patterns in other, unseen sets).
- *Left-double-click*: Over a pattern slot, this action quickly toggles the mute/unmute status, which is confusing. But it ultimately brings up the pattern editor (sequence editor) for that pattern.

## 8.2 Performance Editor Window

The "performance editor" window is also known as the "song editor" window. It's main sections are the "piano roll" (perfroll) and the "performance time" (perftime) sections, discussed in the following sections. Also, some keystrokes are handled by the frame of the window.

- `Ctrl-z`. Undo.
- `Ctrl-r`. Redo.

### 8.2.1 Performance Editor Piano Roll

Note that the keystrokes in this table (see table 2 "Performance Window Piano Roll" on page 95) require that the focus first be assigned to the piano roll by left-clicking in an empty area within it. Otherwise, another section of the performance editor might receive the keystroke.

Table 2: Performance Window Piano Roll

| **Action** | **Normal** | **Double** | **Shift** | **Ctrl** | **Mod4** |
|---|---|---|---|---|---|
| Space | Start playback | — | — | — | — |
| Esc | Stop playback | — | — | — | — |
| Period (.) | Pause playback | — | — | — | — |
| Del | Cut section | — | — | — | — |
| c key | — | — | — | Copy | — |
| p key | Paint mode | — | — | — | — |
| v key | — | — | — | Paste | — |
| x key | Escape paint | — | — | Cut | — |
| z key | Zoom out | — | — | Undo | — |
| 0 key | Reset zoom | — | — | — | — |
| Z key | Zoom in | — | — | Undo | — |
| Left-arrow | Move earlier | — | — | — | — |
| Right-arrow | Move later | — | — | — | — |
| Left-click | Select section | — | — | — | — |
| Right-click | Paint mode | — | Paint mode | Paint mode | Lock Paint mode |
| Scroll-up | Scroll up | — | Scroll Left | Scroll Up | — |
| Scroll-down | Scroll down | — | Scroll Right | Scroll Down | — |

This section of the performance editor also handles the start, stop, and pause keys. These can be modified in the **Options / Keyboard** page. A "section" in the performance editor is actually a box that specifies a trigger for the pattern in that sequence/pattern slot. Note that the "toggle other slots" action occurs only if shift-left-clicked in the "names" area of the performance editor. Left-click is used to select performance blocks if clicked within a block, or to deselect them if clicked in an empty area of the piano roll. Also note that all scrolling is done by the internal horizontal and vertical step increments. Some features of this window for *Sequencer64*, as noted in section 5 "Song Editor" on page 77, are explained here:

- *p*: Enters the paint mode, until right-click is pressed or until the "x" key is pressed.
- *x*: Exits the paint mode. Think of the made-up term "x-scape".

- *z*: Zooms out the performance view. It makes the view look smaller, so that more of the performance can be seen. Opening a second performance view is another way to see more of the performance.
- *0*: Resets the zoom to its normal value.
- *Z*: Zooms in the performance view, making the view larger, so that more details of the performance can be seen.
- *Left Arrow*: Moves the selected item to the left (earlier in time) in the performance layout.
- *Right Arrow*: Moves the selected item to the right (later in time) in the performance layout.
- *Mod4-right-click, release*: Locks the paint mode, until right-click is pressed.
- Once selected (rendered in grey), a pattern section (trigger) can be moved by the mouse. To move it using the left or right arrow keys, the paint mode must be entered, but only via the "p" key.

### 8.2.2 Performance Editor Time Section

- `l`. Set to move L marker.
- `r`. Set to move R marker.
- `x`. Escape ("x-scape") the movement mode.
- `Left`. Move the selected marker left.
- `Right`. Move the selected marker right.

This section of the performance editor is also known as the "measure ruler" or the "bar indicator", and is discussed in section 5.2.3 "Song Editor / Arrangement Panel / Measures Ruler" on page 85. See table 3 "Performance Editor Time Section" on page 96.

Table 3: Performance Editor Time Section

| Action | Normal | Double | Shift | Ctrl | Mod4 |
|---|---|---|---|---|---|
| l | Move L [1] | — | — | — | — |
| r | Move R [1] | — | — | — | — |
| x | Escape Move | — | — | — | — |
| Left-Click | Set L [2] | — | — | — | — |
| Middle-Click | — | — | — | — | — |
| Right-Click | Set R [2] | — | — | — | — |

1. Activates movement of this marker using the left and right arrow keys. Movement is in increments of the snap value. This mode is exited by pressing the 'x' key. Also see note [2].
2. Controlled in the pertime section.

The new features of this window for *Sequencer64*, as noted in section 5.2.3 "Song Editor / Arrangement Panel / Measures Ruler" on page 85, are:

- *l*: Enters a mode where the left and right arrow keys move the L marker, until the "x" key is pressed.
- *r*: Enters a mode where the left and right arrow keys move the R marker, until the "x" key is pressed.
- *x*: Exits the marker-movement mode.

### 8.2.3 Performance Editor Names Section

Table 4: Performance Editor Names Section

| Action | Normal | Double | Shift | Ctrl | Mod4 |
|---|---|---|---|---|---|
| Left-Click | Toggle track | — | Toggle other tracks | — | — |
| Middle-Click | — | — | — | — | — |
| Right-Click | New/Edit menu | — | — | — | — |

## 8.3 Pattern Editor

The pattern/sequencer editor piano roll is a complex and powerful event editor; table 5 "Pattern Editor Piano Roll" on page 98, doesn't begin to cover its functionality. Here are some keystrokes handled by the main frame of the piano roll:

- `Ctrl-L`. Bring up the LFO event modulation editor.
- `Ctrl-W`. Exit the sequence (pattern) editor.
- `Ctrl-Page Up`. Zoom in.
- `Ctrl-Page Down`. Zoom out.
- `Shift-Page Up`. Scroll leftward.
- `Shift-Page Down`. Scroll rightward.
- `Shift-Home`. Scroll leftward to the beginning.
- `Shift-End`. Scroll rightward to the end.
- `Page Down`. Scroll downward.
- `Page Up`. Scroll upward.
- `Home`. Scroll upward to the beginning.
- `End`. Scroll downward to the end.
- `Delete`. Deletes (not cuts) the currently-selected notes in the piano roll; can be undone with the **Undo** button.

### 8.3.1 Pattern Editor Piano Roll

Here are the keystrokes handled by the piano roll: These keystrokes require that the focus be set to the piano roll by clicking in it with the mouse.

- `Ctrl-r`. Redo.
- `Ctrl-a`. Select all.
- `Ctrl-Left`. Shrink selected notes.
- `Ctrl-Right`. Grow selected notes.
- `Delete`. Remove selected notes.
- `Backspade`. Remove selected notes.
- `Home. Set sequence to beginnging of sequence`. (Verify!)
- `Enter, Return`. Paste the selected notes at the current position.

And here is the table, which includes items not described above:

Table 5: Pattern Editor Piano Roll

| Action | Normal | Double | Shift | Ctrl | Mod4 |
|---|---|---|---|---|---|
| Del | Delete Selected | — | — | — | — |
| c | — | — | — | Copy | — |
| p | Paint mode | — | — | — | — |
| v | — | — | — | Paste | — |
| x | Escape Paint | — | — | Cut | — |
| z | Zoom Out | — | Zoom In | Undo | — |
| 0 | Reset Zoom | — | — | — | — |
| Left-Arrow | Move Earlier [1] | — | — | — | — |
| Right-Arrow | Move Later [1] | — | — | — | — |
| Up-Arrow | Increase Pitch | — | — | — | — |
| Down-Arrow | Decrease Pitch | — | — | — | — |
| Left-Click | Deselect | — | — | — | — |
| Right-Click | Paint mode | — | Edit Menu | Edit/Edit Menu | Lock Paint mode |
| Left-Middle-Click | Grow Selected | — | Stretch Sel. | — | — |
| Scroll-Up | Zoom Time In | — | Scroll Left | Zoom Time In | — |
| Scroll-Down | Zoom Time Out | — | Scroll Right | Zoom Time Out | — |

1. Once selected (and thus rendered in grey), a pattern segment can be moved by the mouse. To move it using the left or right arrow keys, the paint mode must be entered, but only via the `p` key – the right mouse button deselects the greyed pattern. Too tricky, we might try fixing it later.

Features of this window section for *Sequencer64*, as noted in section 4.3.1 "Pattern Editor / Piano Roll Items" on page 67, are:

- *p*: Enters the paint mode, until right-click is pressed or until the `x` key is pressed. Notes are added by clicking or click-dragging.
- *x*: Exits ("x-scapes") the paint mode.
- *z*: Zooms out.
- *0*: Resets zoom to its normal value.
- *Z*: Zooms in.
- *.*: The period (configurable) does the pause function.
- *Left Arrow*: Moves selected events to the left.
- *Right Arrow*: Moves selected events to the right.
- *Up Arrow*: Moves selected notes upward in pitch.
- *Down Arrow*: Moves selected notes downward in pitch.
- *Mod4-Right-Click*: Locks the paint mode, until right-click is pressed again.

### 8.3.2 Pattern Editor Event Panel

- `Ctrl-x`. Cut.
- `Ctrl-c`. Copy.
- `Ctrl-v`. Paste.
- `Ctrl-z`. Undo.
- `Delete`. Delete (not cut!) the selected events.

- `p`. Enter "paint" (also known as "adding") mode.
- `x`. Escape ("x-scape") the paint mode.

### 8.3.3 Pattern Editor Data Panel

Currently, no keystroke support is provided in the data panel. One potential upgrade would be the ability to change the value of the event with the Up and Down arrow keys.

### 8.3.4 Pattern Editor Virtual Keyboard

Table 6: Pattern Editor Virtual Keyboard

| Action | Normal | Double | Shift | Ctrl | Mod4 |
|---|---|---|---|---|---|
| Left-Click | Play note | — | — | — | — |
| Right-Click | Toggle labels | — | — | — | — |

## 8.4 Event Editor

- `Down`. Move one slot down.
- `Up`. Move one slot up.
- `Page Down`. Move one frame down.
- `Page Up`. Move one frame up.
- `Home`. Move to top frame.
- `End`. Move to bottom frame.
- `Asterisk, KP Multiply`. Delete the currently-selected event.

# 9 Sequencer64 Meta Event / SysEx Support

*Sequencer64* attempts better support for MIDI Meta and System Exclusive events and a Tempo track. It supports the display of Set Tempo and Time Signature events. They can also be added and edited, in various ways. For example, see section 6 "Event Editor" on page 85. Only the first Time Signature event is used to modify playback. System Exclusive support is also still in progress, but very incomplete. This section consolidates the description of the meta-event support. The following topics apply:

1. Tempo display min/max in "usr" settings.
2. Tempo display in main window.
3. Tempo display in pattern editor.
4. Tempo display in song editor.
5. Tempo and Time signature display and editing in the event editor.

First, we need to note *how* the tempo track is implemented in *Sequencer64*. Rather than make a SeqSpec track for the tempo events, we use the MIDI specification mandate that Tempo events should occur only in the first track. *Sequencer64* treats Set Tempo and Time Signature as full-fledged MIDI events that can be viewed (and later, edited) in the existing user-interface. Notes and other events can occur in the same track.

## 9.1 "usr" BPM Display Settings

*Sequencer64* allows the tempo to range from 1 to 600 BPM (beats per minute). This range is hardwired into the application. To display tempo with a little more granularity, *Sequencer64* provides scaling for the tempo displays. These values are found in the "usr" file:

```
0           # midi_bpm_minimum
360         # midi_bpm_maximum
```

See section 11.4 ""usr" File / User MIDI Settings" on page 137, for more information. This setting can only be made by editing the "usr" file while *Sequencer64* is not running. Note that this setting affects the global BPM setting ("c_bpmtag").

## 9.2 Composite Display of Tempos

The following figure shows a composite picture of the various representations of Set Tempo events.

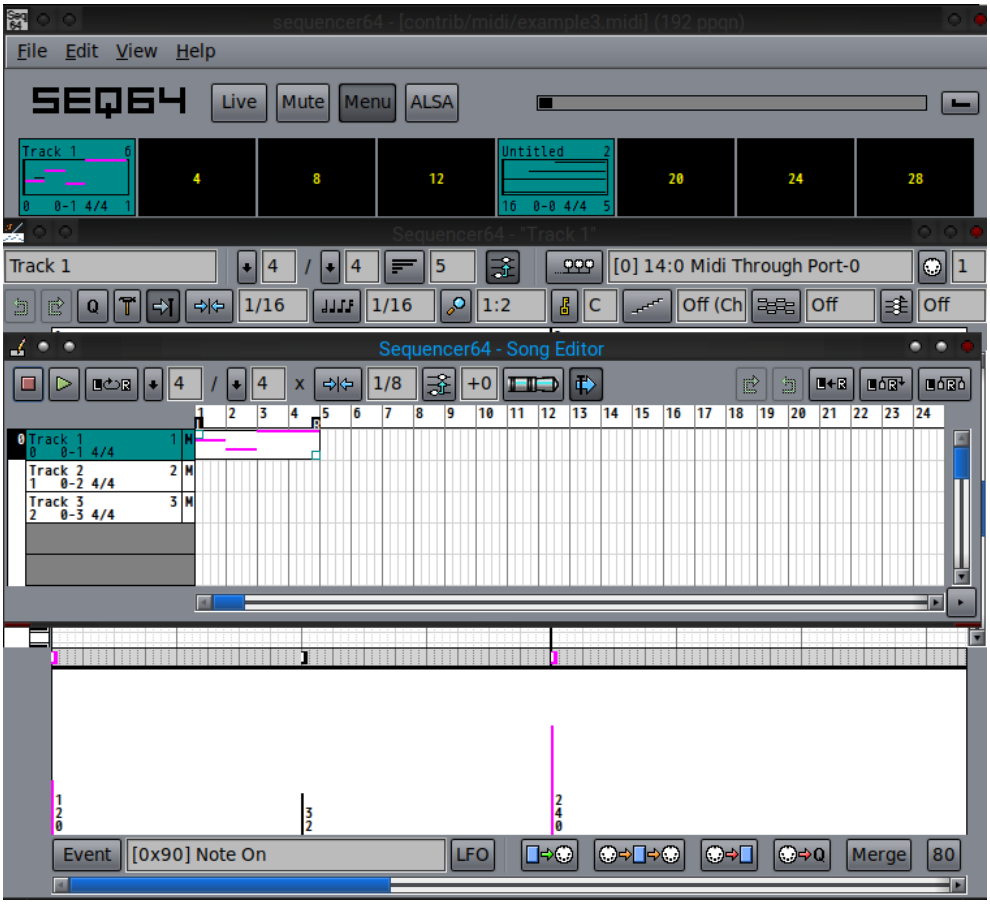



Figure 78: Various Tempo Displays

The *top* of the figure shows the magenta tempo lines in a pattern slot that is currently being edited. This view edited, but the event editor and the main window's BPM settings can be used to add, delete,

or adjust the tempo. The *middle* panel shows the very similar representation of the tempo in the song editor. This view does not allow editing of the tempo events. The *bottom* shows tempo as an event (in the event strip) and a data value in the data pane. A tempo event can be added here by holding the Ctrl key and painting an event in the event strip, and it can then be modified by same method that note velocities can be edited. Tempo events are *always* shown in the event strip and the data pane, no matter what other **Event** type has been selected.

## 9.3 Tempo in the Main Window

The tempo is shown as a solid magenta-colored line at the relative height for the tempo, based on the minimum and maximum values configured in the "usr" file as discussed above. This pattern-slot tempo display is rudimentary. It doesn't allow for ramping of the tempo at present (except by recording while holding the BPM spin-control), and cannot be directly edited in this window. However, tempos can be logged or recorded via magenta-colored controls at the bottom of the main window.


Figure 79: Tempo Recording Controls

The 0th pattern slot shown in the figure is Track 1, the MIDI Tempo track. The magenta lines show the tempos already in that track. Now look at the BPM control. The first button to its right ("0") is the tempo-tap button, used for setting a tempo by tapping in time to music. The light-magenta button that comes next, when pressed while playback is occurring, logs a tempo event at the current progress location and the current BPM value in the BPM spin-field. The dark magenta button to the right of that toggles the mode of recording the changes to the BPM spin-button while playback is occurring.

Although pattern 0 might start out with a length of only a measure or two, the timer continually ticks upward, and tempo events that are recorded after the end of the track at still recorded, and *they will extend the length of the tempo track*. If the "show sequences key" option is enabled, the length of each track, in measures, is shown at the top right of each main window pattern slot, so it can be tracked by the user.

Once tempo events have been recorded, they can be tweaked (or deleted) either in the pattern editor or in the event editor. Generally, they are treated like control events that are always available. Deleting all tempo events will not reduce the (possibly new) length of the sequence. The Tempo track will *not* change tempo unless that track is unmuted. This behavior is a feature, not a bug.

# 10 Sequencer64 "rc" Configuration File

There are two *Sequencer64* configuration files: `sequencer64.rc` and `sequencer64.usr`. See section 11 "Sequencer64 "usr" Configuration File" on page 124; it describes the "usr" file, is handled a bit differently than the "rc" file.

The *Sequencer64* configuration file originally was named `.seq24rc`, and it was stored directly in the user's `$HOME` directory, following the convention of *Seq24*. To avoid interference with one's existing installation of *Seq24*, we created a new file to take its place, with a fall-back to the original file-name if the new file does not exist, or if *Sequencer64* is running in legacy mode. In addition, one can change the configuration directory and the base name of the "rc" and "usr" files from the command-line.

After you run *Sequencer64* for the first time (in non-legacy mode), it will generate a `sequencer64.rc` file in your home *configuration* directory:

```
/home/ahlstrom/.config/sequencer64/sequencer64.rc
```

It contains the the data for remote MIDI control, computer keyboard control, MIDI clock, JACK transport, and a few other settings.

*Sequencer64* will *always* overwrite the `sequencer6.rc` file upon quitting. One must therefore quit *Sequencer64* before making manual modifications to the `sequencer64.rc` file. Note that many of its settings can be modified in the **Options** dialog (see section 2.2.8 "Menu / File / Options" on page 17). There is an old, but complete, example of the *Seq24* "rc" file at [25]. It includes a setup for the Novation Launchpad device.

Now let's covered each section of the "rc" file in order.

## 10.1 "rc" File / Comments

The very top of the "rc" file that *Sequencer64* generates is a stock banner showing the version of *Sequencer64* to which this file applies, the name of the configuration file, and when it was written. The user can also add a comments section that explain what the user's setup is in brief:

```
[comments]

Comments added to this section are preserved.  Lines starting with
a '#' or '[', or that are blank, are ignored.  Start lines that must
be blank with a space.
```

A blank line (not even a space) ends the comment section.

## 10.2 "rc" File / MIDI Control

Like *Seq24*, *Sequencer64* provides a way to control the application to some extent via a MIDI controller, such as a MIDI keyboard or a MIDI pad device. The current section describes this feature; additional resources and ideas can be found at `linuxaudio.org` ([14]).

New with version 0.96 of *Sequencer64* is the ability to offload the MIDI control section to a separate file. Simply move the whole `[midi-control]` section to a separate file in the *Sequencer64* configuration directory, and add the following snippet:

```
[midi-control-file]
   nanomap.rc        # contains a whole [midi-control] section
```

As with the normal "rc", this file is rewritten upon exit, so don't bother trying to add comments to it. The rest of this section applied either to that "rc" or the normal "rc" file.

The MIDI control section begins with the following "INI"-style group marker tag:

```
[midi-control]
   74      # MIDI controls count (74/84/96)
```

The number (74) is the number of lines in the MIDI Control section. The extended automation values bring this number up to 84, and the most recent version of *Sequencer64* brings this up to 96. Note that earlier version will complain about numbers higher than what they can handle. If so, edit the "rc" file to reduce that number.

Even in the latest version of *Sequencer64*, not all of these new values are yet usable, and there are also some values reserved for future expansion. Currently, the "start", "pause", "stop", and "bpm" page controls, and the "performance record", "MIDI THRU", "MIDI RECORD", and "MIDI Quantized RECORD" have been implemented. (To come: playlist control).

Each MIDI control line has the following format:

```
   74     [0 0 0 0 0 0]   [0 0 0 0 0 0]   [0 0 0 0 0 0]
```

The first number is an *internal* control number ranging from 0 to 73, 83, or 95 depending on the version of *Sequencer64*. The three bracketed sections represent MIDI controls to toggle, turn on, and turn off a pattern or mute group. However, some MIDI controls extend the meanings of these brackets for additional functionality.

```
 ------------------ on/off
| ------------------ inverse
| |  -------------- MIDI status (event) byte (e.g. note on)
| | |  ------------ data 1 (e.g. note number)
| | | |  ---------- data 2 min
| | | | |  -------- data 2 max
| | | | | |
```

```
            v v v v v v
  74       [0 0 0 0 0 0]   [0 0 0 0 0 0]   [0 0 0 0 0 0]
Index:      Toggle          On              Off
Playback:   Pause           Start/Play      Stop
Playlist:   By-Value        Next            Previous
```

The first number is an index number, starting at 0. It indicates what function the control line will affect. The numbers in the leftmost brackets define a *toggle* filter; the numbers in the middle brackets define a *on* filter; the numbers in the rightmost brackets define a *off* filter. Additional functions are shown that extend these basic functions, such as changing a selection, controlling playback, or activating a feature.

The numbers inside the brackets define six values that set up the control. The layout of each filter inside the brackets is as follows:

**[OPR INV STAT D1 D2min D2max]**

- **OPR = on/off**
- **INV = inverse**
- **STAT = MIDI status byte** (channel ignored)
- **D1 = data1**
- **D2min = data2 min**
- **D2max = data2 max**

If **OPR (on/off)** is set to 1, it will match the incoming MIDI against the **STAT (MIDI status byte)** pattern. and perform the action (on/off/toggle) if the data falls in the range specified. All values are in decimal.

**Note**: In legacy versions (*Seq24* and early versions of *Sequencer64*), the channel nybble of the MIDI control (and all other incoming MIDI events) were stripped off. This is no longer the case, and thus opens up many more events useful for MIDI control. But do note that events that are actually recorded end up getting the channel number of the pattern into which they are recorded.

The **INV (inverse)** field will make the pattern perform the opposite action (*off* for *on*, *on* for *off*) if the data falls outside the specified range. This is cool because one can map several sequences to a knob or fader.

The **STAT (MIDI status byte)** field is a MIDI status byte number in decimal notation. The channel nybble of this byte is ignored. One can look up the possible status values up in the MIDI messages tables; the relevant data can be found at [15]. As the channel on which the events are sent is ignored, it is sufficient to use the values for channel 1; that is, 0.

The last three fields describe the range of data that will match. The **D1 (data1)** field provides the actual MIDI event message number to detect, in decimal. This item could be a Note On/Off event or a Control/Mode change event, for example.

The **D2min (data2 min)** field is the minimum value of the event for the filter to match. For Note On/Off events, this would be the velocity value, for example.

The **D2max (data2 max)** field is the maximum value of the event for the filter to match.

For each pattern, we can set up MIDI events to turn a pattern on, off, or to toggle it, or to control some other function. If the incoming MIDI event value matches a value present in the filter, it will *toggle*

(first field), *enable* (second field) or *disable* (third field) the sequence, or perform some kind of automation control.

As a quick example, let us set up a Note On event on channel 2 and key value 48 that will increment *Sequencer64*'s BPM value each time it is pressed. A Note On event is 0x90 hex, or 144 decimal. Channel 2 is 0x1 hex or 1 decimal. Adding the Note On value to the the channel number yields 0x91 hex, or 145 decimal. Then we pick a note value of 48, which is one of the C keys. We don't care about the velocity, so we allow all values (0 to 127). We add the following entry to `~/.config/sequencer64/sequencer64.rc`):

```
[midi-control]
 . . .
# bpm up:
64 [0 0   0   0   0   0] [1 0 145  48   0 127] [0 0   0   0   0   0]
```

Now, whenever we press key 48 (C), we see that the BPM value in the main patterns panel increments by 1.

The MIDI control setup resembles a matrix. This matrix is divided into a number of sections depending on the overall functionality of the MIDI controls in the section:

1. **Pattern Group** (rows 0 to 31).
2. **Mute-In Group** (rows 32 to 63).
3. **Automation Group** (rows 64 to 73).
4. **Extended Automation Group** (row 74 to 95).

### 10.2.1 "rc" File / MIDI Control / Pattern Group

The pattern group consists of 32 lines (0 to 31), one for each pattern slot shown in the Pattern window. It provides a way to control the arming/disarming (muting/unmuting) of each pattern shown in the main window. Note that the main window shows the *active* screen-set. These MIDI controls affect the *active* screen-set.

This block of matrix elements, numbered from 0 to 31, represent control functions (toggle, mute, unmute) for the 32 patterns of the active screen-set. These 32 rows correspond to the hot-keys assigned in the **File / Options / Keyboard / Control keys [keyboard-group]** configuration panel.

Here is an example of setting mute/unmute control for the first four patterns of the active screen-set. Note that the numbers are decimal numbers, and that 144 is 90 hex (a Note On event) and 128 is 80 hex (a Note Off event).

```
                          on (enabled)------------
                            |  Note On            | Note Off
                            |    |  Note #        |    | Note #
                            |    |    |   Vel     |    |    |
     --- off (disabled)     |    |    |   Range   |    |    |
    |                       |    |    |   |  |    |    |    |
    v                       v    v    v   v  v    v    v    v
 0 [0 0   0   0   0   0] [1 0 144   0   0 127] [1 0 128   0   0 127]
 1 [0 0   0   0   0   0] [1 0 144  16   0 127] [1 0 128  16   0 127]
```

```
2 [0 0   0   0   0   0] [1 0 144  32   0 127] [1 0 128  32   0 127]
3 [0 0   0   0   0   0] [1 0 144  48   0 127] [1 0 128  48   0 127]
#    Toggle                 On                       Off
```

The "toggle" section is empty and disabled. The "on" section specifies that a Note On event of any velocity will unmute a pattern, and the note number determines which pattern. A Note Off of any velocity will mute a pattern. These settings are for the control grid of a Novation Launchpad.

Look at line "0. The first number, 0, indicates the first pattern (pattern numbering starts from 0). The first section, **Toggle**, is off (inactive). All values are 0. There is no setup to use MIDI control to toggle pattern 1 here.

On to the second section, **On**:

- The **On** section starts with **OPR** = 1, so it is on (1 = active).
- The **inverse** value is off (0 = inactive).
- The **MIDI status byte**, 144, which is 0x90 (hex), which is a Note On event on channel 0. However, the channel is ignored.
- The **data1** values sets the actual Note value to 0, meaning the lowest possible MIDI note (pitch) value.
- **data2 min** value sets the minimum value to 0.
- **data2 max** sets the maximum value to 127.

Thus, receiving any Note On velocity for note 0 will turn sequence 1 *on*. This is the second pattern; in the default setup, key `q` would operate on this pattern as well.

On to the **Off** section:

- The **Off** field is on (active).
- The **inverse** value is off (0 = inactive).
- The **MIDI status byte**, 128, which is 0x80 (hex), which 128, which is 0x80 (hex), which is a Note Off event on channel 0.
- The **data1** values sets the actual Note value to 0, meaning the lowest possible MIDI note (pitch) value.
- **data2 min** value sets the minimum value to 0.
- **data2 max** sets the maximum value to 127.

Thus, receiving any Note Off velocity for note 0 will turn sequence 1 *off*.

So, basically, pattern 1 starts when any Note On for MIDI note 0 is received, and it stops when any Note Off for MIDI note 0 is received. One can easily extend this so that Note On/Off values from 0 to 31 control the corresponding pattern slot.

Obviously, one might not want Note On/Off events from any channel to trigger events, so some other event would likely be more useful. Here's a little table of the decimal numbers for some commonly-used MIDI controls:

- **128** or **129** for any Note On or Note Off events.
- **160** Polyphonic aftertouch.

- **176** Control Change event.
- **192** Program change.
- **208** Aftertouch.
- **224** Pitch wheel.

The following example would map a row of sequences to one knob sending out changes for Control Code 1:

```
#    Toggle                 On                        Off
0 [0 0 0 0 0 0]      [1 1 176 1   0   15]      [0 0 0 0 0 0]
1 [0 0 0 0 0 0]      [1 1 176 1  16   31]      [0 0 0 0 0 0]
2 [0 0 0 0 0 0]      [1 1 176 1  32   47]      [0 0 0 0 0 0]
3 [0 0 0 0 0 0]      [1 1 176 1  48   63]      [0 0 0 0 0 0]
4 [0 0 0 0 0 0]      [1 1 176 1  64   79]      [0 0 0 0 0 0]
5 [0 0 0 0 0 0]      [1 1 176 1  80   95]      [0 0 0 0 0 0]
6 [0 0 0 0 0 0]      [1 1 176 1  96  111]      [0 0 0 0 0 0]
7 [0 0 0 0 0 0]      [1 1 176 1 112  127]      [0 0 0 0 0 0]
```

The **on** field is on (active). Inverse is active. The **MIDI status byte**, 176, is 0xB0 (hex), which is a Control Change event (channel ignored). **data1** is 1, which is the controller number for a Modulation Wheel. The **data2** ranges are set so that, as the controller data increases (as the modulation-wheel knob is turned, so to speak), patterns 0 through 7 come on one at a time until all are running.

### 10.2.2 "rc" File / MIDI Control / Pattern Group Multiples

This section describes a feature of the pattern-group that needs it own section for emphasis. This section describes using a single MIDI control to control a number of operations at one time. Essentially, if a particular MIDI control row is repeated, each repetition has its own effect on the patterns, which permits one MIDI control event to control multiple patterns at once.

This control can be used, for example, to emulate the *Ableton Live row control* functionality. Here is a sample that uses the lowest range of MIDI notes to control the muting and unmuting of patterns:

```
# Pattern-group section:
0 [0 0 0 0 0 0]   [1 0 144  0   0 127]  [1 0 128    0   0 127]
1 [0 0 0 0 0 0]   [1 0 144  1   0 127]  [1 0 128    1   0 127]
2 [0 0 0 0 0 0]   [1 0 144  2   0 127]  [1 0 128    2   0 127]
3 [0 0 0 0 0 0]   [1 0 144  3   0 127]  [1 0 128    3   0 127]
4 [0 0 0 0 0 0]   [1 0 144  0   0 127]  [1 0 128    0   0 127]
5 [0 0 0 0 0 0]   [1 0 144  1   0 127]  [1 0 128    1   0 127]
6 [0 0 0 0 0 0]   [1 0 144  2   0 127]  [1 0 128    2   0 127]
7 [0 0 0 0 0 0]   [1 0 144  3   0 127]  [1 0 128    3   0 127]
8 [0 0 0 0 0 0]   [1 0 144  0   0 127]  [1 0 128    0   0 127]
9 [0 0 0 0 0 0]   [1 0 144  1   0 127]  [1 0 128    1   0 127]
10 [0 0 0 0 0 0]  [1 0 144  2   0 127]  [1 0 128    2   0 127]
11 [0 0 0 0 0 0]  [1 0 144  3   0 127]  [1 0 128    3   0 127]
12 [0 0 0 0 0 0]  [1 0 144  0   0 127]  [1 0 128    0   0 127]
```

```
13 [0 0 0 0 0 0]  [1 0 144  1   0 127]  [1 0 128    1   0 127]
14 [0 0 0 0 0 0]  [1 0 144  2   0 127]  [1 0 128    2   0 127]
15 [0 0 0 0 0 0]  [1 0 144  3   0 127]  [1 0 128    3   0 127]
```

Observer that MIDI On (144) and Off (128) events appear four times for each note value of 0, 1, 2, and 3. Each note value thus controls four patterns – one whole row in a 4x8 pattern. When note 0 is pressed, patterns 0, 4, 8, and 12 turn on. When note 0 is released, they turn off.

### 10.2.3 "rc" File / MIDI Control / Mute-In Group

This section controls 32 groups of mutes. A group is a set of patterns that can toggle their playing state together. Every group contains all 32 sequences in the active screen set. So, this part of the MIDI Control section is used for muting and unmuting (and toggling) a group of patterns.

The mute-in group onsists of 32 lines (32 to 63), one for each pattern box shown in the Pattern window. It provides a way to control the mute groups. A group is a set of sequences that can arm their playing state together; every group contains all 32 sequences in the *active* screen-set.

These 32 values represent the same actions as the the **File / Options / Keyboard / Mute-group slot [mute-group]** configuration panel. See section 2.2.8.3 "Menu / File / Options / Keyboard" on page 23.

What is the different between the **mute-in group** section and the **mute group** section? The former defines the MIDI control values that can affect the muting of a group, while the latter specifies the armed patterns that are part of a group.

### 10.2.4 "rc" File / MIDI Control / Automation

The automation control keys occupy the entries from 64 to 73. These entries control *Sequencer64* actions like changing the BPM value, screen-set, etc. *Sequencer64* adds some more entries, from 74 to 83, to control additional *Sequencer64* functions such as performance record, solo, etc.

One issue with this group is that the original control functions are a bit wasteful. For example there are two control lines for BPM: BPM up and BPM down. These could have been combined into one line, with the "on" group meaning "up", and the "off" group meaning "down". *Sequencer64* at present does not change this setup, to avoid breaking existing MIDI control sections.

Each item in this group consists of one line. Each line specifies a MIDI event that can cause a given *Sequencer64* user-interface operation to occur. Here are the original automation-group settings:

1. **bpm up**
2. **bpm down**
3. **screen-set up**
4. **screen-set down**
5. **mod replace**
6. **mod snapshot**
7. **mod queue**
8. **mod gmute**
9. **mod glearn**
10. **screen-set play**

#### 10.2.4.1 Automation / BPM Up and Down

The BPM Up MIDI control increments the beats-per-minute setting, as if the up-arrow has been clicked, or the up-arrow key pressed, in the BPM user-interface control. This increment is the "step increment" which defaults to 1, but can be modified by changing the "bpm_step_increment" value in the "usr" configuration file. See section 11.4 ""usr" File / User MIDI Settings" on page 137.

Similarly, the BPM Down MIDI control decrements the beats-per-minute setting, as if the down-arrow has been clicked/pressed.

Here is a sample group for changing the BPM in both directions. The value 176 is B0 hex, which is a Control Change event. 104 is 68 hex, and represents an undefined control. 105 is 69 hex, and is also undefined. So both MIDI events can be used without interfering with playback.

Question: Why are both the "on" and "off" sections defined?

```
# bpm up
64 [0 0   0   0   0   0] [1 0 176 104 127 127] [1 0 176 104 127 127]
# bpm down
65 [0 0   0   0   0   0] [1 0 176 105 127 127] [1 0 176 105 127 127]
```

Note that these controls correspond to the hot-keys assigned in the **File / Options / Keyboard / Control keys [keyboard-group]** "BPM Up" and "BPM Down" configuration panel items.

#### 10.2.4.2 Automation / Screen-Set Up and Down

The Screen-Set Up MIDI control increments to the next screen-set. Once the screen-set has been altered, mute-groups and other actions apply to that screen set.

Similarly, the Screen-Set Up MIDI control decrements to the previous screen-set.

```
# bpm up
66 [0 0   0   0   0   0] [1 0 176 104 127 127] [1 0 176 104 127 127]
# bpm down
67 [0 0   0   0   0   0] [1 0 176 105 127 127] [1 0 176 105 127 127]
```

Note that these controls correspond to the hot-keys assigned in the **File / Options / Keyboard / Control keys [keyboard-group]** "BPM Up" and "BPM Down" configuration panel items.

#### 10.2.4.3 Automation / Mod Replace

The Mod Replace MIDI control sets the "replace" status flag. Then, when the user manually clicks a pattern slot, that pattern is unmuted, and all the rest are muted. Thus, this MIDI control is kind a of "Solo" function. It works whether in "Live" or "Song" mode.

Note that this control corresponds to the hot-key assigned in the **File / Options / Keyboard / Control keys [keyboard-group]** "Replace/Solo" configuration panel item.

#### 10.2.4.4 Automation / Mod Snaphot

The Mod Snapshot MIDI control causes the playing statuses of all active (i.e. having data) patterns to be saved. When turned off, the original playing status is restored. Thus, two MIDI events need to be allocated to this functionality. Compare to section section 3.2.3.1 "Pattern Keys" on page 49for a better idea of how it works.

Note that this control corresponds to the hot-keys assigned in the **File / Options / Keyboard / Control keys [keyboard-group]** "Snapshot 1" and "Snapshot 2" configuration panel items.

#### 10.2.4.5 Automation / Mod Queue

The Mod Queue MIDI control sets up the "queue" status flag. Then, when the user manually clicks a pattern slot, that pattern is queued, and will play at the next cycle of the pattern.

Here is an example from [14], which shows how to set up the "Sustain" control-change event to queue or un-queue a sequence: The *Akai MPK Mini* has a Sustain button and we can set the Sustain MIDI event (with MIDI status byte 176 [0xB0] to represent a Controller event, and control/mode change number 64 [0x40] to represent the Sustain or Pedal control) up as the queue modifier in the `mod queue` entry:

```
# mod queue
#    Toggle                  On                        Off
70 [0   0   0   0   0   0  ] [1   0   176 64 127 127] [1   0  176 64  0  0]
#   OPR INV STA D1  mn mx     OPR INV STA D1 mn  mx   OPR INV STA D1  mn mx
#                                       ^  ^                    ^  ^
#                                       |  |                    |  |
#                                       |   ----Sustain---------|--
#                                        -------Control Change--
```

So when the Sustain button is held down, and one presses one of the pads on the *MPK Mini*, the corresponding sequence gets queued.

Note that this control corresponds to the hot-key assigned in the **File / Options / Keyboard / Control keys [keyboard-group]** "Queue" configuration panel item. There is also a "Q" button on the Gtkmm-2.4 user-interface.

#### 10.2.4.6 Automation / Mod Mute Group

The Mod Group Mute MIDI control sets up a "mute group". More to come on this one, see section 3.1 "Patterns / Top Panel" on page 36.

This control sets an internal "mode-group" flag on.

When activated, *Sequencer64* loops through all of the 32 screen-sets, and through the active pattern in each screen-set, and sets each pattern as muted or unmuted, depending on the saved mute state.

#### 10.2.4.7 Automation / Mod Mute Group

The Mod Group Learn MIDI control sets up a "group learn". However, as the group-learn key is a modifier key that needs to be held, we're not quite sure how this works with MIDI control.

This control sets two internal flags on : "mode-group" and "group-learn". The first flag indicates that we will be handling mute-groups. The second flag indicates that we are learning these mute-groups, effectively recording the current status of all the patterns in all of the screen-sets.

These statuses ultimately get saved in the `[mute-group]` section of the "rc" file.

Note that this control corresponds to the "L" button in the main window user-interface. It can also be accessed by the hard-wired hot-key, Ctrl-L.

#### 10.2.4.8 Automation / Screen-Set Play

This MIDI control sets the playing screen-set, but we're not completely sure how it works yet.

### 10.2.5 "rc" File / MIDI Control / Extended Automation

These additional control items were requested by users, to control additional features of the application. Each item in this group consists of one line. The following sections show how to set up extended automation controls.

1. **Stop/Pause/Start**. Emulate the Stop, Pause, and Start keys, using Toggle for pause, Off for stop, and On for start.
2. **Record**. For recording a live performance by recording the mute/unmute states that the musician played. Not yet functional, thought the feature has been added as of version 0.94.
3. **Solo on/off**. Not yet functional. See the "Mod Replace" functionality.
4. **Thru toggle**. Not yet functional.
5. **Reserved for expansion** (a few of these are reserved).

#### 10.2.5.1 Ext Automation / Stop/Pause/Start

Here, we will set up *Sequencer64* so that the first three MIDI white keys (notes 0, 2, and 4) will become "Stop", "Pause", and "Start" buttons.

The first step, if not already done, is to install the newest version (**0.96.0** and above) of *Sequencer64*, run it, and then exit. Verify in the regenerated "rc" file (`~/.config/sequencer64/sequencer64.rc`) that the following line exists:

```
96      # MIDI controls count (74/84/96)
```

Then go to line 74, which is the Stop/Pause/Play MIDI control. Replace it with:

```
# start playback (pause, start, stop):
74 [1 0 144   2   0 127] [1 0 144   4   0 127] [1 0 144   0   0 127]
```

This sets up MIDI Note On (144) with note numbers 2 (toggle/pause), 4 (on/start), and 0 (off/stop). Any velocity (0 to 127) will work to trigger these actions. Now we are ready to test this feature. One can use a MIDI keyboard to do so, but here we will use the *VMPK* ([34]) virtual MIDI piano keyboard application for this test. Refer to the figure below. In addition, let's set up the standard and the new, extended, BPM (beats/minute) MIDI control values.

```
# bpm up: Note On 9
64 [0 0   0   0   0   0] [1 0 144   9   0 127] [0 0   0   0   0   0]
# bpm down: Note On 7
65 [0 0   0   0   0   0] [1 0 144   7   0 127] [0 0   0   0   0   0]
```

The section above sets up the standard (step-size) BPM controls, which correspond to the fine control possible with the up and down arrows of the BPM spinner in the main window. It sets up Note On 7 to be the BPM-down control, and Note On 9 to be the BPM-up control. These two keys are the dark-cyan keys shown in the figure.

```
# bpm page up:  Note On 11
78 [0 0   0   0   0   0] [1 0 144  11   0 127] [0 0   0   0   0   0]
# bpm page down:  Note On 5
79 [0 0   0   0   0   0] [1 0 144  05   0 127] [0 0   0   0   0   0]
```

That section uses the extended controls (74 to 83) set up the coarse (page-size) BPM controls, which correspond to the larger jumps possible with the Page Up and Page Down keys when the BPM spinner has focus. It sets up Note On 5 to be the BPM-page-down control, and Note On 11 to be the BPM-page-up control. These two keys are the bright-cyan keys shown in the figure.

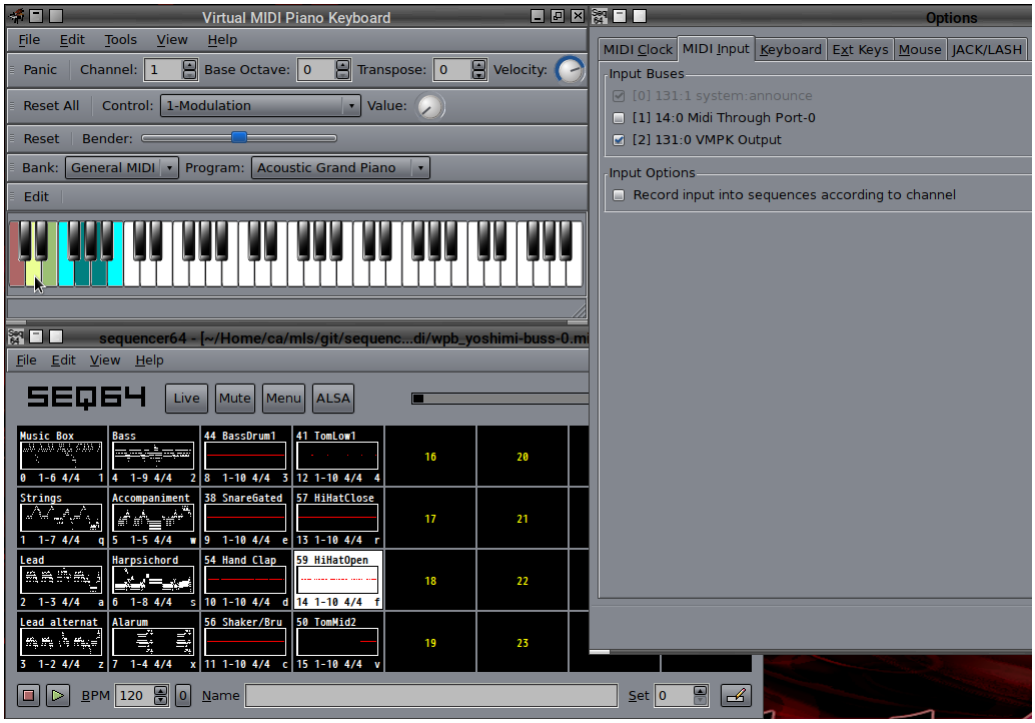



Figure 80: Stop/Pause/Start ALSA Test Setup

One can copy these settings from the sample file `contrib/simple-midi-control-section.rc`, if one wants to try them. It also illustrates some other setups that we use for testing purposes, but are not described here.

Set up *VMPK* to use the lowest octave by setting **Base Octave** to `0`. The red, yellow, and green keys shown will be our stop, pause, and start keys. Also set the velocity to a value ranging from 1 to 127. Do not use a zero velocity, as it seems that VMPK will not transmit Note On messages with a zero velocity.

Next, run *Sequencer64*, using the following command line to make sure that it is using ALSA and using automatic mode to connect the ALSA MID ports:

```
$ seq64 -A -a
```

Then open a MIDI file. Next, open the **File / Options / MIDI Input** tab, and make sure that the **VMPK Output** is check-marked as shown in the figure. If desired, also connect up to some kind of synthesizer so that the song can be heard.

Finally, press the third white key (shown as green in the figure) to start playback. The second white key (yellow in the figure) will pause and resume playback. The first white key (red in the figure) will stop (and rewind) playback.

Then play with the BPM MIDI control keys. Note that the size of the BPM step-increment and the BPM page-increment are configurable in the `[user-midi-settings]` section of the "usr" configuration file, using the following values in that section:

```
1        # bpm_precision
1.0      # bpm_step_increment
10       # bpm_page_increment
```

See section 11.4 ""usr" File / User MIDI Settings" on page 137; it has information about the usage and enabling of these settings.

Obviously, this setup is not useful for performance, but serves as a good example to verify this MIDI control.

One thing we noticed while implementing this functionality is that there is really no need to have two lines for pairs such as BPM up/down and screen-set up/down. Also, is screen-set play now partly redundant? No matter, we will not break the user's existing setup.

#### 10.2.5.2 Ext Automation / Performance Record

To do.

#### 10.2.5.3 Ext Automation / Solo

To do.

#### 10.2.5.4 Ext Automation / MIDI Thru

To do.

#### 10.2.5.5 Ext Automation / BPM Page Up

To do.

#### 10.2.5.6 Ext Automation / BPM Page Down

To do.

#### 10.2.5.7 Ext Automation / Screen-Set By Number

To do.

#### 10.2.5.8 Ext Automation / MIDI Record

To do.

#### 10.2.5.9 Ext Automation / MIDI Quantized Record

To do.

#### 10.2.5.10 Ext Automation / Fast Forward

To do.

#### 10.2.5.11 Ext Automation / Rewind

To do.

#### 10.2.5.12 Ext Automation / Top

To do.

#### 10.2.5.13 Ext Automation / Select Playlist

To do.

#### 10.2.5.14 Ext Automation / Select Song

To do.

## 10.3 "rc" File / Mute-Group Section

This section is delimited by the `[mute-group]` construct. It controls 32 groups of mutes in the same way as defined for `[midi-control]`. A group is set of sequences that can toggle their playing state together. Every group contains all 32 sequences in the active screen set.

```
[mute-group]
1024    # group mute value count
0 [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0]
1 [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0]
2 [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0]
...      ...               ...               ...               ...
31 [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0]
```

The initial number, 1024 is probably the total count of 32 x 32 sequences. In this group are the definitions of the state of the 32 sequences in the playing screen set when a group is selected. Each set of brackets defines a group:

```
[state of the first 8 sequences] [second 8] [third 8] [fourth 8]
```

After the list of sequences and their MIDI events, one can set *Sequencer64* to handle MIDI events and change some more settings in `sequencer64.rc`.

What is the different between the **mute-in group** section and the **mute group** section? The former defines the MIDI control values that can affect the muting of a group, while the latter specifies the patterns that are part of a group.

## 10.4 "rc" File / MIDI-Clock Section

The MIDI Clock fields will contain the clocking state from the last time *Sequencer64* was run. Turn off the clock with a 0, or on with a 1 (which means to send MIDI Song Position, and MIDI Continue if starting after tick 0), or on with positioning with a 2, which sends MIDI Start and then bbegins clocking after the position reaches a modulo of the **Clock Start Modulo value)**. Luckily, the user-interface makes it easy to select the desire value, and has tool-tips to instruct the user. This section has 16 entries, one for each MIDI output buss that *Sequencer64* supports.

This configuration item is the same as the **MIDI Clock** tab described in paragraph 2.2.8.1 "Menu / File / Options / MIDI Clock" on page 17

Here is the format:

```
[midi-clock]
16
 0 0  #  [1] seq24 1
 1 0  #  [2] seq24 2
 2 0  #  [3] seq24 3
 3 0  #  [4] seq24 4
```

```
 4 0  #  [5] seq24 5
 5 0  #  [6] seq24 6
 6 0  #  [7] seq24 7
 7 0  #  [8] seq24 8
 8 0  #  [9] seq24 9
 9 0  # [10] seq24 10
10 0  # [11] seq24 11
11 0  # [12] seq24 12
12 0  # [13] seq24 13
13 0  # [14] seq24 14
14 0  # [15] seq24 15
15 0  # [16] seq24 16
```

That sample would be written one had started up *Sequencer64* in manual-alsa-mode. On our system, where we have Timidity running, and erroneously have also specified 3 MIDI busses that we do not have, in the `sequencer64.usr` file:

```
[midi-clock]
5    # number of MIDI clocks/busses
# Output buss name: [0] 14:0 2x2 A (SuperNova,Q,TX81Z,DrumStation)
0 0  # buss number, clock status
# Output buss name: [1] 128:0 2x2 B (WaveStation,ESI-2000,MV4,ES-1,ER-1)
1 0  # buss number, clock status
# Output buss name: [2] 128:1 PCR-30 (303)
2 0  # buss number, clock status
# Output buss name: [3] 128:2 TiMidity port 2
3 0  # buss number, clock status
# Output buss name: [4] 128:3 TiMidity port 3
4 0  # buss number, clock status
```

## 10.5 "rc" File / MIDI-Meta-Events Section

The new MIDI Meta events section is the start of additional options supporting meta events as normal events in *Sequencer64*.

```
[midi-meta-events]
10      # tempo_track_number
```

Normally, as per the MIDI specification, the first track (track 1 in track numbering, or pattern 0 in *Sequencer64* numbering) is *the* official track for certain MIDI meta events, such as Set Tempo and Time Signature. However, to accommodate existing tunes and their set arrangement, we allow the user to go into **File / Options / MIDI Clock** and change the tempo track to another pattern.

Please note that the user can insert Set Tempo events into any track via the pattern editor or the event editor. But, when recording tempo events, they will always be written to the patten having the

tempo-track number.

## 10.6 "rc" File / Keyboard Control Section

The keyboard control is a dump of the keys that *Sequencer64* recognises, and each key's corresponding sequence number. Note that the first number corresponds to the number of sequences in the active screen set.

```
[keyboard-control]
32     # number of keys
# Key #  Sequence #   Key name
44  31         # comma
49  0          # 1
50  4          # 2
51  8          # 3
52  12         # 4
53  16         # 5
54  20         # 6
55  24         # 7
56  28         # 8
97  2          # a
98  19         # b
99  11         # c
100  10        # d
101  9         # e
102  14        # f
103  18        # g
104  22        # h
105  29        # i
106  26        # j
107  30        # k
109  27        # m
110  23        # n
113  1         # q
114  13        # r
115  6         # s
116  17        # t
117  25        # u
118  15        # v
119  5         # w
120  7         # x
121  21        # y
122  3         # z
```

## 10.7 "rc" File / Keyboard Group Section

This section is the same as [**keyboard-control**], but to control groups of patterns, rather than individual patterns, using keystrokes. The keyboard group specifies more automation for the application. The first number specifies the key number, and the second number specifies the Group number.

Additional control items:

1. **# bpm up and down**. Keys to control BPM (beats per minute).
2. **# screen set up and down**. Keys for changing the active screenset.
3. **# group functionality on, off, learn**. Note that the group learn key is a modifier key to be held while pressing a group toggle key.
4. **#replace, queue, snapshot_1, snapshot_2, keep queue**. These are the other modifier keys explained in section 3a.

To see the required key codes when pressed, run `seq24` with the `--show-keys`.

Some keys should not be assigned to control sequences in *Sequencer64* as they are already assigned in the *Sequencer64* menu (with `Ctrl`).

This configuration item is the same as the **Keyboard** tab described in section 2.2.8.3 "Menu / File / Options / Keyboard" on page 23.

```
[keyboard-group]
# Key #, group #
32
33  0         # exclam
34  1         # quotedbl
35  2         # numbersign
36  3         # dollar
37  4         # percent
38  5         # ampersand
40  7         # parenleft
47  6         # slash
59  31        # semicolon
65  16        # A
66  28        # B
67  26        # C
68  18        # D
69  10        # E
70  19        # F
71  20        # G
72  21        # H
73  15        # I
74  22        # J
75  23        # K
77  30        # M
78  29        # N
81  8         # Q
82  11        # R
83  17        # S
```

```
84  12        # T
85  14        # U
86  27        # V
87  9         # W
88  25        # X
89  13        # Y
90  24        # Z
39 59         # bpm up, down: apostrophe semicolon
93 91 65360   # screen set up, down, play: bracketright bracketleft Home
236 39 65379  # group on, off, learn: igrave apostrophe Insert
# replace, queue, snapshot_1, snapshot 2, keep queue:
65507 65508 65513 65514 92  # Control_L Control_R Alt_L Alt_R backslash
1             # show_ui_sequence_key and pattern measures (1=true/0=false)
32            # space start sequencer
65307         # Escape stop sequencer
0 #  show sequence numbers (1 = true / 0 = false);  ignored in legacy mode
```

Note that most of these group-control keys are shifted versions of the keystrokes that control the individual sequences. Also note the `Control_L` and `Control_R` notations a few lines above. Please avoid using any Control key combinations in the "rc"/Keyboard configuration. Control keys are the province of the user-interface (*Gtk+*) and assigning them can cause surprising behavior! It is also wise to avoid the `Alt` key.

When in group-learn mode, the `Shift` key cannot be hit, so the group-learn mode automatically converts the keys to their shifted versions. This feature known as *shift-lock* or *auto-shift*.

## 10.8 "rc" File / JACK Transport

This section holds the settings for both JACK transport and for native JACK MIDI mode.

The JACK Transport options are also command-line options, as indicated in the comments below.

This configuration item is the same as the **Jack Sync** tab described in section 2.2.8.6 "Menu / File / Options / Jack Sync" on page 30.

```
[jack-transport]
# jack_transport - Enable slave sync with JACK Transport.
0
# jack_master - Sequencer64 attempts to serve as JACK Master.
0
# jack_master_cond - Sequencer64 is master if no other master exists.
0
# song_start_mode (applies mainly if JACK is enabled)
# 0 = Playback in live mode. Allows muting and unmuting of loops.
# 1 = Playback uses the song editor's data.
1
```

An additional item, new, specifies if native JACK MIDI input/output is to be used.

```
# jack_midi - Enable JACK MIDI, which is a separate option from
# JACK Transport.
1
```

Please note that only *one* of jack_transport, jack_master, and jack_master_cond should be selected (set to 1) at a time. Also note that JACK transport is separately configurable from JACK MIDI, and each uses a different JACK client internally.

## 10.9 "rc" File / MIDI Clock Mod Ticks

This configuration item is the same as the **Clock Start Modulo** option described in paragraph 2.2.8.1 "Menu / File / Options / MIDI Clock" on page 17.

```
[midi-clock-mod-ticks]
64
```

## 10.10 "rc" File / MIDI Meta Events

This section defines some features of MIDI meta-event handling. Normally, tempo events are supposed to occur in the first track (pattern 0). But one can move this track elsewhere to accomodate one's existing body of tunes. If affects where tempo events are recorded. The default value is 0, the maximum is 1023. A pattern must exist at this number for it to work.

```
[midi-meta-events]
0    # tempo_track_number
```

## 10.11 "rc" File / MIDI Input

This configuration item is the same as the **MIDI Input** tab described in paragraph 2.2.8.2 "Menu / File / Options / MIDI Input" on page 21. The "1" is a line count, and would equal the number of supported input ports. This "rc" entry here has two variables; the first is the record number or port number, and the second number indicates whether it is disabled (0), or enabled (1).

```
[midi-input]
1   # number of MIDI busses
# The first number is the port number, and the second number
# indicates whether it is disabled (0), or enabled (1).
# [0] 0:0 seq64 midi in
0 0
# If set to 1, this option allows the master MIDI bus to record
# (filter) incoming MIDI data by channel, allocating each incoming
# MIDI event to the sequence that is set to that channel.
# This is an option adopted from the Seq32 project at GitHub.
0   # flag to record incoming data by channel
```

There is no user-interface item for the following value, but it does correspond to the `--manual-alsa-ports` command-line option.

## 10.12 "rc" File / Manual ALSA Ports

The name of this setting is a bit of a misnomer in a couple of ways:

1. It actually refers to the usage of *virtual* MIDI ports. These are ports that are set up by the application so that other devices or applications can connect to the MIDI application. Where the "manual" idea comes in is that the user can manually choose the connections to be made.
2. This option is not just for ALSA. It can also be used when *Sequencer64* is running in native JACK mode, so support virtual JACK ports that can be connected manually (e.g. in the *QJackCtl* application.)

```
[manual-alsa-ports]
# Set to 1 to have sequencer64 create its own ALSA ports and not
# connect to other clients.  Use 1 to expose all 16 MIDI ports to
# JACK (e.g. via a2jmidid).  Use 0 to access the ALSA MIDI ports
# already running on one's computer, or to use the autoconnect
# feature (Sequencer64 connects to existing JACK ports on startup.
1
```

The opposite of `--manual-alsa-ports` is `--auto-alsa-ports`. The auto-alsa-ports option forces *Sequencer64* to use the system's existing ALSA ports. This is necessary in order to play tunes through software synthesizers that use ALSA MIDI.

Turning on the manual-alsa-ports option is necessary if one wants to use the legacy *Sequencer64* (`sequencer64`) with JACK. It is *not* necessary if using the native JACK MIDI version, `seq64`. However, if one needs to avoid the auto-connect feature of `seq64`, then the manual option is necessary.

It will create ports as per the settings in the "user" configuration file's `user-midi-bus-definitions` and `user-midi-bus-N` sections. These definitions can be used by JACK for connection, and these definitions can be used to specifically rename the ports that exist in the system. However, this option is misleading if one wants to have access to the actual ALSA ports that exist on the system. The next option gets around that issue.

## 10.13 "rc" File / Reveal ALSA Ports

Again, this option applies to both ALSA and native JACK.

```
[reveal-alsa-ports]
# Set to 1 to have sequencer64 ignore any system port names
# declared in the 'user' configuration file.  Use this option if
# you want to be able to see the port names as detected by ALSA.
```

```
1   # flag for reveal ALSA ports
```

Turning on the reveal-alsa-ports option is necessary if one wants to see the actual ALSA port names defined by the system. It will ignore the settings in the "user" configuration file's `user-midi-bus-definitions` and `user-midi-bus-N` sections. If this option is turned on, the definitions in the "user" configuration file are *not* read from that file.

## 10.14 "rc" File / Interaction Method

This configuration item is the same as the **Mouse** tab described in paragraph 2.2.8.5 "Menu / File / Options / Mouse" on page 28.

```
[interaction-method]
# 0 - 'seq24' (original seq24 method)
# 1 - 'fruity' (similar to a certain fruity sequencer we like)
0   # interaction_method
```

Note that the "fruity" mouse support is not available in the Qt user-interface.

There is an option to use the Mod4 (Super, or Windows) key in the Pattern Editor to lock the editing of a note. When this mode is enabled, and Mod4 is pressed while the mouse right-button is released, the editing pencil icon remains, and notes can be added. This feature is useful for crippled trackpads and trackpad drivers that cannot provide two simultaneous button presses.

```
# Set to 1 to allow seq24 to stay in note-adding mode when
# the right-click is released while holding the Mod4 (Super or
# Windows) key.
1   # allow_mod4_mode
```

Note that the "edit-lock" support is not available in the Qt user-interface. It is just as easy to click in that pattern editor and press the 'p' key to enter "paint" (edit) mode.

This option comes from the *seq32* project. It allows for pattern-splitting in the Song editor at snap points, rather than just at the middle of the pattern.

```
# Set to 1 to allow Sequencer64 to split performance editor
# triggers at the closest snap position, instead of splitting the
# trigger exactly in its middle.  Remember that the split is
# activated by a middle click.
0   # allow_snap_split
```

Not sure that this support is available in the Qt user-interface.

This option allows one to enable/disable the ability to double-click in a pattern slot in the main window to bring it up for editing. This can interfere with a live performance where muting/unmuting come fast enough to be seen as a double-click.

```
# Set to 1 to allow a double-click on a slot to bring it up in
# the pattern editor.  This is the default.  Set it to 0 if
# it interferes with muting/unmuting a pattern.
1   # allow_click_edit
```

Not sure that this support is available in the Qt user-interface.

## 10.15 "rc" File / LASH Session

The following configuration item is the same as the `--lash` or `--no-lash` options described in section 14 "Sequencer64 Man Page" on page 155. If set to 0, LASH session support is disabled. If set to 1, LASH session support is enabled. However, if LASH support is not built into the application, neither option has any effect – there is no LASH support. To determine if LASH support is built in, run sequencer64 from the command line with the `--version` option, and see if LASH is mentioned.

```
[lash-session]
# Set the following value to 0 to disable LASH session management.
# Set the following value to 1 to enable LASH session management.
# This value will have no effect is LASH support is not built into
# the application.  Use the --help option to see if LASH is part of
# the options list.
1     # LASH session management support flag
```

## 10.16 "rc" File / Auto Option Save

This new item determines if the "rc" configuration file is saved upon exit of *Sequencer64*. The legacy behavior is to save it, which can sometimes be inconvenient when one is just trying out some command-line options.

```
[auto-option-save]
# Set the following value to 0 to disable the automatic saving of the
# current configuration to the 'rc' file.  Set it to 1 to
# follow legacy seq24 behavior of saving the configuration at exit.
# Note that, if auto-save is set, many of the command-line settings,
# such as the JACK/ALSA settings, are then saved to the configuration,
# which can confuse one at first.  Also note that one currently needs
# this option set to 1 to save the configuration, as there is not a
# user-interface control for it at present.
0     # auto-save-options-on-exit support flag
```

## 10.17 "rc" File / Last Used Directory

The following item refers to the last directory in which one opened or saved a MIDI file.

```
[last-used-dir]
/home/ahlstrom/Home/ca/mls/git/sequencer64/contrib/midi/
```

### 10.18 "rc" File / Recent Files

The following item preserves a list of the last few MIDI files loaded. It is not filled when a MIDI file is loaded via a play-list.

```
[recent-files]
4
/home/ahlstrom/Home/ca/mls/git/sequencer64-alternate/contrib/midi/2Bars.midi
contrib/midi/b4uacuse-seq24.midi
contrib/midi/Bars.midi
contrib/midi/b4uacuse-GM-format.midi
```

### 10.19 "rc" File / Play-List

This is a feature new with version 0.96.0 of *Sequencer64*. It provides a configured set of named play-lists in a play-list file, and a flag to activate it.

```
[playlist]
0     # playlist_active, 1 = active, 0 = do not use it
# Provides the name of a play-list.  If there is none, use '""'.
# Or set the flag above to 0.
/home/ahlstrom/.config/sequencer64/sample.playlist
```

See section 12 "Sequencer64 Play-Lists" on page 141. It describes the setup, layout, and usage of a *Sequencer64* playlist.

## 11 Sequencer64 "usr" Configuration File

There are two *Sequencer64* configuration files: `sequencer64.rc` and `sequencer64.usr`. See section 10 "Sequencer64 "rc" Configuration File" on page 102; it describes the "rc" file. It is a bit different in how it is handled.

The *Sequencer64* "usr" (or "user") configuration file provides a way to give more informative names to the MIDI busses, MIDI channels, and MIDI controllers of a given system setup. This configuration will override the default values of some drop-down lists and menu items, and make them reflect your names for them. In *Sequencer64* it, also includes some items that affect the user-interface's look, and other new configuration items.

Unlike the "rc" file, the "user" file is *not* written every time *Sequencer64* exits. If the "user" files does not exist, one is created, but it is normally not overwritten thereafter. To cause it to be overwritten at exit, run *Sequencer64* with the `-u` or `--user-save` option:

```
$ seq64 --user-save
```

This option is recommended when one installs a new version of *Sequencer64*, which might add new options to the "user" file. See section 14 "Sequencer64 Man Page" on page 155; it discusses more options involving the "user" file.

Another difference between the "rc" file and the "user" file is that the "user" file currently has no graphical user-interface dialog to configure the "user" settings. One has to edit the file manually.

The original purpose for the "user" file was to create familiar names for the system MIDI devices. By default, the list of MIDI devices that *Sequencer64* shows depends on one's system setup and whether the manual-alsa-port option is specified or not. Here's our system, which has Timidity installed and running as a service, and the `[manual-alsa-port]` option turned off, shown in a composite view with all menus one can look at for MIDI settings:

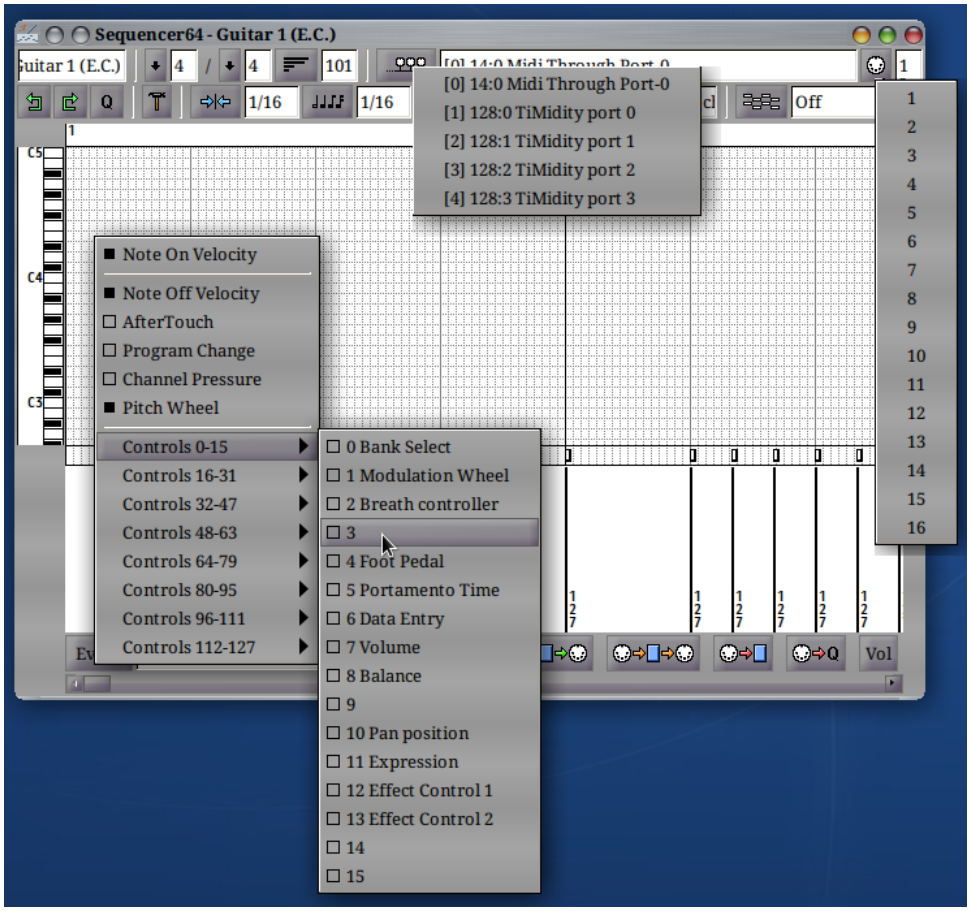



Figure 81: Sequencer64 Composite View of Native Devices

At the top center, the dropdown menu contains the 5 MIDI busses/ports supported by this computer. At right, the MIDI channel shows the channels numbers that can be picked for buss 0. At bottom left, we see the default controller values that *Sequencer64* includes. We have no idea if these correspond to any controllers that the selected MIDI buss supports. We *can* use this dropdown to see if any such controller events are in the loaded MIDI file, of course; a solid black square indicates that such an event was found

in the pattern.

Assume we have 3 MIDI "buss" devices hooked to our system: two Model "2x2" MIDI port devices, and an old PCR-30 MIDI controller keyboard. Let's number them:

1. Model 2x2 A
2. Model 2x2 B
3. PCR-30

Then assume that we have nine different MIDI instruments in our kit. Let's number them, too:

1. Waldorf Micro Q
2. SuperNova
3. DrumStation
4. TX81Z
5. WaveStation
6. ESI-2000
7. ES-1
8. ER-1
9. TB-303

The Waldorf Micro Q, the SuperNova, and the DrumStation all have a large number of special MIDI controller values for affecting the sound they produce. The DrumStation accepts MIDI controllers that change various features of the sound of each type of drum it supports.

The buss devices can be configured to route certain MIDI channels to certain MIDI devices. Assume we have them set up this way:

1. Model 2x2 A
    - SuperNova: channels 1 to 8
    - TX81Z: channels 9 to 11
    - Waldorf Micro Q: channels 12 to 15
    - DrumStation: channel 16
2. Model 2x2 B
    - WaveStation: channels 1 to 4
    - ESI-2000: channels 5 to 14
    - ES-1: channel 15
    - ER-1: channel 16
3. PCR-30
    - TB-303: channel 1

How can we get *Sequencer64* to show these items with the proper names associated with each device, channel, and controller value? We use the oddly-named **"user" configuration file**.

The *Seq24* configuration file was called `.seq24usr`, and it was stored in the user's `$HOME` directory. *Sequencer64* uses a new file-name to take its place, with a fall-back to the original file-name if the new file does not exist, or if *Sequencer64* is running in legacy mode. After one runs *Sequencer64* for the first time (or after deleting the configuration files), it will generate a `sequencer64.usr` file in your home directory:

```
/home/ahlstrom/.config/sequencer64/sequencer64.usr
```

It allows you to give an alias to each MIDI bus, MIDI channel, and MIDI control codes, per channel. The file-name is a bit misleading... do not confuse this file with the `sequencer64.rc` file.

The process for setting up the user file is to:

1. Define one or more MIDI busses, the name of each, and what instruments are on which channels. Each buss is configured in a section of the form "`[user-midi-bus-X]`", where "X" ranges from 0 on up. Each buss then defines up to 16 channel entries. Each entry includes the channel number and the number of a section in the user-instrument section described next.
2. Define all of the instruments and their controller names, if they have them. Each instrument is configured in a section of the form "`[user-instrument-X]`", where "X" ranges from 0 on up.

Let's walk through the structure of this setup, since it is a little bit tricky. Here is a diagram of the relationships between the buss definitions and instrument definitions:

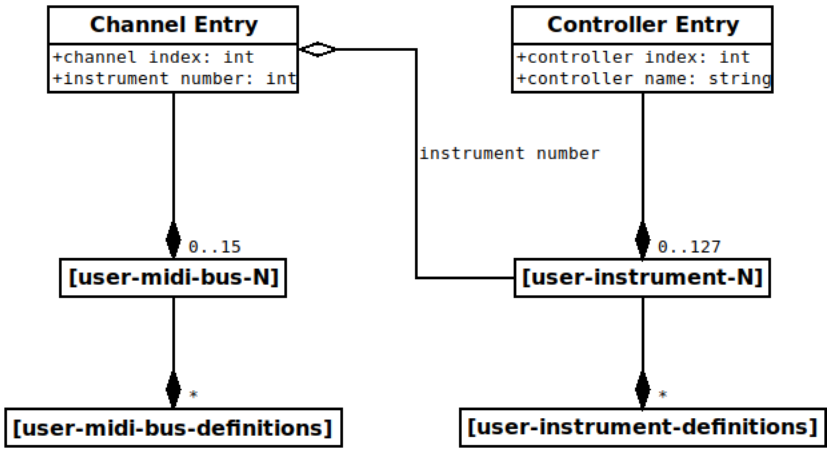



Figure 82: Busses and Instruments in the "usr" File

The first section in the "usr" file (after `[comments]`) is `[user-midi-bus-definitions]`. The solid diamond link, with the "*" marker, indicates that this section contains an arbitrary number ("*") of `[user-midi-bus-N]` sections, where "N" ranges from 0 on upward. These correspond to the MIDI busses expected to be in the system, ignoring the "announce" buss.

Each of the busses contains 16 (0 to 15) channel entries. These channels are referred to as "instrument numbers", and are represented as and linked to "instruments" in the `[user-instrument-definitions]` section. Each instrument contains up to 128 controller values; these controller values are available in the **Event** button in the Pattern Editor, and their names are shown.

So, each instrument is setup as a "channel" in a particular "buss". In the Pattern Editor, when a particular buss and channel is selected, the **Event** menu entries should match the controller entries set up in the "usr" file.

Taking our list of devices and channels we created above, which can be seen in the *Sequencer64* sample file `contrib/configs/sequencer64.usr.example`, and deducting 1 from each device number and channel number (so that numbering starts from 0), and consulting the device manuals to determine the controller values it supports, we can assemble a "user" configuration file that makes the setup visible in *Sequencer64*.

Peruse the next couple of sections to understand a bit about the format of this file. Look at the example files in the `contrib/configs` directory as well, to see the whole thing put together. Once satisfied, go to section 11.6 ""usr" File / Device and Control Names" on page 139, and see what it all looks like.

## 11.1 "usr" File / MIDI Bus Definitions

This section begins with an "INI" group marker `[user-midi-bus-definitions]`. It defines the number of user busses that will be configured in this file.

```
[user-midi-bus-definitions]
3     # number of user-defined MIDI busses
```

This means that the `sequencer64.usr` file will have three MIDI buss sections: [user-midi-bus-0], [user-midi-bus-1], and [user-midi-bus-2]. Here's is an annoted example of one such section:

```
[user-midi-bus-0]
2x2 A (SuperNova,Q,TX81Z,DrumStation)     # name of the device
16                                        # number of channels

# NOTE: Channels are 0-15, not 1-16.  Instruments set to -1 = GM

0 1                                       # channel and instrument
1 1      # Instrument #1 of the [user-instrument-definitions] section
2 1
. . .
7 1
8 3      # Instrument #3 of the [user-instrument-definitions] section
9 3
10 3
11 0     # Instrument #0 of the [user-instrument-definitions] section
12 0     # This is the Waldorf Micro Q device defined below
13 0
14 0
15 2     # Instrument #2 of the [user-instrument-definitions] section
```

Here's an example of one that needs only one override:

```
[user-midi-bus-2]                         # Instrument 2, see ch. 15 above
PCR-30 (303)
1                                         # number of channels
0 8                                       # channel and instrument
# The rest default to -1... General MIDI
```

Note that these user-instrument entries can be quickly disabled by changing the count values to 0. Also, these instrument-definition sections are read from the "user" configuration file only if the `--reveal-alsa-ports` option is *off* ("0"); this command-line option can also be specified in the `[reveal-alsa-ports]` section of the "rc" file, see section 10.13 ""rc" File / Reveal ALSA Ports" on page 121, where this section is discussed. Otherwise, the actual port names reported by ALSA are shown. The `user-midi-bus-definitions` and `user-midi-bus-N` sections can be misleading if one wants to have access to the actual ALSA ports that exist on the system. Therefore, if the `--reveal-alsa-ports` option is turned on, then the definitions in the "user" configuration file are *not* read from that file. The following figures show the results of various settings with an active "user" file. They have been clipped to save space.

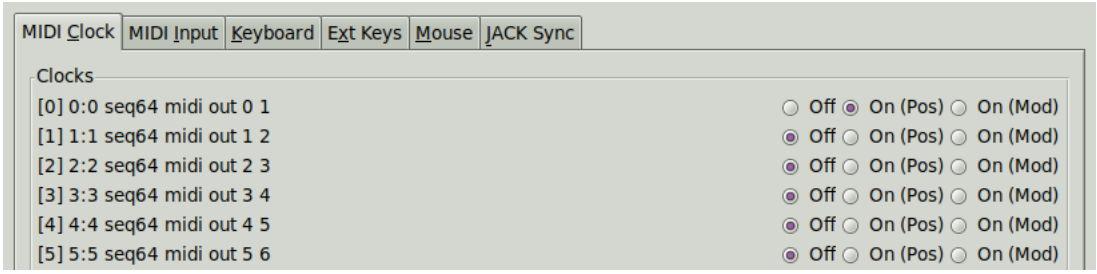



Figure 83: Clocks View, -m (–manual-alsa-ports)

Above, the virtual (manual) output ports are shown just as created by *Sequencer64*. The `--reveal-alsa-ports` option is *off* here.

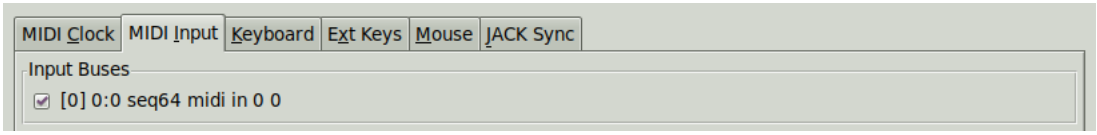



Figure 84: Inputs View with -m (–manual-alsa-ports) Option

Above, the single virtual (manual) input port is shown just as created by *Sequencer64*. Again, the `--reveal-alsa-ports` option is *off* here.

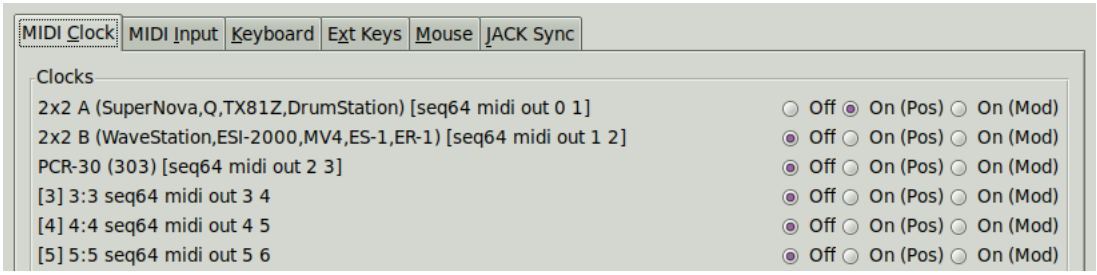



Figure 85: Clocks View, -m (–manual-alsa-ports) and -R (–hide-alsa-ports)

Above, by adding the "hide" ports option, the system port labels are replaced by the labels from the "usr" file.

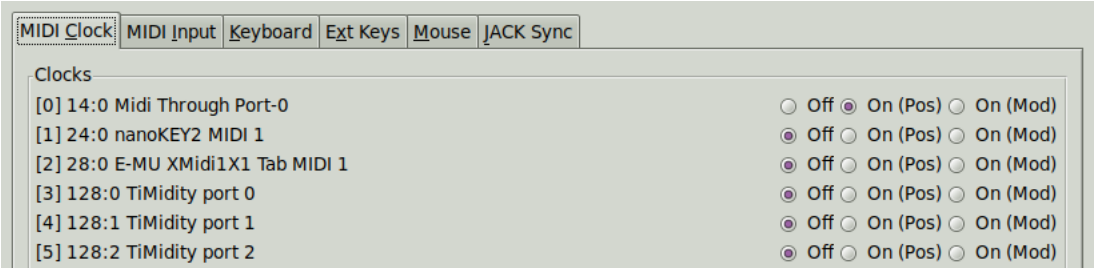



Figure 86: Clocks View, -r (–reveal-alsa-ports)

Above, the "reveal" ports option overrides the device names given in the "usr" file, so that the native system names of the output ports are shown.

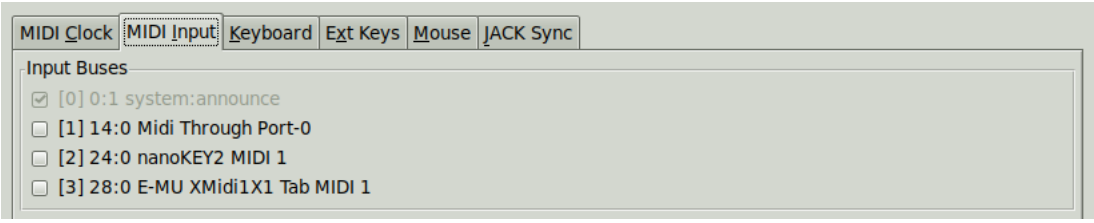



Figure 87: Inputs View with -r (–reveal-alsa-ports) Option

Above, the "reveal" ports option overrides the device names given in the "usr" file, so that the native system names of the input ports are shown. However, *Sequencer64* no longer overrides the names of the input ports via the "usr" file. This is done to save some trouble in displaying the input port names, which are shown only in this dialog. We may consider offering a separate override section for the input ports in the future.

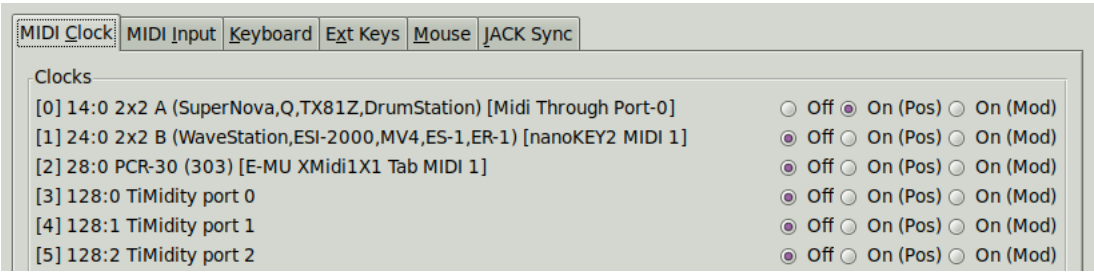



Figure 88: Clocks View with -R (–hide-alsa-ports) Option

The figure above shows how hiding the system port names shows the names defined in the "usr" file. But notice that the actual port names are shown in square brackets, for reference.

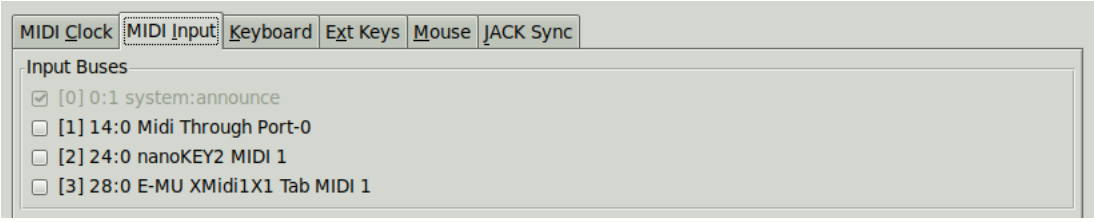



Figure 89: Inputs View with -R (–hide-alsa-ports) Option

Although the "hide" ports option is specified above, this view is currently also the normal view of the input ports, even with device names defined in the "usr" file. At this time, there's no real need to show the user-instrument names on the input port. If there turns out to be such a need, the definitions would likely need to be different from the definitions for the output ports. Another complexity is the possible existence, under ALSA, of the `system:announce` port.

## 11.2 "usr" File / MIDI Instrument Definitions

This section begins with an "INI" group marker `[user-instrument-definitions]`. It defines the number of user instruments that will be configured in this file. This section defines characteristics, such as the meanings of MIDI controller values, of the instruments themselves, not the MIDI busses to which they attached.

```
[user-instrument-definitions]
9     # number of user instrument
```

So this "usr" file will define 9 instruments. We provide only one section as an example. Note that items without text default to the values prescribed by the General MIDI (GM) specification.

```
[user-instrument-0]
Waldorf Micro Q                     # name of instrument
128                                 # number of MIDI controllers
0                                   # first controller value, unnamed
1 Modulation Wheel
2 Breath Control
3
4 Foot Control
   . . .
119
120 All Sound Off (0)
121 Reset All Controllers (0)
122 Local Control (0-127) (Off,On)
123 All Notes Off (0)
124                                 # defaults to GM
125 Unsupported
126 Unsupported
127                                 # defaults to GM
```

Note the unnamed control numbers above. An unnamed control number might be an unsupported control number. It is termed to be "inactive". In this case, the event menu of the pattern editor will show the default name of this controller. Again, though, the function denoted by this name might not be supported by the device. In that case, it might be better to call it "Unsupported". See the examples above. Here is an instrument that has synthesis parameters that can be controlled:

```
[user-instrument-1]
SuperNova
128
0 Bank Select MSB
1 Modulation Wheel
2 Breath Controller
3 Arp Pattern Select
   . . .
121 Reset Controllers
122 Local Control [*]
123 All Notes Off
124 All Notes Off
125 All Notes Off
126 All Notes Off
127 All Notes Off
```

Here is an instrument that perhaps has no controllers, or perhaps is simply not configured by the musician yet:

```
[user-instrument-4]
WaveStation
0
```

The sample file `contrib/configs/sequencer64-timidity-yoshimi.usr.example` contains examples of some other kinds of instruments. It is a minimal resource, useful to study when creating one's own settings.

## 11.3 "usr" File / User Interface Settings

This section, new to *Sequencer64*, begins with an "INI" group marker `[user-interface-settings]`.

It provides for a feature we will hopefully be able to complete some day: the complete specificition of the appearance of the user-interface. There is plenty of room to change the appearance of *Sequencer64* already! Please try the settings and see what you like.

```
#   ======== Sequencer64-Specific Variables Section ========

[user-interface-settings]
```

```
# These settings specify the soon-to-be-modifiable sizes of
# the Sequencer64 user-interface elements.

# Specifies the style of the main-window grid of patterns.
#
# GTK:
#   0 = Normal style, matches the GTK theme, has brackets.
#   1 = White grid boxes that have brackets.
#   2 = Black grid boxes (no brackets, our favorite).
#
# Qt:
#   0 = Slot coloring matches Kepler34.
#   1 = Slot coloring more like GTK.
1       # grid_style



# Specifies box style box around a main-window grid of patterns.
# 0  = Draw a whole box around the pattern slot.
# 1  = Draw brackets on the sides of the pattern slot.
# 2 and up = make the brackets thicker and thicker.
# -1 = same as 0, draw a box one-pixel thick.
# -2 and lower = draw a box, thicker and thicker.
2       # grid_brackets



# Specifies the number of rows in the main window.
# Values of 4 (the default) through 8 (the best alternative value)
# are allowed.
4       # mainwnd_rows



# Specifies the number of columns in the main window.
# At present, only values from 8 (the default) to 12 are supported.
8       # mainwnd_cols



# Specifies the maximum number of sets, which defaults to 1024.
# It is currently never necessary to change this value.
32      # max_sets



# Specifies the border width in the main window.
0      # mainwid_border
```

```
# Specifies the border spacing in the main window.
2      # mainwid_spacing



# Specifies some quantity, it is not known what it means.
0      # control_height



# Specifies the initial zoom for the piano rolls.  Ranges from 1.
# to 32, and defaults to 2 unless changed here.
2      # zoom



# Specifies if the key, scale, and background sequence are to be
# applied to all sequences, or to individual sequences.  The
# behavior of Seq24 was to apply them to all sequences.  But
# Sequencer64 takes it further by applying it immediately, and
# by saving to the end of the MIDI file.  Note that these three
# values are stored in the MIDI file, not this configuration file.
# Also note that reading MIDI files not created with this feature
# will pick up this feature if active, and the file gets saved.
# It is contagious.
#
# 0 = Allow each sequence to have its own key/scale/background.
#     Settings are saved with each sequence.
# 1 = Apply these settings globally (similar to seq24).
#     Settings are saved in the global final section of the file.
1      # global_seq_feature



# Specifies if the old, console-style font, or the new anti-
# aliased font, is to be used as the font throughout the GUI.
# In legacy mode, the old font is the default.
#
# 0 = Use the old-style font.
# 1 = Use the new-style font.
1      # use_new_font



# Specifies if the user-interface will support two song editor
# windows being shown at the same time.  This makes it easier to
```

```
# edit songs with a large number of sequences.
#
# 0 = Allow only one song editor (performance editor).
# 1 = Allow two song editors.
1      # allow_two_perfedits



# Specifies the number of 4-measure blocks for horizontal page
# scrolling in the song editor.  The old default, 1, is a bit
# small.  The new default is 4.  The legal range is 1 to 6, where
# 6 is the width of the whole performance piano roll view.
4      # perf_h_page_increment



# Specifies the number of 1-track blocks for vertical page
# scrolling in the song editor.  The old default, 1, is a bit
# small.  The new default is 8.  The legal range is 1 to 18, where
# 18 is about the height of the whole performance piano roll view.
8      # perf_v_page_increment



# Specifies if the progress bar is colored black, or a different
# color.  The following integer color values are supported:
#
# 0 = black
# 1 = dark red
# 2 = dark green
# 3 = dark orange
# 4 = dark blue
# 5 = dark magenta
# 6 = dark cyan
6      # progress_bar_colored



# Specifies if the progress bar is thicker.  The default is 1
# pixel.  The 'thick' value is 2 pixels.  (More than that is not
# useful.  Set this value to 1 to enable the feature, 0 to disable
# it.
1      # progress_bar_thick



# Specifies using an alternate (darker) color palette.  The
# default is the normal palette.  Not all items in the user
# interface are altered by this setting, and it's not perfect.
```

```
# Set this value to 1 to enable the feature, 0 to disable it.
0      # inverse_colors



# Specifies the window redraw rate for all windows that support
# that concept.  The default is 40 ms.  Some windows used 25 ms.
40     # window_redraw_rate



# Specifies using icons for some of the user-interface buttons
# instead of text buttons.  This is purely a preference setting.
# If 0, text is used in some buttons (the main window buttons).
# Otherwise, icons are used.  One will have to experiment :-).
0      # use_more_icons (currently affects only main window)



# Specifies the number of set window ('wid') rows to show.
# The long-standing default is 1, but 2 or 3 may also be set.
# Corresponds to 'r' in the '-o wid=rxc,f' option.
2      # block_rows (number of rows of set blocks/wids)
```

This option is *not supported* in the Qt version. Instead, the user can create an arbitrary number of live-frame windows.

```
# Specifies the number of set window ('wid') columns to show.
# The long-standing default is 1, but 2 may also be set.
# Corresponds to 'c' in the '-o wid=rxc,f' option.
2      # block_columns (number of columns of set blocks/wids)



# Specifies if the multiple set windows are 'in sync' or can
# be set to arbitrary set numbers independently.
# The default is false (0), means that there is a single set
# spinner, which controls the set number of the upper-left 'wid',
# and the rest of the set numbers follow sequentially.  If true
# (1), then each 'wid' can be set to any set-number.
# Corresponds to the 'f' (true, false, or 'indep') in the
# '-o wid=rxc,f' option.  Here, 1 is the same as 'indep' or false,
# and 0 is the same as f = true.  Backwards, so be careful.
1      # block_independent (separate set spinner for blocks/wids)
```

Note that the window-redraw rate option is meant more for experimentation than anything else. It probably doesn't affect CPU usage much, but might provide a smoother-running cursor on some systems.

## 11.4 "usr" File / User MIDI Settings

This section begins with an "INI" group marker `[user-midi-settings]`. It supports files with different PPQN, and and allows one to specify the global defaults for tempo, beats per measure, and so on.

```
[user-midi-settings]

# These settings specify MIDI-specific value that might be
# better off as variables, rather than constants.
# Specifies parts-per-quarter note to use, if the MIDI file.
# does not override it.  Default is 192, but we'd like to go
# higher than that.  BEWARE:  STILL GETTING IT TO WORK!
192     # midi_ppqn



# Specifies the default beats per measure, or beats per bar.
# The default value is 4.
4       # midi_beats_per_measure/bar



# Specifies the default beats per minute.  The default value
# is 120, and the legal range is 1 to 600.
120     # midi_beats_per_minute



# Specifies the default beat width. The default value is 4.
4       # midi_beat_width



# Specifies the buss-number override. The default value is -1,
# which means that there is no buss override.  If a value
# from 0 to 31 is given, then that buss value overrides all
# buss values specified in all sequences/patterns.
# Change this value from -1 only if you want to use a single
# output buss, either for testing or convenience.  And don't
# save the MIDI afterwards, unless you really want to change
# all of its buss values.
-1     # midi_buss_override
```

For the new 0.90 series, additional values for the `[user-midi-settings]` section have been added:

```
# Specifies the default velocity override when adding notes in the
# sequence/pattern editor.  This value is obtained via the 'Vol'
# button, and ranges from 0 (not recommended :-) to 127.  If the
```

```
      # value is -1, then the incoming note velocity is preserved.
      80     # velocity_override (-1 = 'Free')



      # Specifies the precision of the beats-per-minutes spinner and
      # MIDI control over the BPM value.  The default is 0, which means
      # the BPM is an integer.  Other values are 1 and 2 decimal digits
      # of precision.
      1      # bpm_precision



      # Specifies the step increment of the beats/minute spinner and
      # MIDI control over the BPM value.  The default is 1. For a
      # precision of 1 decimal point, 0.1 is a good value.  For a
      # precision of 2 decimal points, 0.01 is a good value, but one
      # might want somethings a little faster, like 0.05.
      0.1    # bpm_step_increment



      # Specifies the page increment of the beats/minute field. It is
      # used when the Page-Up/Page-Down keys are pressed while the BPM
      # field has the keyboard focus.  The default value is 10.
      5.0    # bpm_page_increment



# Specifies the minimum value of beats/minute in tempo graphing.
# By default, the tempo graph ranges from 0.0 to 127.0.
# This value can be increased to give a magnified view of tempo.
0       # midi_bpm_minimum



# Specifies the maximum value of beats/minute in tempo graphing.
# By default, the tempo graph ranges from 0.0 to 127.0.
# This value can be increased to give a magnified view of tempo.
360       # midi_bpm_maximum
```

The `velocity-override` option fixes a long standing (from *Seq24*) bug where the actual incoming note velocity was always replaced by a hard-wired value.

The `bpm-precision`, `bpm-step-increment`, and `bpm-page-increment` values allow more precise control over tempo, which makes it easier to match the tempo of external music sources. Note that the step-increment is used by the up/down arrow buttons, the up/down arrow keys, and the MIDI BPM control values. The page-increment is used if the BPM field has focus and the Page-Up/Page-Down keys are pressed, and new MIDI control values have been added to support coarse MIDI control of tempo.

The `midi-bpm-minimum` and `midi-bpm-maximum` settings are used in scaling the display of Tempo events. By adjusting these values, one can more easily see the variations in tempo. In a main window pattern slot, or in the song editor tempo track, this range is scaled to the full range of note values, 0 to 127. Generally, one wants to select a range that keeps the main tempo line at the middle height of the pattern display.

To obtain these new settings, remember to backup the existing *sequencer64.usr*, then run *Sequencer64* with the `--user-save` option, and then do a "diff" on the new file and the original to merge any old values that need to be preserved. Then make any further tweaks to the new values.

## 11.5 "usr" File / User Options

This section begins with an "INI" group marker `[user-options]`. It provides for additional options keyed by the `-o`/`--option` options. This group of options serves to expand the options that are available, since *Sequencer64* is running out of single-character options. This group of options are shown below.

```
# The daemonize option is used in seq64cli to indicate that the
# application should be gracefully run as a service.
0       # option_daemonize
```

If this option is not used when running `seq64cli`, then the application stays in the console window and dumps informational output to it. If this option is in force, then the only way to affect `seq64cli` is to send a signal (e.g. SIGKILL) to it, or use MIDI control.

```
# This value specifies an optional log-file that replaces output
# to standard output and standard error.  To indicate no log-file,
# the string "" is used.
"seq64.log"
```

This log-file is written to the same directory as the "rc" and "usr" files. If this file-name is empty, then a valid file-name must be specified in the "–option log=filename.log" option. Note that this file is always written to the *Sequencer64* configuration directory.

## 11.6 "usr" File / Device and Control Names

Okay, now we have this file copied to our home directory:

```
/home/ahlstrom/.config/sequencer64/sequencer64.usr
```

If we'd already run *Sequencer64* at least once, we'd have overwritten the skeleton sample file that *Sequencer64* writes by default. We now have a full-fledged "user" file.

However, because we don't actually have all that equipment (we got the example from the Web, for cryin' out loud), let's see what we end up with when we run *Sequencer64* this time and show the pattern editor settings:

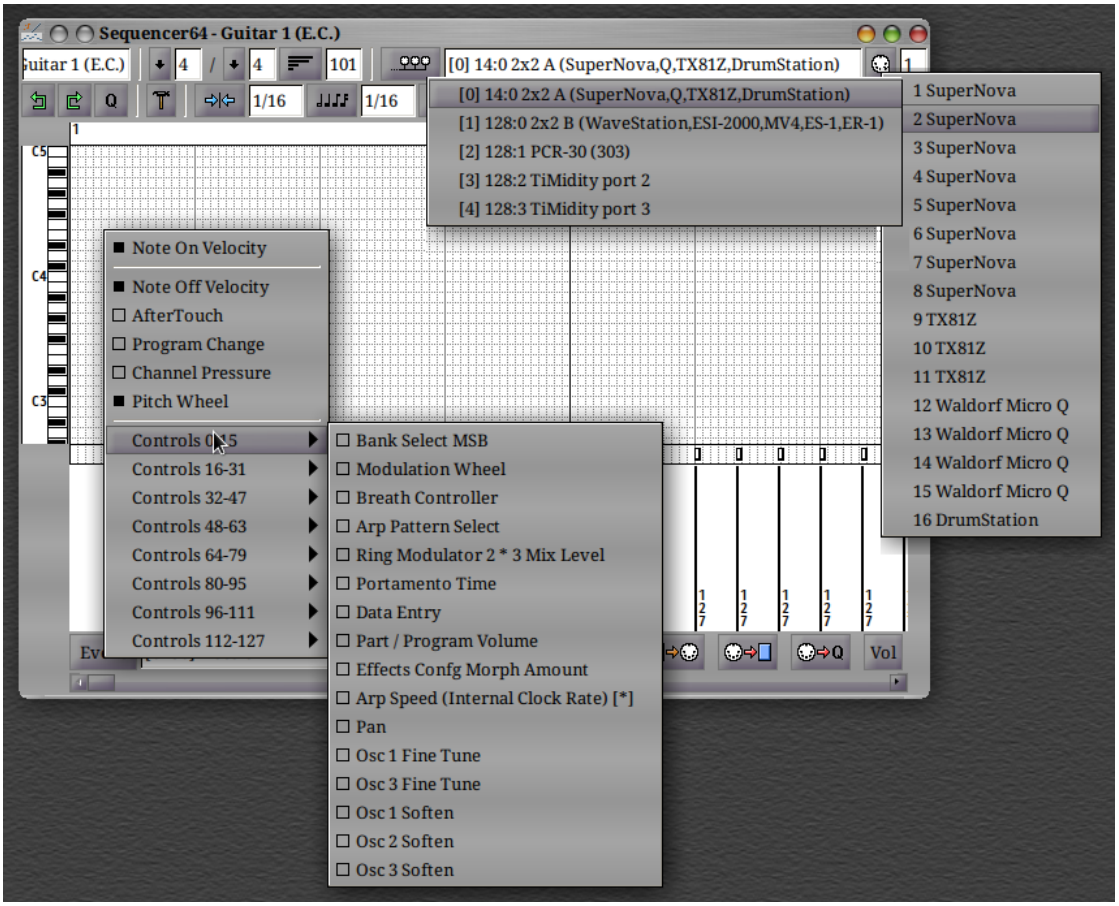



Figure 90: Sequencer64 Composite View of Non-Native Devices

Compare that diagram to Figure 81 "Sequencer64 Composite View of Native Devices" on page 125. If the original figure, we saw the 5 native busses (ports) on our system, their bare-bones channel numbers, and the default controller values. In this new figure, we see the three buss devices (ports), plus the two Timidity ports. If we stopped the Timidity service, these would go away.

Look at the selected buss, "[0]". It's 16 channels are now associated with the devices to which the channels have been assigned. Thus, when we have a new pattern we've created in *Sequencer64*, we can assign it to exactly the buss and device we want.

If we don't have port-mappers installed, and thus have only one playback device plugged into the buss, we can still create a setup that shows the device and a specific program setup. Doing so would be tedious, but perhaps there's some automated way to do it? Lastly, note the following figure.

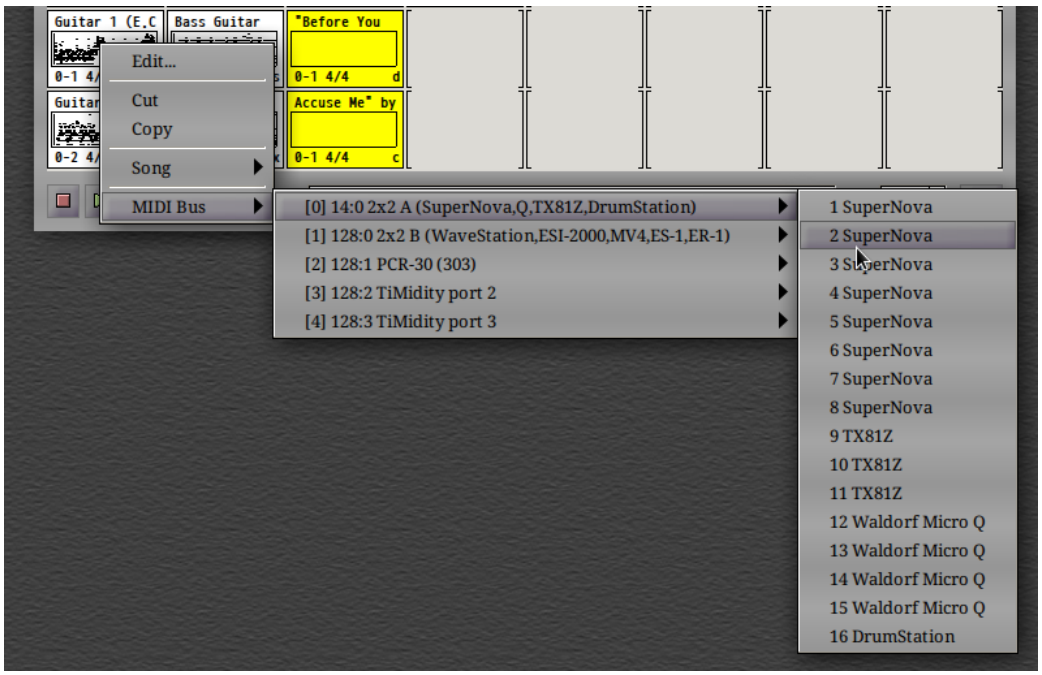



Figure 91: The MIDI Bus Menu for a Specific Pattern

This figure shows that we can also select the desired port and channel directly from the main window. There's more to the "user" configuration file than we've exposed here.

# 12 Sequencer64 Play-Lists

*Sequencer64* now supports play-lists. A play-list is a variation on the "rc" file, conventionally ending with the extension `.playlist`. It contains a number of "playlist" sections, each with a human-readable title, selectable via a MIDI control number. Each playlist section contains a list of songs, each selectable via a MIDI control number or by moving to the next or previous song in the list.

Movement between the playlists and the songs is accomplished via MIDI control; see section 10.2 ""rc" File / MIDI Control" on page 103. Using MIDI control makes it possible to use the `seq64cli` headless version of *Sequencer64* in a live setting. In the normal user-interface, play-list movement can also be done manually via the four arrow keys on the computer keyboard.

The playlist can be specified on the command-line, or in the "rc" file (see section 10.19 ""rc" File / Play-List" on page 124). If it is specified on the command line, that playlist setup will be written to the "rc" file. It can be removed by specifying a blank play-list name.

The play-list format is defined in the following section. Later sections describe the various user-interfaces.

## 12.1 Sequencer64 Play-Lists / Format

The play-list file, by convention, has a file-name of the form `sample.playlist`. The play-list file starts with a hardwired top banner that the user can edit with a text editor. It can also have an optional comments section, much like the "rc" and "usr" files. It is *not* overwritten when *Sequencer64* exits.

```
[comments]

Comments added to this section are preserved.  Lines starting with
a '#' or '[', or that are blank, are ignored.  Start lines that must
be blank with a space.
```

A blank line (not even a space) ends the comment section. Following the comments section is a `[playlist-options]` section.

```
[playlist-options]
1   # If set to 1, when a new song is selected, unmute all its patterns
```

That option allows the load of the next song to enable the patterns in that song.

Following the options section are one or more `[playlist]` sections. Here is the layout of a sample playlist section.

```
[playlist]

# Playlist number, arbitrary but unique. 0 to 127 recommended
# for use with MIDI playlist control.
126

# Display name of this play list.
"Music for Serious Dogs"

# Storage directory for the song-files in this play list.
contrib/midi/

# Provides the MIDI song-control number, and also the
# base file-name (tune.midi) of each song in this playlist.
# The playlist directory is used, unless the file-name contains its
# own path.
70 allofarow.mid
71 CountryStrum.midi
72 contrib/wrk/longhair.wrk
```

A play-list file can have more than one `[playlist]` section. This allows for partitioning songs into various groups that can be easily selected (e.g. based on the mood of the musician or the audience).

After the `[playlist]` tag comes the play-list number. This number can be any non-negative value. However, in order to use MIDI control to select the playlist, this number should be limited to the range 0 to 127. If there is more than one `[playlist]` section, they are ordered by this number, regardless of where they sit in the play-list file.

Next comes a human-readable name for the playlist, which is meant to be displayed in the user-interface. If surrounded by quotes, the quotes are removed before usage.

Next is the song-storage directory. This directory is the default location in which to find the songs. It can be an absolute directory or a relative directory. However, be wary of using relative directories, since they depend on where *Sequencer64* is run. Also, if a song's file-name has its own directory component, that overrides the default song-storage directory.

Lastly, there is a list of MIDI song file-names, preceded by their numbers. As with the playlist numbers, it is recommended to keep them between 0 and 127, for usage with MIDI control. And the songs are ordered by this number, rather than by their position in the list.

## 12.2 Sequencer64 Play-Lists / "rc" File

The most consistent way to specify a play-list is to add an entry like the following to the "rc" file:

```
[playlist]
# Provides a configured play-list and a flag to activate it.
0     # playlist_active, 1 = active, 0 = do not use it
# Provides the name of a play-list.  If there is none, use '""'.
# Or set the flag above to 0.
/home/ahlstrom/.config/sequencer64/sample.playlist
```

This setup allows a play-list file to be specified and activated. If the name of the play-list file does *not* contain a directory, then the play-list file is search for in the user's *Sequencer64* configuration directory.

If the play-list file-name is empty (i.e. set to `""`, then there is no play-list active.

## 12.3 Sequencer64 Play-Lists / Command Line Invocation

The command-line options to specify (and activate) the play-list feature are:

```
-X playlist_file
--playlist playlist_file
```

The play-list file is either a base-name (e.g. `sample.playlist`) or a name that includes the full path to the play-list file (e.g. `data/sample.playlist`). If no path is specified, the directory is the currently set *Sequencer64* configuration-file directory.

Please note that any play-list file specified on the command line will be written into the "rc" file's `[playlist]` section when *Sequencer64* exits.

## 12.4 Sequencer64 Play-Lists / Verification

Currently, when *Sequencer64* loads a play-list file, every song in the file is verified by loading it. If any load fails, then the playlist will fail to load.

We may add an option to avoid that check, though it goes fast on modern hardware.

## 12.5 Sequencer64 Play-Lists / User Interfaces

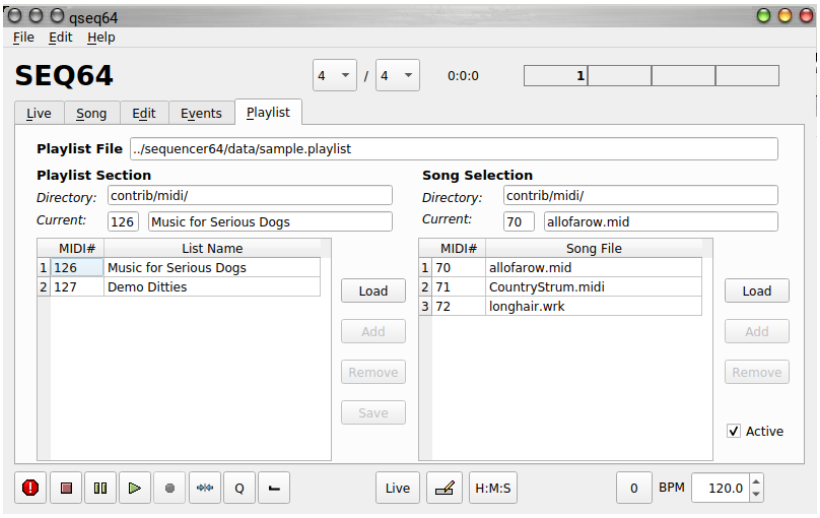
### 12.5.1 Sequencer64 Play-Lists / User Interfaces / Gtkmm-2.4

The Gtkmm-2.4 user-interface is currently the normal user-interface. However, it will ultimately be overtaken by the Qt 5 user-interface. So, for Gtkmm-2.4, *Sequencer64* merely allows the selection of the play-list and its songs via the four arrow keys, and it simply displays the current play-list name and the current song in the main window.

The `Up` and `Down` arrows move forward or backward through the list of play-lists, and the The `Right` and `Left` arrows move forward or backward through the list of songs for the currently-selected play-list.

### 12.5.2 Sequencer64 Play-Lists / User Interfaces / Qt 5

The Qt 5 user-interface will eventually become the standard. And it will (soon) support the full display, selection, and editing of the play-lists and the song-list for each play-list. The current playlist tab supports viewing the play-lists and songs, but not yet editing them.




Qt 5 Playlist Tab

Actually, the playlist currently has to be loaded from the command line or from the **File** menu.

# 13 Sequencer64 Qt 5 and Windows

With version 0.95.1 and above, *Sequencer64* provides builds using the *Qt 5* GUI framework and a modified version of the PortMidi framework (it cleans up the code and removes the need for *Java*). With this additional work, we now have ... *Sequencer64 for Windows*.

In *Linux*, the Qt 5 user-interface can be used with either the modified *PortMidi* library or the more powerful modified *RtMidi* library.

The Qt 5 user-interface is similar the *Kepler34* project [9]. But it uses the internal functions of *Sequencer64* and adds a number of features not present in *Kepler34*. The Qt version has a tabbed interface, plus some separate windows, and different ways of handling the main window (the "Live" tab),

the performance window (the "Song" tab), and the editing window (the "Edit" tab). It adds an "Events" tab and a "Playlists" tab. For portability reasons, the Qt user-interface will eventually be the *official* version for *Sequencer64*.

There is still some *Sequencer64* functionality that the Qt 5 version lacks:

1. No support for the special coloring of empty pattern slots or the pattern currently being edited. A low priority.
2. A reduced-functionality options/preferences dialog, lacking an editor for keystroke commands.
3. Keystroke support is not as comprehensive as the Linux version.
4. Modified support for multi-wid. The Qt interface allows for any number of external live frames. An arbitrary number of sets (banks) can be shown at once, each in its own window. The window in focus is the active set.

We are incorporating existing *Sequencer64* user-interface features into this Qt 5 version as time and bug-fixes allow. With version 0.96.0, we have improved the Qt/Windows version by adding the following features:

1. An external pattern editor has been added that matches the Gtkmm-2.4 *Sequencer64* pattern editor. It supports background sequences, chord entry, event merging and extension, scales, snap, etc. The tabbed pattern editor is still in place, but, due to space, it will never support all of the features of the external window.
2. The LFO (low-frequency oscillator) window can be called up from the external pattern editor.
3. The song/performance/trigger editor has been improved, and also made available as an external window.
4. The pattern and song editors now have better support for zooming via keystrokes.
5. A playlist tab has been added to support the new play-list functionality. This tab is still a work in progress. See section section 12 "Sequencer64 Play-Lists" on page 141.

Some ill-advised features from *Sequencer64* will not be migrated. Ultimately, we want to replace the Gtkmm-2.4 user-interface completely. But this is awhile down the road. There are a number of alternate versions of *Sequencer64* using PortMidi:

1. A Gtkmm-2.4 user-interface and PortMidi. This version is Linux-only, and is useful for debugging the internal PortMidi code without worrying about the Qt 5 user-interface.
2. A Qt 5 user-interface and PortMidi on Linux. This version is useful for people who like to experiment, and for debugging Qt 5 and PortMidi issues at the same time.
3. A Qt 5 user-interface and PortMidi on Windows. This version is basically working, but takes some special setup. See section 13.2 "Sequencer64 Windows Setup" on page 152for details.

We also have code in place to support *Mac OSX*, but currently have no OSX system on which to build and test this code. **HELP WANTED!**

In the following sections, we cover the basic differences between the Gtkmm-2.4 and Qt 5 versions of the application, as well as how to set up *Sequencer64 for Windows*. We will not show all of the user-interface elements, but we will mention important differences one will find.

## 13.1 The Qt 5 User Interface

The Qt 5 version of *Sequencer64* has an executable name of `qpseq64` (Linux) or `qpseq64.exe` (Windows). It is also possible to build a Qt 5 version with the RtMidi interface (`qseq64`), and that will ultimately become the default version for Linux. To keep explanations simple, we will refer to the Qt 5 version of *Sequencer64*, using the *PortMidi* engine, as **qpseq64**. The Gtkmm-2.4/RtMidi version (Linux only) will be referred to as **seq64**.

Here is a screenshot of the main user-interface of *qpseq64*, with some of the patterns colored for emphasis.

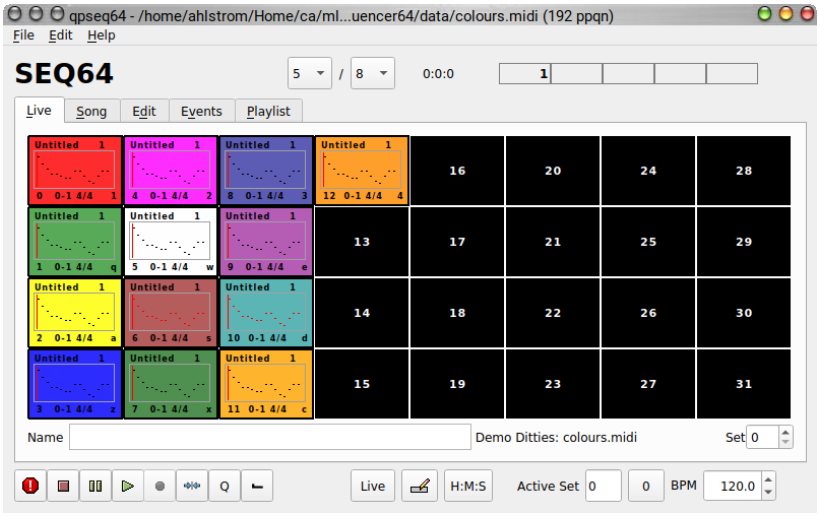



Figure 92: Qt 5 Main Window, Linux

Note the difference in layout from the Gtkmm-2.4 version. Also notice that some controls from the Gtkmm-2.4 version are missing. Some of these missing items will ultimately be added.

There are some issues, in our opinion, with the *Kepler34* coloring under various situations (e.g. muting and queuing). So we have adapted the `grid_style` option in the "usr" configuration file so that, if set to "1", the slots look more like the Gtkmm-2.4 version of the slots:

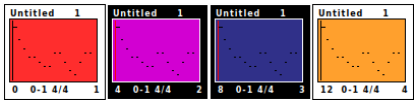


Figure 93: Qt 5 Slots in Gtk Style

In this style, the outer border is either white (muted) or black (unmuted). Also, queuing is indicated in the legacy manner (the central box is colored gray), rather than with a black frame drawn around the slot. Choose which style you like, and send feedback about what changes would be desirable for either style.

Apart from different window decoration, the Windows version looks the same. Coding an application for both Windows and Linux is very instructive, and can improve the overall code significantly. It certainly forced us to add some new features to handle the operating-system differences!

### 13.1.1 Qt 5 Live Slot Menu

The Qt 5 version of the right-click pattern/slot menu is slightly different from the Gtkmm-2.4 version.

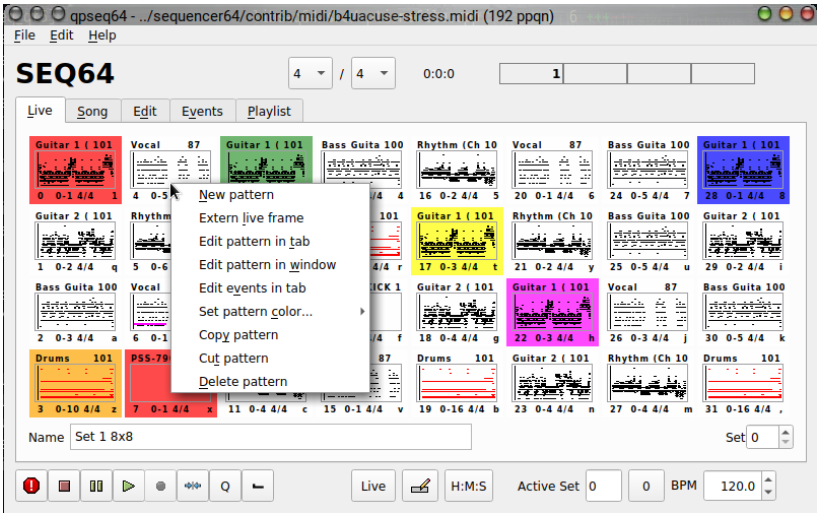



Figure 94: Qt 5 Slot Menu

It lets one add an external window for the pattern slots (live-frame), and lets one open a pattern editor in a tab or in an external window. It lets one edit events textually in a tab. Here are the menu entries to describe:

1. **New pattern**
2. **External live frame**
3. **Edit pattern in tab**
4. **Edit pattern in window**
5. **Edit events in tab**
6. **Copy pattern**
7. **Cut pattern**
8. **Delete pattern**

Let's describe them.

**1. New pattern.** Creates a new pattern in the slot. If there is already a pattern there, the user is prompted about a sequence already present, with a yes/no question about overwriting it and creating a new blank sequence.

**2. External live frame.** With Qt 5, we decided not to reimplement the Gtkmm-2.4 version's "multiwid" features, which allows one to specify a matrix of screen-sets. This option creates an additional "Live" frame, which is external to the Live frame tab. This live-frame shows the set/bank corresponding to the slot number (tricky!) over which the menu was opened. This won't work if used on a higher set. When an external live-frame has focus, it represents the *active set/bank*. The active set is shown in the main window in the **Active Set** field, just to be clear.

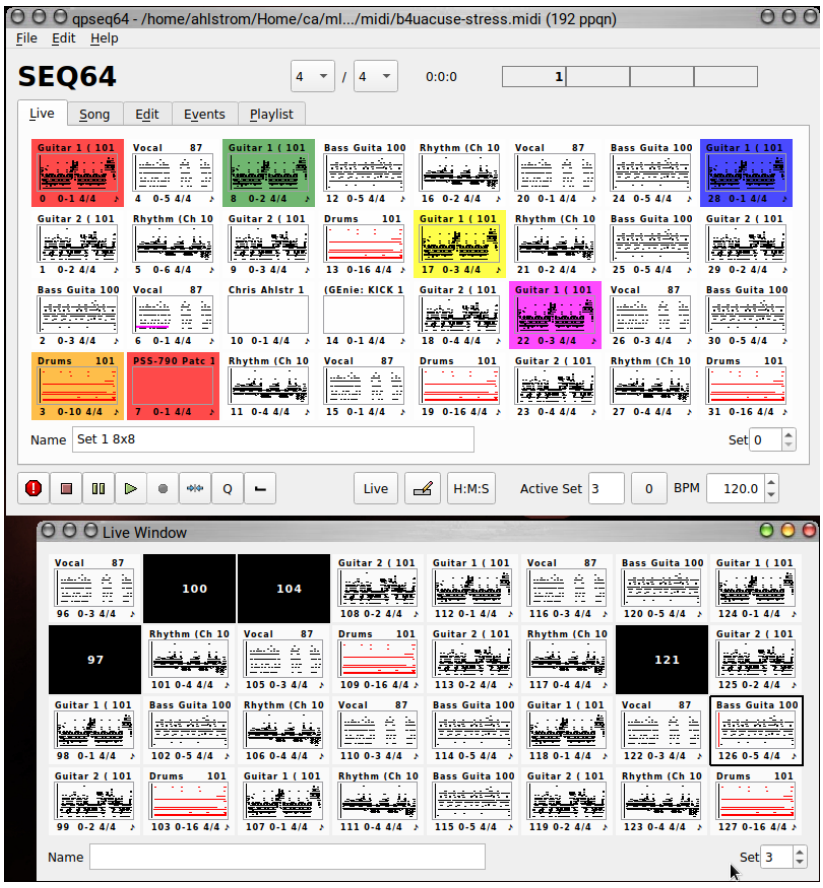



Figure 95: Qt 5 External Live Frame

**3. Edit pattern in tab.** Sets up a pattern editor in the "Edit" tab. This editor is not as versatile as the external-window pattern editor described in the next section.

**4. Edit pattern in window.** Sets up a pattern editor a separate window. This pattern editor works a lot like the Gtkmm-2.4 version, though it does not support the "fruity" mode of editing.

**5. Edit events in tab.** Sets up an event editor in the "Edit" tab. This editor is a basic editor and mostly useful for finding and deleting unwanted events, or adding events that are not otherwise available.

**6. Copy pattern.** Copies the selected pattern.

**7. Cut pattern.** Cuts (and copies) the selected pattern.

**8. Delete pattern.** Deletes the selected pattern.

### 13.1.2 Qt 5 Live Tab

The **Live** tab of *qpseq64*, as shown above, is very similar to the *seq64* version. It supports the varisets mode (the alternate row-by-column sizes of *seq64*). In addition, external versions of the Live frame can be instantiated outside of the main window, so that the user can view and work with a number of set/banks at the same time.

The size of this window can be changed to a certain degree via the command-line option `--option scale=x.y`, where `x.y` can range from 0.4 to 3.0. This can be made permanent via the `window-scale`

setting in the "usr" file. For example, these two options should be used together: `-o sets=8x8 -o scale=1.5`. Experiment with varying values for these options.

In fact, all of the tabs except for the **Events** and **Playlist** tabs will expand properly to use additional space if the application is resized, whether by a command-line option or by dragging a corner of the window.

### 13.1.3 Qt 5 Song Tab

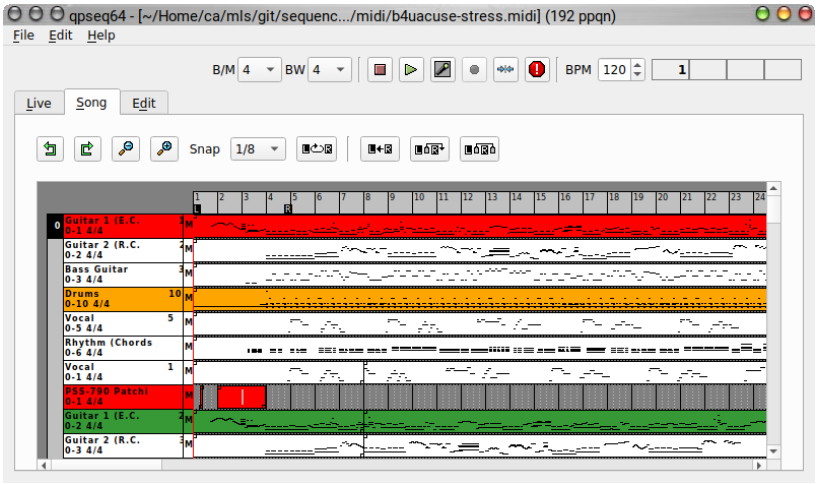



Figure 96: Qt 5 Song Window, Linux

This window is very similar to the *seq64* version. Additional features may be added as time goes on. In addition, this frame can be opened in its own external window by pressing the "Song Editor" button at the bottom of the main window.

### 13.1.4 Qt 5 Edit Tab

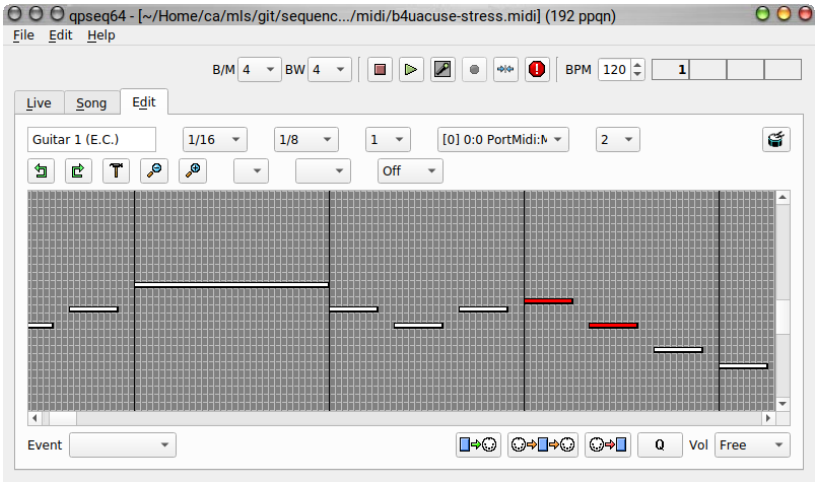



Figure 97: Qt 5 Edit Window, Linux

This version of the pattern editor is a bit behind *seq64* for features. There really isn't enough space in the main window for all the features of the *Sequencer64* pattern editor. Note that the event-data editor panel is not visible here. One must scroll down to the bottom of the window to see it and use it. Note that that the **LFO** window is not accessible here. Finally, note that the note heights and vertical grid spacing are modifiable via a key-height option, which is a feature the Gtkmm-2.4 user-interface does not offer.

With version 0.95.1, the pattern/slot popup menu has an option to open the pattern in an external window that has a lot more features than the pattern editor in the tab. It also has a different scrolling mechanism, more like that of *seq64*. Note that only one of the tab/external pattern editors can be shown at the same time for a given pattern. This pattern editor is pretty much identical to the Gtkmm-2.4 version, except that it does not provide a "fruity" mode of mouse usage. Sorry.

### 13.1.5 Qt 5 Events Tab

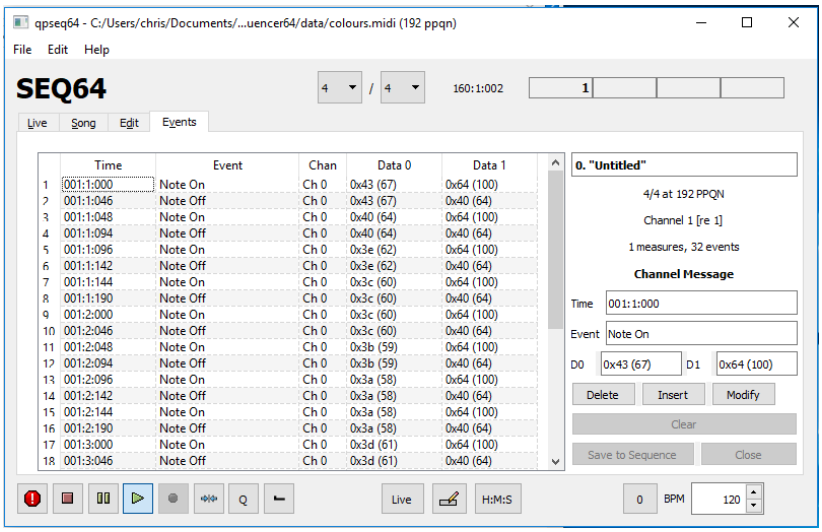



Figure 98: Qt 5 Events Window

This window works a lot like section 6 "Event Editor" on page 85; please see that section for user instructions. Please remember that, at the present time, this view is meant only for viewing events and making minor tweaks to a pattern, such as eliminating problematic events. It is not intended to be a full-featured event editor. In particular, note that editing within each cell in the event table is not supported. Event editing must be done in the right panel. An event must be explicitly selected by clicking on it, before it can be modified or deleted.

Also, if one adds or modifies an event to show up an the end of the pattern, by modifying the time stamp, the event will appear there, but the user-interface will remain at the current event.

### 13.1.6 Qt 5 Playlist Tab

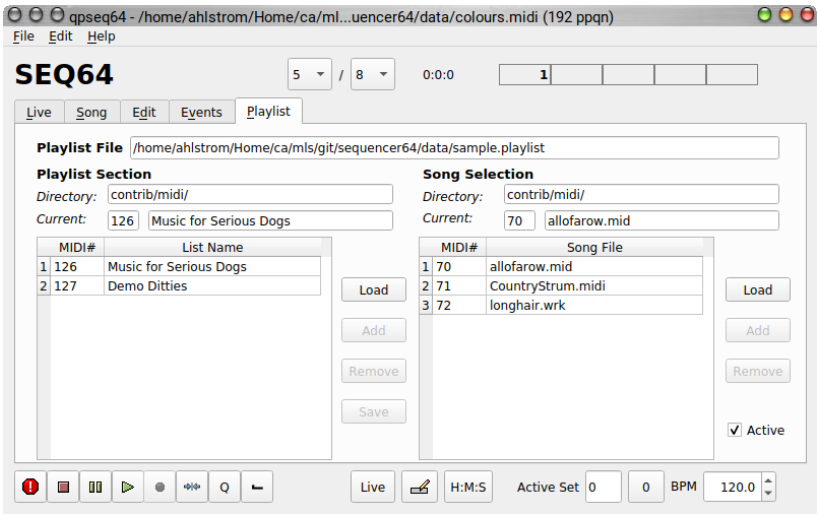



Figure 99: Qt 5 Playlist Window

The play-list functionality can be viewed in this tab. Currently, this tab only shows the playlist items. One can select a playlist, and view the song-titles in it, but editing and loading the play-lists and songs does not yet work. One must edit the play-list file manually for now, and load it from the **File / Open Playlist** menu.

Also note that the play-list title can be seen in the main live frame, and that play-lists and songs-can be selected with the arrow keys in that window.

See section section 12 "Sequencer64 Play-Lists" on page 141; there are many things that can be done with the playlist feature, including controlling it via MIDI events.

### 13.1.7 Qt 5 Edit / Preferences

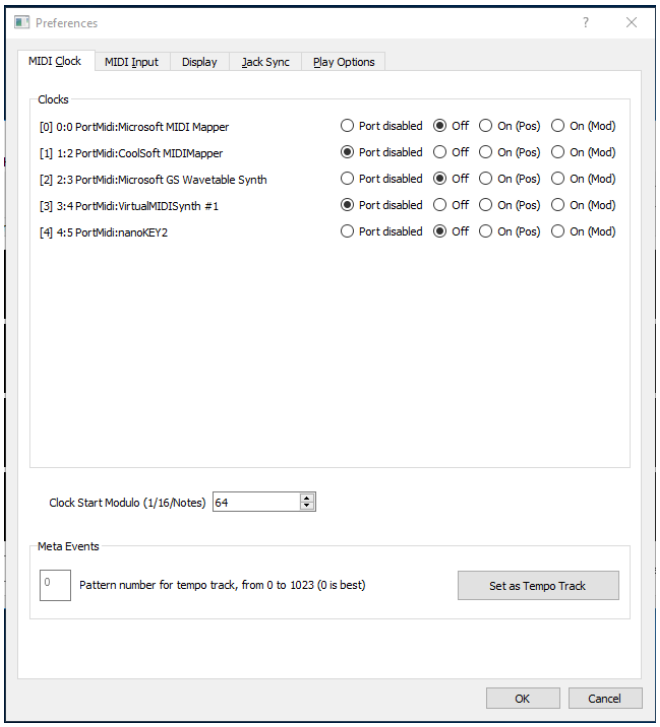



Figure 100: Qt 5 Clock Preferences, Windows

In a newer version of this document, we will show the other preferences tabs. There are many tabs we still need to add to this dialog. In particular, be sure to go to the **MIDI Input** tab and make sure that your desired input device (e.g. a MIDI keyboard) are shown and are enabled. Since the Qt GUI pretty much hardwires the keystrokes used for various functions such as muting and queuing, we might not provided a keystroke mapping editor. To be determined. In any event, many of the keystrokes configured in the "rc" file are supported, but one might have to edit the "rc" file directly, keeping in mind that it uses Gtkmm-2.4 conventions for keystrokes. *Linux* users can edit the keystrokes and copy the "rc" configuration into the Qt 5 "rc" file, `qpseq64.rc`.

## 13.2 Sequencer64 Windows Setup

*Sequencer64 for Windows* (*qpseq64*) can be installed as a portable Zip package anywhere the user desires. The package is self-contained, and contains all the the DLLs needed to run the program. *qpseq64* can also be installed via an NSIS-based installer, such as `sequencer64_setup_0.96.0-0.exe`.

### 13.2.1 Sequencer64 Windows Issues

When first starting *qpseq64* on *Windows*, one might experience some issues. One issue is that the *Microsoft MIDI Mapper*, rumored to be removed in *Windows 8* and beyond, is still detected by the internal PortMidi library used in *qpseq64*. Another issue is that the built-in *Microsoft* wave-table synthesizer might not be accessible.

We installed the *CoolSoft MIDIMapper* ([4]) and *VirtualMIDISYnth* ([5]) to try to get around these issues, and tried to turn off the `Windows System Sound` setup of *"Allow applications to restrict access to this device."* But we still had inaccessible devices, and the resulting errors would cause *qpseq64* to abort. So we had to spend a lot of time adding support for the disabling of inaccessible ports, and saving and restoring the "rc" setup properly in the face of device-access errors.

Here is some output logging on our Windows, generated using the `qpseq64` command-line option `-o log=virtualmidi.log`, which dumps the log file into

```
C:/Users/chris/AppData/Local/sequencer64/virtualmidi.log
```

```
qpseq64
C:/Users/chris/AppData/Local/sequencer64/virtualmidi.log
2018-05-13 09:06:58
[MIDIMAPPER] 'mapper in : midiInGetDevCaps() error for device 'MIDIMAPPER':
   'The specified device identifier is out of range'
pm_winmm_general_inputs(): no input devices
PortMidi MMSystem 0: Microsoft MIDI Mapper output opened
PortMidi MMSystem 1: CoolSoft MIDIMapper output closed
PortMidi MMSystem 2: Microsoft GS Wavetable Synth output opened
PortMidi MMSystem 3: VirtualMIDISynth #1 output closed
[Opened MIDI file,
   'C:/Users/chris/Documents/Home/sequencer64/data/b4uacuse-gm-patchless.midi']
[Writing rc configuration C:/Users/chris/AppData/Local/sequencer64/qpseq64.rc]
PortMidi call failed: [-1] 'Bad pointer'
PortMidi call failed: [-1] 'Bad pointer'
Begin closing open devices...
Warning: devices were left open. They have been closed.
```

By disabling the "closed" devices in the `qpseq64.rc` file, we are able to start. Here are the settings:

```
5    # number of MIDI clocks/busses
# Output buss name: [0] 0:0 PortMidi:Microsoft MIDI Mapper
0 0    # buss number, clock status
# Output buss name: [1] 1:2 PortMidi:CoolSoft MIDIMapper
1 -1    # buss number, clock status
# Output buss name: [2] 2:3 PortMidi:Microsoft GS Wavetable Synth
2 0    # buss number, clock status
# Output buss name: [3] 3:4 PortMidi:VirtualMIDISynth #1
3 -1    # buss number, clock status
```

These settings tell *Sequencer64* to ignore output busses 1 and 3, in order to allow it to start without error.

We still have some minor issues at start up and at exit, but are now able to play a tune on the wavetable synthesizer using the `--bus 2` option, which forces output to PortMidi buss 2, "Microsoft GS Wavetable Synth".

If an error occurs at start-up, a file called `erroneous.rc` is written to the configuration directory, and can be examined for issues with ports.

### 13.2.2 Sequencer64 Windows Configuration Files

When you first run *qpseq64* on *Windows*, it will create a new configuration file, with inaccessible devices denoted in the `[midi-clock]` section of `C:/Users/username/AppData/Local/sequencer64/qpseq64.rc` by a `-1` value.

On *Linux*, the normal directory location of the *Sequencer64* configuration files is

```
/home/username/.config/sequencer64
```

There are various configuration file names, depending on the name of the built application:

```
sequencer64.rc      The RtMidi Native ALSA/JACK version.
sequencer64.usr     The same, but for user-interface settings.
seq64portmidi.rc    The PortMidi Gtkmm 2.4 version.
seq64portmidi.usr   The same, but for user-interface settings.
qseq64.rc           The RtMidi Qt 5 version for Linux.
qseq64.usr          The same, but for user-interface settings.
qpseq64.rc          The PortMidi Qt 5 version for Windows.
qpseq64.usr         The same, but for user-interface settings.
```

On *Windows*, the conventional configuration file location is different:

```
C:/Users/username/AppData/Local/sequencer64
```

The files are:

```
qpseq64.rc          The PortMidi Qt 5 version for Windows.
qpseq64.usr         The same, but for some user-interface settings.
```

Typical of *Microsoft*, access to the `AppData` directory is not obvious... "we cannot have users monkeying with the configuration". To access `AppData`, highlight the user-name directory (e.g. `C:/Users/chris`, then append "AppData" to the end of it. Voila! It is now visible in *Windows Explorer*. It is a Windows *thang*. Then click on the `sequencer64` directory. Open `qpseq64.rc` and see what is in it (scroll down a bit):

```
[midi-clock]
2    # number of MIDI clocks/busses
# Output buss name: [0] 0:0 PortMidi:Microsoft MIDI Mapper
0 0  # buss number, clock status
# Output buss name: [2] 1:1 PortMidi:Microsoft GS Wavetable Synth (virtual)
1 0  # buss number, clock status
# Output buss name: [3] 1:1 PortMidi:nanoKEY2
2 0  # buss number, clock status
```

```
[midi-input]
1    # number of input MIDI busses
# The first number is the port number, and the second number
# indicates whether it is disabled (0), or enabled (1).
# [1] 0:1 PortMidi:nanoKEY2
0 1
```

These settings can be accessed via **Edit / Preferences / MIDI Clock** and **Edit / Preferences / MIDI Input** to alter the ports accessible, in the *Windows* version of *Sequencer64*. The operating system may have some devices locked out, though. Generally, the device needs to have a "System Sound" setting that allows an application to grab exclusive access to the device. For now, do a web search for this error message, if you have issues:

```
'The specified device is already in use.  Wait until it is free, and then
try again.'
```

# 14 Sequencer64 Man Page

This section presents the contents of the *Sequencer64* man page, but not exactly in *man* format. Also, an item or two are shown that somehow didn't make it into the man page, and minor corrections and formatting tweaks were made. For example, we replaced the underscore with the hyphen in the names of some options. The legacy Seq24 options, which use underscores or are missing the option hyphen, are still unofficially supported.

`$HOME/.config/sequencer64/sequencer64.rc` holds the "rc" settings for *Sequencer64*.

`$HOME/.config/sequencer64/sequencer64.usr` holds the "user" settings for *Sequencer64*.

But the old style names are used for the "legacy" mode. See the `--legacy` option below. Here is the basic command line:

```
sequencer64 [OPTIONS] [FILENAME]
seq64 [OPTIONS] [FILENAME]
```

*Sequencer64* accepts the following options, plus an optional name of a MIDI file. Please note that many of the options are 'sticky'. If they are used on the command-line, their settings are saved to the configuration files when *Sequencer64* exits.

`-h --help`
Display a list of all command-line options, then exit.

`-v --version`
Display the program version, then exit.

`-H --home [directory]`
**New:** Change the "home" directory from `.contrib/sequencer64` (always relative to `$HOME`. This option causes the `sequencer64.rc` and `sequencer64.usr` files to be loaded from or saved to a different directory. Format: `--home dirname`.

`-l --legacy`
**New:** Save the MIDI file in the old Seq24 format, as unspecified binary data, instead of as a legal MIDI

track with meta events. Also read the configuration, if provided, from the "legacy" `~/.seq24rc` and `~/.seq24usr` files.

The user-interface will indicate this mode with a small text note. This mode is also used if *Sequencer64* is invoked as the `seq24` command (one can create a soft link to the sequencer64 executable to make that happen).

`-b --bus [buss]`
**New:** Supports modifying the buss number on all tracks when a MIDI file is read. All tracks are loaded with this buss-number override. This feature is useful for testing, making it easy to map the MIDI file onto the system's current hardware/software synthesizer setup. Also note that this option applies the MIDI buss override to any new sequences, as well.

Format: `--bus bussnumber` or `--buss bussnumber`.

Most of the time, one will want to set this value to -1.

`-q --ppqn [ppqn]`
**New:** Supports modifying the PPQN value of Sequencer64, which is defaults to a value of 192. This setting should allow MIDI files to play back at the proper speed, and be written with the new PPQN value. This feature is basically done. One can load MIDI files of arbitrary PPQN, and they play normally and look normal in the editor windows. They can also be saved, and reloaded with the new PPQN value. Format: `--ppqn ppqnnumber`. The *ppqnnumber* can range from 32 to 19200. It can also be set to 0... in this case, *Sequencer64* uses the PPQN from the loaded file as its internal PPQN value.

`-L --lash`
**New:** If LASH support is compiled into the program, this option enables it. If LASH support is not compiled into the program, this option will not be shown in the output of the –help option.

`-n --no-lash`
**New:** If LASH support is compiled into the program, this option disables it, even if the default or configuration file set it. If LASH support is not compiled into the program, this option will not be shown in the output of the –help option.

`N/A --file [filename]`
Load a MIDI file on startup. **Bug:** This option does not exist. Instead, specify the file itself as the last command-line argument.

`-m --manual-alsa-ports`
*Sequencer64* won't attach the system's existing ALSA ports. Instead, it will create is own set of input and output busses/ports.

`-a --auto-alsa-ports`
*Sequencer64* will attach the system's existing ALSA ports. This variant is useful for overriding the rc configuration file.

`-r --reveal-alsa-ports`
**New:** *Sequencer64* will show the names of the ALSA port that the system defines, rather than the names defined in the 'user' configuration file.

`-R --hide-alsa-ports`
*Sequencer64* will show the names of the ALSA port that the 'user' configuration file define, rather than the names defined by ALSA.

`-A --alsa`
*Sequencer64* will not run the JACK support, even if specified in the configuration file. The configuration options are sticky (they are saved), and sometimes they aren't what you want to run.

`-s --show-midi`
Dumps incoming MIDI to the screen.

`-p --priority`
Runs at higher priority with a FIFO scheduler.

`N/A --pass-sysex`
Passes any incoming SYSEX messages to all outputs. Not yet supported.

`-i --ignore [number]`
Ignore ALSA device [number].

`-k --show-keys`
Prints pressed key value.

`-K --inverse`
Changes the color scheme for the sequence editor and performance editor piano rolls. It basicially inverts the colors. It can be considered kind of a "night mode".

`-X --playlist [filename]`
This option loads the given file-name as a play-list file. See section 12 "Sequencer64 Play-Lists" on page 141. This file provides one or more play-list entries, each providing a list of one or more songs. Once loaded, the user can use the four arrow keys to move between play-lists and the songs in each play-list. The play-list entries are also controllable via MIDI control values set in the "rc" file. See the Sequencer64 manual for more information. Note that, once set, this option is, by default, saved in the "rc" file.

`-x --interaction-method [number]`
Select the mouse interaction method. 0 = seq24 (the default); and 1 = fruity loops method. The latter does not completely support all actions supported by the Seq24 interaction method, at this time.

The following options will not be shown by –help if the application is not compiled for JACK support.

`-j --jack-transport`
*Sequencer64* will sync to JACK transport.

`-J --jack-master`
*Sequencer64* will try to be JACK master.

`-C --jack-master-cond`
JACK master will fail if there is already a master.

`-M --jack-start-mode [x]`
When *Sequencer64* is synced to JACK, the following play modes are available: 0 = live mode; and 1 = song mode, the default.

`-S --stats`
Print statistics on the command-line while running. Not available unless this option has been compiled in at build time, which can be determined by using the `--version` option.

`-U --jack-session-uuid [uuid]`
Set the UUID for the JACK session.

`-u --user-save`
Save the "user" configuration file when exiting Sequencer64. Normally, it is saved only if not present in the configuration directory, so as not to get stuck with temporary settings such as the –bus option. Note that the "rc" configuration option are generally also saved. But see the "auto-option-save" directive in the "rc" file. It is new with version 0.9.9.15.

`-f --rc filename`
Use a different "rc" configuration file. It must be a file in the user's $HOME/.config/sequencer64 directory or the directory specified by the –home option. Not supported by the –legacy mode. The '.rc' extension is added if no extension is present in the filename.

`-F --usr filename`
Use a different "usr" configuration file. It must be a file in the user's $HOME/.config/sequencer64 directory or the directory specified by the –home option. Not supported by the –legacy mode. The '.usr' extension is added if no extension is present in the filename.

`-c --config basename`
Use a different configuration file base name for the 'rc' and 'usr' files. For example, one can specify a full configuration for "testing", for "jack", or for "alsa", to set up `testing.rc` and `testing.usr`, `jack.rc` and `jack.usr`, `alsa.rc` and `alsa.usr`.

`-o --option opvalue`
Provides additional options, since the application is running out of single-character options. The `opvalue` set supported is:

- `daemonize`. Makes the `seq64cli` application fork to the background.
- `no-daemonize`. Makes the `seq64cli` application run in the foreground, where it is easy to see the informational output written to the console.
- `log=filename.log`. Reroutes standard error and standard output messages to the given log-file. This file is located in the directory for the "rc" and "usr" files (which can be altered via the `-H/--home` directory option). If this file is already present, additional log information is appended to it. If the "=filename.log" portion is missing, the log-file name in the `[user-options]` section of the "usr" file is used as the default log-file, if this file-name is not empty.
- `wid=3x2,i`. Not supported in the Qt version. Instead, an arbitrary number of external live-frame windows can be created. Provides for multiple main windows ("mainwids") to be shown in a big grid, so that multiple sets can be viewed at the same time. The default is to show the usual single mainwid. The first number specified is the row count, which can range from 1 to 3. The second number specified is the column count, which can only be 1 or 2. The third parameter starts with 't' ("true") to indicate that the multiple sets will be controlled by a single set spinner, which is the default. Specifying 'f' ("false") or 'i' ("indep") will show one set spinner for each mainwid. Note that it is possible, in this mode, to show the same set in two different mainwids, but this is not recommended, as there are minor unavoidable issues with that. Also note that the format for this command line option is very strict, no deviations or added spaces. Finally, to save these options to the "usr" file, add the `--user-save` option to the command line. In that file, the options modified are `block_rows` and `block_columns`.
- `sets=8x8`. This option, informally known as "variset", allow some changes in the set size and layout from the default 4x8 = 32 sets layout. **Warning:** *seq24* was hardwired for supporting 32 patterns per set, and there are still places where that is true. Thus, consider this option to be experimental. To save these options to the "usr" file, add the `--user-save` option to the command line. In that file, the options modified are `mainwnd_rows` and `mainwnd_cols`.
- `scale=x.y`. This option scales the main window by the factor x.y, which can range from 0.75 to 3.0. This makes the main window larger, and most of its contents larger. It does not currently scale the fonts used in the display. A scale factor of 2.5 is about the maximum useful value. If multi-wid is active, then you probably need a larger monitor to use this factor.

`$HOME/.config/sequencer64.rc` holds the main configuration settings for all versions of Sequencer64. If it does not exist, it will be generated when Sequencer64 exits. If it does exist, it will be rewritten with

the current configuration of Sequencer64. Many, or most, of the command-line options are "sticky", in that they will be written to the configuration file.

`$HOME/.config/sequencer64.usr` stores the MIDI-configuration settings and some of the user-interface settings for Sequencer64. If it does not exist, it will be generated with a minimal configuration when Sequencer64 exits. If it does exist, it will be rewritten with the current configuration of Sequencer64, but *only* if the `--user-save` option was provided on the command-line. Note that the `--legacy` option causes the old configuration-file names (`.seq64rc` and `.seq64usr` in the `$HOME` directory) to be used.

The current Sequencer64 project homepage is a git repository at

`https://github.com/ahlstromcj/sequencer64.git`.

Up-to-date instructions can be found in the project at

`https://github.com/ahlstromcj/sequencer64-doc.git`.

The old Seq24 project homepage is at `http://www.filter24.org/seq24/` the new one is at `https://edge.launchpad.net/seq24/`. It is released under the GNU GPL license. Sequencer64 is also released under the GNU GPL license.

*Sequencer64* was written by Chris Ahlstrom ahlstromcj@gmail.com (with a fair amount of help). *Seq24* was written by Rob C. Buse seq24@filter24.org and the *Sequencer64* team.

This manual page was written by Dana Olson `mailto:seq24@ubuntustudio.com` with additions from Guido Scholz `mailto:guido.scholz@bayernline.de` and Chris Ahlstrom `mailto:ahlstromcj@gmail.com`.

# 15 Sequencer64 Headless Version

*Sequencer64* can be built as a command-line application, as described in section 17 "Building Sequencer64" on page 165. That is, it can be run from the command-line, but has no user interface. It can also be instantiated as a Linux daemon, for totally headless usage. Because there is not a lot of visibility into a headless process, the setup for `seq64cli` is a little complex, and the musician must get used to blind MIDI control.

## 15.1 Sequencer64 Headless Setup

The first step in setting up a headless `seq64cli` session is to make sure that the GUI version (`seq64`) works as expected. The GUI and headless configurations need to do the following:

1. Access the correct inputs, especially a keyboard or pad controller that can be used for controlling the sequencer via MIDI, as well as inputing notes.
2. The MIDI input must be configured with some `[midi-control]` values, so that the headless sequencer can do things like stop and start playback, select the next playlist or song, or change other sequencer controls. Also note that an alternative is to provide a `[midi-control-file]` specified in the "rc" file.
3. Access the desired outputs, in order to play sounds. This can sometimes be tricky, because *Sequencer64* can route all patterns to the same output, or can let the patterns decide the outputs for themselves.
4. Use the desired play-list. The headless sequencer can only select songs to play via a pre-configured play-list.

Sometimes odd problems, such as the output synthesizer not working, not appearing in the list of outputs, can prove a real puzzle. Here are the steps used in this test; adapt them to your setup. For simplicity, JACK is not running, and so ALSA is in force.

**First**, after booting, plug in the MIDI keyboard or MIDI control pad. Our example here will use the *Korg nanoKEY2* keyboard.

**Second**, start the desired (software) synthesizer. We will use the synth *Yoshimi*, with a stock setup from our "Yoshimi Cookbook" project. The order of starting the keyboard/pad and the synthesiser will alter the port numbers of these items. Best to do things in the same order every time... be consistent.

**Third**, to validate the setup, run a command from the command-line such as:

```
seq64 -b 2 -v -X data/sample.playlist
```

The buss number ("2") may need to be different on your setup to get sound routed to the correct synthesizer. Also, the path to the playlist might need to be an absolute path; normally playlists are stored in the `HOME/.config/sequencer64` directory and accessed from there. Verify that the main window shows the playlist name, and that the arrow keys modify the play-list or song selection. If that works, verify that the MIDI keyboard or pad controller works to change the selection. Verify that the current song plays through the synthesizer that was started. If this setup works (MIDI controls have the proper effect and the tunes play through the synthesizer), proceed to the next step.

**Fourth**, edit the `sequencer64.rc` and `seq64cli.rc` files as described below so that the former's settings of `[midi-clock]` or `[midi-clock-file]` are entered into the latter "rc" configuration. For our testing, we use a separate MIDI-control file (`nanomap.rc`), which is set up in the main "rc" file to be read via a `[midi-clock-file]` setting. The `nanomap.rc` file sets up our *nanoKEY2* as shown in this figure:

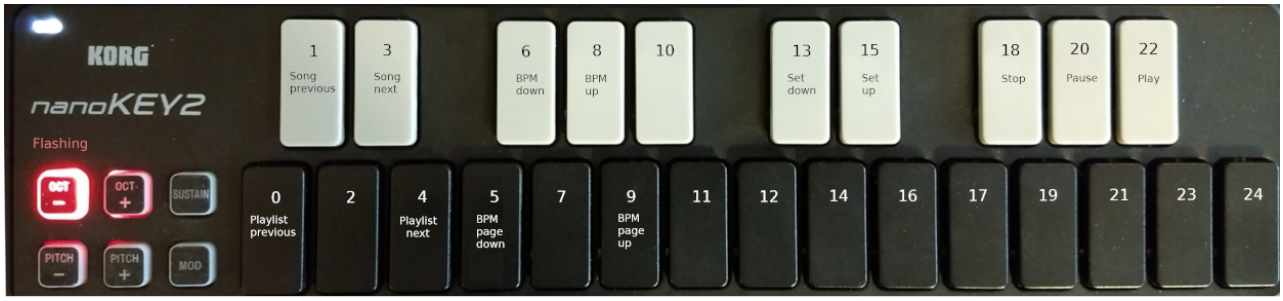



Figure 101: Sample nanoKEY2 Control Setup

In this figure, the **OCT -** button on the nanoKEY2 is pressed until it is flashing (not seen in the figure). This means that the lowest note on the nanoKEY2 is MIDI note 0, the lowest note possible. With these settings, the playlists and songs can be loaded and then played and paused. The `seq64cli.rc` file is edited so that the rather large `[midi-control]` section is replaced by the following:

```
[midi-control-file]
nanomap.rc    # (/home/ahlstrom/.config/sequencer64/nanomap.rc)
```

The `nanomap.rc` file is included in the "data" directory of the source-code package.

**Fifth**, test the command-line *Sequencer64* by running the following command (your setup might vary) on the command line:

```
$ ./Seq64cli/seq64cli -b 2 -X data/sample.playlist} -v
```

There is a play-list option to automatically unmute the sets when a new song is selected. If set, then the first song should be ready to play. If it plays, and the play-list seems to work (as indicated by the console output and the proper playback), then run `seq64cli` as a daemon:

```
$ ./Seq64cli/seq64cli -b 2 -X data/sample.playlist} -v -o daemonize
```

The keyboard controls and sound output should still work.

# 16 Concepts

The *Sequencer64* program is basically a loop-playing machine with a fairly simple interface. Before we describe this interface, it is useful to present some concepts and definitions of terms as they are used in *Sequencer64*. Various terms have been used over the years to mean the same thing (e.g. "sequence", "pattern", "loop", and "slot"), so it is good to clarify the terminology.

## 16.1 Concepts / Terms

This section doesn't provide comprehensive coverage of terms. It covers terms that puzzled the author at first or that are necessary to understand the *Sequencer64* program.

### 16.1.1 Concepts / Terms / armed

An armed sequence is a sequence (see section 16.1.19 "Concepts / Terms / sequence" on page 164) that will be heard. "Armed" is the opposite of "muted". Performing an *arm* operation in *Sequencer64* means clicking on an "unarmed" sequence in the patterns panel (the main window of *Sequencer64*). An unarmed sequence will not be heard, and it has a white background. When the sequence is *armed*, it will be heard, and it has a black background. A sequence can be armed or unarmed in three ways:

- Clicking or Shift-clicking on a sequence/pattern box.
- Pressing the hot-key for that sequence/pattern box.
- Opening up the Song Editor and starting playback; the sequences arm/unarm depending on the layout of the sequences and triggers in the piano roll of the Song Editor.

### 16.1.2 Concepts / Terms / bank

See section 16.1.17 "Concepts / Terms / screen set" on page 164. This term is *Kepler34*'s name for "screen set".

### 16.1.3 Concepts / Terms / buss (bus)

A *buss* (also spelled "bus" these days; https://en.wikipedia.org/wiki/Busbar) is an entity onto which MIDI events can be placed, in order to be heard or to affect the playback. A *buss* is just another name for port. See section 16.1.13 "Concepts / Terms / port" on page 163.

### 16.1.4 Concepts / Terms / export

A *export* in *Sequencer64* is a way of writing a song-performance to a more standard MIDI file, so that it can be played by other sequencers. An export collects all of the unmuted tracks that have performance information (triggers) associated with them, and creates one larger trigger for each track, repeating the events as indicated by the original performance.

### 16.1.5 Concepts / Terms / group

A *group* in *Sequencer64* is one of up to 32 previously-defined mute/unmute patterns in the active screen set. A group is a set of patterns, in the current screen-set, that can arm (unmute) their playing state together. Every group contains all 32 sequences in the active screen set. This concept is similar to mute/unmute groups in hardware sequencers. Also known as a "mute-group".

### 16.1.6 Concepts / Terms / loop

*Loop* is a synonym for *pattern* or *sequence*, when used in existing *Seq24* documentation. Each loop is represented by a box (pattern slot) in the Pattern (main) Window.

### 16.1.7 Concepts / Terms / measures ruler

The *measures ruler* is the bar at the top of the Pattern Editor and Song Editor windows that shows the numbering of the measures in the song. Left, right, or end markers can be dropped on this ruler to set durations to be played, looped, expanded, or collapsed.

Note: The original *Seq24* documentation calls this item the *bar indicator*.

### 16.1.8 Concepts / Terms / event strip

The *event strip* is the bar at the bottom of the Pattern Editor window that shows the location of events in the pattern. For Note On and Note Off events, it is shown in gray to warn the user to be careful in moving these events.

### 16.1.9 Concepts / Terms / muted

The opposite of section 16.1.1 "Concepts / Terms / armed" on page 161.

### 16.1.10 Concepts / Terms / MIDI clock

*MIDI clock* is a MIDI timing reference signal used to synchronize pieces of equipment together. MIDI clock runs at a rate of 24 ppqn (pulses per quarter note). This means that the actual speed of the MIDI clock varies with the tempo of the clock generator (as contrasted with time code, which runs at a constant rate). *Sequencer64* emits MIDI clock events, but does not (yet) follow MIDI clock.

### 16.1.11 Concepts / Terms / pattern

A *Sequencer64 pattern* (also called a "sequence" or "loop") is a short unit of melody or rhythm in *Sequencer64*, extending for a small number of measures (in most cases). Each pattern is represented by a box in the Patterns window.

Each pattern is editable on its own. All patterns can be layed out in a particular arrangement to generate a more complex song.

*pattern* is a synonym for *loop* or *sequence*. It is our preferred term.

### 16.1.12 Concepts / Terms / performance

In the jargon of *Sequencer64*, a *performance* is an organized collection of patterns. This layout of patterns is created using the Song Editor, sometimes called the "performance editor". This window controls the song playback in "Song Mode". The playback of each track is controlled by a set of triggers created for that track.

### 16.1.13 Concepts / Terms / port

A *port* is just another name for buss. See section 16.1.3 "Concepts / Terms / buss (bus)" on page 162. Each port can support 16 MIDI channels.

### 16.1.14 Concepts / Terms / pulses per quarter note

The concept of "pulses per quarter note", or PPQN, is very important for MIDI timing. To make it a bit more confusing, sometimes these pulses are referred to as "ticks", "clocks", and "divisions". To make it even more confusing, there are separate timing concepts to understand, such as "tempo", "beats per measure", "beats per minute", "MIDI clocks", and more.

While a full description of all these terms, and how they are calculated, is beyond the scope of this document, we will try to clarify the discussion when such confusion could be an issue.

### 16.1.15 Concepts / Terms / queue mode

To "queue" a pattern means to ready it for playback on the next repeat of a pattern. A pattern can be armed immediately, or it can be queued to play back the next time the pattern restarts. Pattern toggles occur at the end of the pattern, rather than being set immediately.

A set of queued patterns can be temporarily stored, so that a different set of playbacks can occur, before the original set of playbacks is restored.

The "keep queue" functionality allows the queue to be held without holding down a button the whole time. Once this key is pressed, then the hot-keys for any pattern can be pressed, over and over, to queue each pattern.

### 16.1.16 Concepts / Terms / replace

Replacement is a form of muting/unmuting. When the "replace" key is pressed while click a sequence, that sequence is unmuted, and all of the other sequences are muted.

### 16.1.17 Concepts / Terms / screen set

The *screen set* is a set of patterns that fit within the 8x4 grid of loops/patterns in the Patterns panel. *Sequencer64* supports multiple screens sets, up to 32 of them, and a name can be given to each for clarity. The Gtkmm-2.4 version of *Sequencer64* supports an 8x8 grid and 64 patterns per screen set.

### 16.1.18 Concepts / Terms / bank

The term *bank* comes from the *Kepler34* application. It means the same thing as *screen set*.

### 16.1.19 Concepts / Terms / sequence

*Sequence* is another synonym for *pattern*, used in some of the *Seq24* documentation. *Loop* and *track* are other synonyms. Each sequence is represented by a box (pattern slot) in the Patterns window. Each sequence can be layed out to repeat in various ways in the Song editor window.

Note that other sequencer applications) use the term "sequence" to apply to the complete song, and not just to one track or pattern in the entire song.

### 16.1.20 Concepts / Terms / snapshot

A *Sequencer64 snapshot* is simply a briefly preserved state of the patterns. One can press a snapshot key, change the state of the patterns for live playback, and then release the snapshot key to revert to the state when it was first pressed. (One might call it a "revert" key, instead.)

### 16.1.21 Concepts / Terms / song

A *song* is a collection of patterns in a specific temporal layout, as assembled via the Song Editor window. See 16.1.12. See 16.1.22.

### 16.1.22 Concepts / Terms / trigger

A *trigger* is a small data structure that indicates when a sequence should be played, and how much of the sequence (including repeats) should be played. A song performance consists of a number of sequences, each triggered in ways that the musician can lay out.

## 16.2 Concepts / Sound Subsystems

### 16.2.1 Concepts / Sound Subsystems / ALSA

*ALSA* is a sound and MIDI system for Linux, with components built into the Linux kernel. It is the main subsystem used by *Sequencer64*. The name of the library used to build *ALSA* projects is `libasound`. See reference [1].

### 16.2.2 Concepts / Sound Subsystems / PortMIDI

*PortMIDI* is a cross-platform API (applications programming interface) for MIDI. It seems to be used in the "portmidi" C++ modules included with the base source-code repository of *Seq24* available (for example) from Debian Linux. See reference [17] for the PortMIDI home page. Unfortunately, the code in the Debian package is not quite ready to build on Windows. We might fix that someday, though Windows is not a high priority.

The SubatomicGlue Windows port of *Seq24* (see reference [32]) bundles a version of the PortMIDI project with the source code for the port. It also provides a complete bundle of the other products (e.g. gtkmm 2.4) needed to build and run the project. (By the way, the Windows port is built with MingW, which provides the GNU compilers and tools. This is a good thing, as Visual Studio Community, though "free", is not "Free".)

### 16.2.3 Concepts / Sound Subsystems / JACK

*JACK* is a cross-platform (with an emphasis on Linux) API and infrastructure for making it easier to connect and reroute MIDI and audio event between various applications and hardware ports. See reference [8].

# 17 Building Sequencer64

The project packaging for *Sequencer64* is aimed at developers. But note that we have Debian packages, conventional tarballs, and *Windows* installers in the "Sequencer64 Packages" project ([31]). If one has packages built for other Linux distributions, let us know, and we can stick them there for others to use.

This section presents a how-to on building the various versions of *Sequencer64* from source code. It's actually pretty easy, with these instructions. And please note that there are many ways to build *Sequencer64*, each described in its own section.

## 17.1 Linux INSTALL Build

There are many build options. Some are modifiable via the normal GNU `configure` script method. Many more are modifiable by editing the source code to **#define** and **#undefine** certain macros. If you don't care about such options, start here. If you want to see what options are available, skip to section 17.2.1 ""configure" Options" on page 167, which has many details one can adjust. The project includes the conventional GNU `configure` script, in the tarball and in the git-cloned project. However, the the `bootstrap` script can be used to start from scratch, as in the following instructions:

1. Preload the dependencies, as listed in section 17.3 "Sequencer64 Build Dependencies" on page 168. If some are missing, the `configure` script will tell you, or, at worst, a build error will.
2. Check-out the desired branch, normally "master". Make a branch, if desired, to make changes. See the file `git.txt` in the `contrib/notes` project directory.
3. From the top project directory, run the commands:

```
$ ./bootstrap
$ ./configure
```

4. For debugging without libtool getting in the way, just run one of the following commands, which run the `configure` script, adding the `--enable-debug` and `--disable-shared` options to it. The bootstrap script is our start-over way of setting up GNU autoconf.

```
$ ./bootstrap --enable-debug
$ ./bootstrap -ed
```

5. Run the make command:

```
$ make
```

6. To install *Sequencer64*, become root and run:

```
# make install
```

Note that an alternate user-interface (Qt) and alternate MIDI engine (portmidi) can be configured and built. See section 17.4 "Linux Qt Builds" on page 169. Also note that one has to build the documentation and Debian packaging separately, they are not part of the default build.

Noted above is the `--enable-debug` option. The following command bootstraps and then configures for release mode, and greatly reduces the amount of compiler output:

```
$ ./bootstrap --enable-release
$ ./bootstrap -er
```

This script option runs:

```
$ ./configure --enable-silent-rules
```

It results in abbreviated output, which makes it easier to see warnings that might pop up. This anti-verbosity option can be overridden at "make" time:

```
$ make V=1
```

`V=0` is another way to quiet down the build. Note that the build can be sped up by telling `make` to use more cores. For example, if one has an 8-core system:

```
$ make -j 8
```

## 17.2 Options for Sequencer64 Features

*Sequencer64* comes with options for the `configure` command and options represented by definable macros in the source code.

### 17.2.1 "configure" Options

The following `configure` options can be specified on the command line:

**1.** `--enable-rtmidi`**.** This option can be bootstrapped directly using "`./bootstrap -er -rm`. This is currently the default build. It creates an executable, `seq64`. This version can do JACK input/output using the native API... no more need to use `a2jmidid` to bridge from ALSA to JACK. It provides a JACK client separate from the legacy JACK-transport client. `seq64` falls back to using ALSA if JACK is not running. Like ALSA, the JACK support has an auto-connect feature that can be disabled using the "manual" ("virtual-port") option.

We term this version "rtmidi" because we originally used the RtMidi ([18]) project as the basis for the native JACK support, but it does not quite fit the usage model of *Sequencer64*, so we heavily refactored it.

**2.** `--enable-qt`**.** This sets up a build *Sequencer64* that uses a Qt user-interface. This version can be build with the RtMidi or PortMidi engines. Try "`./bootstrap -er -rm -qt` or "`./bootstrap -er -pm -qt`.

**3.** `--enable-cli`**.** Bootstrapping directly: "`./bootstrap -er -cli`". It sets up a command-line build of `seq64` that has the program name `seq64cli`. This application must be controlled via MIDI controls set up in the [midi-control] section of the "rc" file. See section 10.2 ""rc" File / MIDI Control" on page 103, for information on those controls, which include start, stop, pause, play-list navigation, and other commands. A single song can be loaded via the last command-line argument. A number of songs can be loaded via a play-list. See section 12 "Sequencer64 Play-Lists" on page 141. There is an option to make the application fork into the background as a daemon.

**4.** `--enable-alsamidi`**.** Bootstrapping directly: "`./bootstrap -er -am`". This executable is basically the original version of *Sequencer64*, with the original executable name of `sequencer64`, which we're keeping around as a backup while we work the remaining nits out of the "rtmidi" version of the application.

**5.** `--enable-portmidi`**.** Bootstrapping directly: "`./bootstrap -er -pm`". This option builds the PortMIDI version of the application, `seq64portmidi`. This version works with *Linux*, but is meant as a way to pre-test the port *seq24* to Windows ("Google" for "seq24 subatomic glue"). This version is used for the *Windows* and *Mac* ports.

**6.** `--disable-highlight`**.** Undefines the `SEQ64_HIGHLIGHT_EMPTY_SEQS` macro, which is otherwise defined by default. If defined, the application highlights empty patterns by coloring them yellow. Applies only to the Gtkmm user-interface.

**7.** `--disable-lash`**.** Undefines the `SEQ64_LASH_SUPPORT` macro, but it is now undefined by default. *Linux* only.

**8.** `--disable-jack`**.** Undefines the `SEQ64_JACK_SUPPORT` macro, which is defined by default. Even if left defined, however, *Sequencer64* will still not use JACK support unless one specifies the various JACK options on the `sequencer64` command-line or turn them on in the "rc" configuration file, `~/.config/sequencer64/sequencer64.rc`. *Linux* only.

**9.** `--disable-jack-session`**.** Undefines the `SEQ64_JACK_SESSION` macro, which is defined if JACK support is defined, and the jack/session.h file is found. Again, this option, if left defined, can be affected by command-line options and options in the "rc" configuration file. *Linux* only.

**10.** `--disable-multiwid`**.** Undefines the `SEQ64_MULTI_MAINWID` macro. If this macro is defined (currently the default), then it is possible to show multiple sets in the main window. See section 3.4 "Patterns / Multiple Panels" on page 55, which describes this mode. Gtkmm only.

To summarize, these option macros define/undefine the following build macros:

- SEQ64_HIGHLIGHT_EMPTY_SEQS
- SEQ64_LASH_SUPPORT
- SEQ64_JACK_SUPPORT
- SEQ64_JACK_SESSION
- SEQ64_MULTI_MAINWID

A lot of options have become mandatory over the years, and are not discussed here. And there may be more macros not discussed. For the latest, see the INSTALL file in the source-code project.

### 17.2.2 Manually-defined Macros in the Code

As we have explored what *Seq24* does as we improve *Sequencer64*, we've found of things that might change the code for the worse in some minds. We mark those changes with macros. And sometimes we tried a change, but left it disabled. Look at those macros, modify them, and build the source code to one's preferences. If one does not see a macro described below, it means we need to catch up with the documentation.

The following items are not part of the configure script, but can be edited manually in the header file libseq64/include/seq64-features.h to achieve the desired settings:

**1.** SEQ64_EDIT_SEQUENCE_HIGHLIGHT. Defined in the perform module. Provides the option to highlight the currently-editing sequence in the main window view and in the song editor. If the sequence is muted, it is highlighted in black text on a cyan background. If it is unmuted, it is highlighted in cyan text on a black background. The highlighting follows whichever pattern editor or event editor has the focus. Gtkmm only.

**2.** SEQ64_USE_EVENT_MAP. Already defined in the event_list module. It enables the usage of an std::multimap, instead of an std::list, to store MIDI events. Because the code does a lot of sorting of events, using the std::multimap is actually a lot faster (especially under debug mode, where it takes many seconds for a medium-size MIDI file to load using the std::list implementation. But the std::multimap can be a limiting factor during playback. We use the list implementation and sort the container only after getting all the events loaded.

**3.** SEQ64_USE_MIDI_VECTOR. Defined in the seq64_features.h file. It enables the usage of an std::vector instead of std::list, to store MIDI data bytes. It provides the preferred alternative to the list for storing and counting the bytes of MIDI data. It stops the reversing of certain events due to the peculiarities of std::list.

**4.** SEQ64_USE_DEBUG_OUTPUT. Enable this macro in the globals.h header file, to see extra console output if the application is compiled for debugging. This macro can be activated only if PLATFORM_DEBUG is defined, which is taken care of by the build process. If set, this macro turns on extra console output for some modules.

Again, note the macro PLATFORM_DEBUG, defined in platform_macros.h if the application is built in debug mode.

## 17.3 Sequencer64 Build Dependencies

With luck, the following dependencies will bring in their own dependencies when installed. Build requirements:

- libgtkmm-2.4-dev (dev is the header-file package)
- libsigc++-2.0-dev
- libjack-jackd2-dev
- liblash-compat-dev (optional)

Runtime requirements:

- libatk-adaptor (and its dependencies)
- libgail-common (and its dependencies)
- valgrind (optional, very useful for debugging)
- gdb (optional, very useful for debugging)
- gprof and gcov (optional, very useful for debugging)

Build tools requirements:

- automake and autoconf
- autoconf-archive
- g++
- make
- libtool

Documentation requirements (very optional):

- doxygen and doxygen-latex
- graphviz
- texlive

Debian packaging (optional):

- debhelper
- fakeroot

## 17.4 Linux Qt Builds

The *Linux* version of *Sequencer64* can be built with the Qt user-interface, using either the PortMIDI or RtMIDI (preferred) MIDI engines.

```
$ ./bootstrap --enable-release -rm -qt
$ ./bootstrap -er -rm -qt
$ ./bootstrap --enable-release -pm -qt
$ ./bootstrap -er -pm -qt
```

Eventually, the Qt and RtMIDI combination will be the official version of *Sequencer64* for *Linux*. Our PortMIDI engine, while modestly improved over legacy PortMIDI, is not quite as complete as our RtMIDI-derived implementation.

## 17.5 Linux Qmake Build

We wanted to build *Sequencer64 for Windows* using GNU tools and the MSYS platform on either *Linux* (cross-compiling) or *Windows*. But it proved more foolproof to create the `*.pro` files necessary for *qmake* and be able to build on *Windows* and *Mac OSX*. This setup and build can be done through *Qt Creator*. The command-line setup is straightforward. From the *Sequencer64* directory, run the following commands, using either the build directory specified by *Qt Creator*, or making your own "shadow build" directory.

```
$ mkdir debug-build
$ cd debug-build
$ qmake -makefile -recursive "CONFIG += debug" ../sequencer64/qpseq64.pro
$ make
```

One can also use "CONFIG += release", or just leave that off entirely.

The `qpseq64.pro` file can also be loaded in the nice IDE, *Qt Creator*, and be configured, built, and debugged there. And one can tweak the GUI elements in that IDE.

## 17.6 Windows Qmake Build

The easiest option for a build on *Windows* is to install *Qt Creator* in its open-source edition. The executable name is `qt-unified-windows-x86-3.0.5-online.exe` or somesuch. Rather than navigate through the corporate pages, just go to https://download.qt.io/archive/online_installers/3.0/ and pick the latest version.

When installing, be sure to select at least the the 32-bit Mingw tools, including `mingw32-make.exe`, and `qmake.exe`. The *Windows* **PATH** must be modified to include the path to both executables, and the excutables `moc.exe`, `uic.exe`, `rcc.exe`, and `windeployqt.exe`. If the installation directory for *Qt Creator* is `ProgramFiles` (e.g. `C:/Program Files`), then add these directories to the user or system PATH:

```
%ProgramFiles%\Qt\5.11.1\mingw53_32\bin
%ProgramFiles%\Qt\Tools\mingw530_32\bin
```

Obviously, the version number might differ from "5.11.1". There is a build script in the `nsis` directory that automates the process of making the executable: `build_release_package.bat`. It also shows the steps needed to do a release build and create a *7-Zip* package, which include editing some macro variables with naming or version information. If one wants to do it manually, follow these steps. (We denote the DOS backslash path separator by "/", for our convenience.)

1. Using "git clone https://github.com/ahlstromcj/sequencer64.git" or the unpacking of a source-code tarball, create the `sequencer64` directory with all of the project files.
2. Change to the directory above this directory.
3. Create an empty "shadow" directory, e.g. `seq64-release`.
4. Change to this "shadow" directory. In that directory, run the following command: `qmake -makefile -recursive "CONFIG += release" ../sequencer64/qpseq64.pro`.

5. Next, run the following command and wait for the build to complete: `mingw32-make > make.log 2>&1`.
6. Open `make.log` and make sure there are no errors. Note that the output directory is inside the "shadow" directory and is called `Seq64qt5/release`.
7. Run the following command to copy necessary *Qt* DLLs to this directory: `windeployqt Seq64qt5/release`.
8. Create the `data` directory in `windeployqt Seq64qt5/release` and copy some data files:
   (a) `mkdir Seq64qt5/release/data`
   (b) `copy ../sequencer64/data/*.rc Seq64qt5/release/data`
   (c) `copy ../sequencer64/data/*.usr Seq64qt5/release/data`
   (d) `copy ../sequencer64/data/*.midi Seq64qt5/release/data`
   (e) `copy ../sequencer64/data/*.pdf Seq64qt5/release/data`
   (f) `copy ../sequencer64/data/*.txt Seq64qt5/release/data`
   (g) `copy ../sequencer64/data/*.playlist Seq64qt5/release/data`
9. Change to the `Seq64qt5` directory and run: `7z a -r qpseq64-release-package-0.96.1.7z release/*`

At this point, the `7z` file is useful as a "portable" package for the application. It can also be used to build the installer, as shown in the next section.

By the way, we have not tried using the Microsoft C++ compiler yet. If you try it and get the code to work, let us know!

### 17.6.1 Windows Installer

Here, we unpack the `7z` release package and then use *NSIS* to build the installer. These steps equire *7-Zip* to be installed and accessible from the DOS command-line, as `7z.exe`. Requires *NSIS 3* to be installed, unless one wants to use NSIS on Linux to build the installer (our preferred method). We build the installer in *Linux* using the *nsis* package. These instructions can be adopted to using the *Windows* GUI interface for *NSIS*.

1. Copy the file `7z a -r qpseq64-release-package-0.96.1.7z` to the top-level project directory, `sequencer64`.
2. Run `7z x qpseq64-release-package-0.96.1.7z` to extract the contents to the `release` directory.
3. Run the following commands:
   (a) `pushd nsis`
   (b) `makensis Seq64Setup.nsi`
   (c) `popd`
4. Verify that the installer works. It's name is like: `sequencer64_setup_0.96.1.exe`

The *bash* script `packages` does all this, plus creates the source-tarball and some other actions for the developers.

# 18 MIDI Format and Other MIDI Notes

## 18.1 Standard MIDI Format 0

*Sequencer64* can read and import SMF 0 MIDI files, and performs channel splitting automatically. When an SMF 0 format is detected, *Sequencer64* puts all of the events into the same sequence/pattern. While the file is being processed, a list of the channels present in the track is maintained.

Tempo and Time Signature events are read, if present. When saving a *Sequencer64* MIDI file, the Tempo and Time Signature events are saved as MIDI events. This allows other sequencers to read a *Sequencer64* MIDI file. This addition of Tempo can fix imported tracks that don't have a measure value.

Once the end-of-track is encountered for that pattern, one new empty pattern is created for each channel found in the original sequence. The events in the main patten are scanned, one by one, and added at the end of the appropriate patten. If the event is a channel event, then the event is inserted into the patten that was created for that channel. If the event is a non-channel event, then each patten gets a copy of that event.

After processing, the MIDI buss information, track name, and other pieces of information are attached to each sequence. The following figure shows in imported SMF 0 tune, split into tracks.

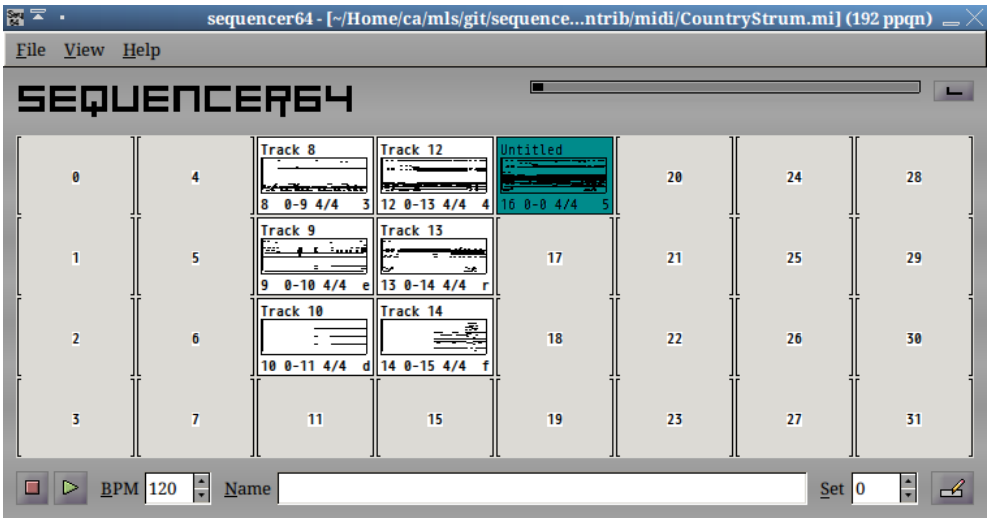



Figure 102: Imported SMF 0 MIDI Song

The imported SMF 0 track is preserved, in pattern slot #16. It is highlighted in a dark cyan color to remind the user that it is a special *Sequencer64* pattern. If the original track had no title, this track is named "Untitled". One will either delete this track before saving the file, or keep it muted.

Each single-channel track is given a title of either the form "N: Track-name" or, if the song was untitled, "Track N". The sequence number of each new track is the internal channel number (always the actual MIDI channel number minus one). The time-signature of each track is set to defaults, unless a time-signature event is encountered in the imported file.

*Sequencer64* supports reading some other information a MIDI SMF 0 track might have, such as the Tempo and the Time Signature. It saves this information in the first track of the MIDI file. The original SMF 0 track is also shown in the song editor, as in the following figure.

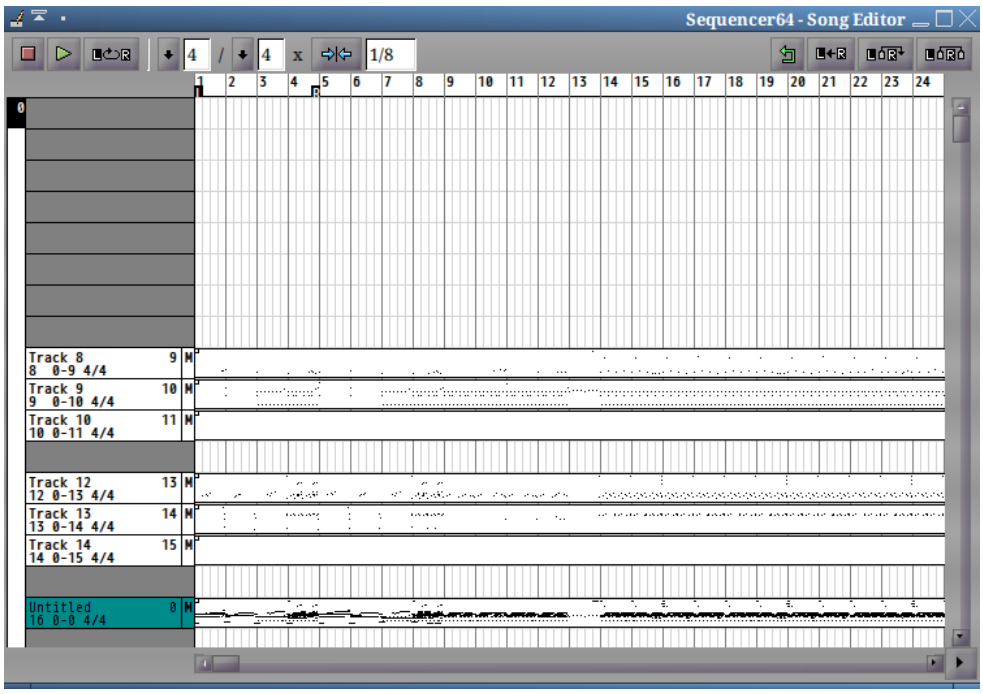



Figure 103: SMF 0 MIDI Song in the Song Editor

One is free to edit the imported tune to heart's content. Here, we added one instance of each track, including the SMF 0 track, to show what the imported song looks like.

## 18.2 Legacy Proprietary Track Format

The authors of *Seq24* took trouble to ensure that the format of the MIDI files it writes are compatible with other MIDI applications. *Seq24* also stores its own information (triggers, MIDI control information, etc) in the file, but marked so that other sequencers can read the file and ignore its *Seq24*-specific information.

*Sequencer64* continues that MIDI-compliant behavior, but has improved the compliance a bit. We call that last chunk of sequencer-specific information the "proprietary track". Before we discuss that last, proprietary track, note that the normal MIDI tracks that precede it include the SeqSpec ("sequencer-specific") control tags.

All of the SeqSpecs are shown in the next table. The `c_triggers` tag is obsolete, but still present.

The `c_musickey`, `c_musicscale`, and `c_backsequence` control tags are new with *Sequencer64*. They are saved as additional information in each sequence in which they have been specified in the sequence editor. For backward compatibility (and because it is probably the more common use case), these items can also be saved globally for the whole MIDI song, as an option.

These tags are preceded by the standard MIDI "FF 7F length" meta-event sequence. The following discussion applies to the final "proprietary" track as saved in the legacy *Seq24* format.

After all the counted MIDI event tracks are read, *Seq24* checks for extra data after them. If there is extra data, *Seq24* reads a long value. The first one encountered is a MIDI "sequencer-specific" (*SeqSpec*) section. It starts with

```
0x24240010
```

Table 7: All SeqSpec Items

| | |
|---|---|
| c_midibus | 24 24 00 01 00 00 00 00 |
| c_midich | 24 24 00 02 00 00 00 00 |
| c_midiclocks | 24 24 00 03 00 00 00 00 |
| c_triggers | 24 24 00 04 00 00 00 00 |
| c_notes | 24 24 00 05 00 00 00 00 |
| c_timesig | 24 24 00 06 00 00 00 00 |
| c_bpmtag | 24 24 00 07 00 00 00 00 |
| c_triggers_new | 24 24 00 08 00 00 00 00 |
| c_mutegroups | 24 24 00 09 00 00 00 00 |
| c_gap_A | 24 24 00 0A 00 00 00 00 |
| c_gap_B | 24 24 00 0B 00 00 00 00 |
| c_gap_C | 24 24 00 0C 00 00 00 00 |
| c_gap_D | 24 24 00 0D 00 00 00 00 |
| c_gap_E | 24 24 00 0E 00 00 00 00 |
| c_gap_F | 24 24 00 0F 00 00 00 00 |
| c_midictrl | 24 24 00 10 00 |
| c_musickey | 24 24 00 11 00 |
| c_musicscale | 24 24 00 12 00 |
| c_backsequence | 24 24 00 13 00 00 00 00 |
| c_transpose | 24 24 00 14 00 00 00 00 |
| c_perf_bp_mess | 24 24 00 15 00 00 00 00 |
| c_perf_bw | 24 24 00 16 00 00 00 00 |
| c_tempo_map | 24 24 00 17 00 00 00 00 |
| c_reserver_1 | 24 24 00 18 00 00 00 00 |
| c_reserver_2 | 24 24 00 19 00 00 00 00 |
| c_tempo_track | 24 24 00 1A 00 00 00 00 |
| c_seq_color | 24 24 00 1B 00 00 00 00 |
| c_seq_edit_mode | 24 24 00 1C 00 00 00 00 |

which is a Seq24 "c_midictrl" proprietary value flagged by the number "0x2424". MIDI requires this marker to be supported. Some applications, like *timidity*, handle it. Others complain about an unexpected header marker. Next, MIDI wants to see this triad of bytes

```
status = FF, type= 7F (proprietary), length = whatever
```

to precede proprietary data.

*Sequencer64* writes this information properly, starting with the `0x242400nn xFF 0x7F` marker. We also need to be able to read legacy Seq24 MIDI files, so that ability has been preserved in *Sequencer64*.

At this point, we have the **c_midictrl** information now. Next, we read a long value, seqs. It is 0.

```
24 24 00 10 00 00 00 00
```

Read the next long value, 0x24240003. This is **c_midiclocks**. We get a value of 0 for "TrackLength" (now a local variable called "busscount"):

```
24 24 00 03 00 00 00 00
```

If the buss-count was greater than 0, then for each value, we would read a byte value represent the bus a clock was on, and setting the clock value of the master MIDI buss. Another check for more data is made.

```
24 24 00 05 00 20 00 00
```

0x24240005 is **c_notes**. The value screen_sets is read (two bytes) and here is 0x20 = 32. For each screen-set:

```
len = read\_short()
```

If non-zero, each of the `len` bytes is appended as a string. Here, len is 0 for all 32 screensets, so the screen-set notepad is set to an empty string. Another check for more data is made.

```
24 24 00 07 00 00 00 78
```

0x24240007 is **c_bpmtag**. The long value is read and sets the perform object's bpm value. Here, it is 120 bpm. Another check for more data is made.

```
24 24 00 09 00 00 04 00
```

0x24240009 is **c_mutegroups**. The long value obtained here is 1024. If this value is not equal to the constant **c_gmute_tracks** (1024), a warning is emitted to the console, but processing continues anyway, 32 x 32 long values are read to select the given group-mute, and then set each of its 32 group-mute-states.

In our sample file, 32 groups are specified, but all 32 group-mute-state values for each are 0.

So, to summarize the legacy proprietary track's data, ignoring the data itself, which is mostly 0 values, as shown in table 8 "SeqSpec Items in Legacy Proprietary Track" on page 175

Table 8: SeqSpec Items in Legacy Proprietary Track

| | | |
|---|---|---|
| c_midictrl | 24 24 00 10 00 00 00 00 | |
| c_midiclocks | 24 24 00 03 00 00 00 00 | (buss count = 0) |
| c_notes | 24 24 00 05 00 20 00 00 | (screen sets = 32) |
| c_bpmtag | 24 24 00 07 00 00 00 78 | (bpm = 120) |
| c_mutegroups | 24 24 00 09 00 00 04 00 | (mg = 1024) |

The new format (again, ignoring the data) takes up a few more bytes. It starts with the normal track marker and size data, followed by a made-up track name ("Sequencer64-S"), as shown in table 9 "SeqSpec Items in New Proprietary Track" on page 176.

Table 9: SeqSpec Items in New Proprietary Track

| | |
|---|---|
| "MTrk" etc. | 4d 54 72 6b 00 00 11 0d 00 ... |
| Track name | 53 65 71 75 65 6e 63 65 72 32 34 2d 53 |
| c_midictrl | ff 7f 04 24 24 00 10 00 (???) |
| c_midiclocks | ff 7f 04 24 24 00 03 00 (buss count = 0) |
| c_notes | ff 7f 46 24 24 00 05 00 20 00... (screen sets = 32) |
| c_bpmtag | ff 7f 08 24 24 00 07 00 00 00 78 (bpm = 120) |
| c_mutegroups | ff 7f a1 08 24 24 00 09 00 00 04 00... (mg = 1024) |

For the new format, the components of the final proprietary track size are as shown here:

1. **Delta time**. 1 byte, always 0x00.
2. **Sequence number**. 5 bytes. OPTIONAL.
3. **Track name**. 3 + 10 or 3 + 15
4. **Series of proprietary specs**:
   - **Prop header**:
     - If legacy format, 4 bytes.
     - Otherwise, 2 bytes + varinum_size(length) + 4 bytes.
     - Length of the prop data.
5. **Track End**. 3 bytes.

Note that we still need to dig into all the new values that have accumulated over the last couple years!

## 18.3 MIDI Information

This section provides some useful, basic information about MIDI data.

### 18.3.1 MIDI Variable-Length Value

A variable-length value (VLV) is a quantity that uses additional bytes and continuation bits to encode large numbers without confusing a MIDI interpreter. See https://en.wikipedia.org/wiki/Variable-length_quantity.

The length of a variable length value obviously depends on the value it represents. Here is a simple list of the numbers that can be represented by a VLV:

```
1 byte:  0x00 to 0x7F
2 bytes: 0x80 to 0x3FFF
3 bytes: 0x4000 to 0x001FFFFF
4 bytes: 0x200000 to 0x0FFFFFFF
```

### 18.3.2 MIDI Track Chunk

```
Track chunk == MTrk + length + track_event [+ track_event ...]
```

- *MTrk* is 4 bytes representing the literal string "MTrk". This marks the beginning of a track.
- *length* is 4 bytes the number of bytes in the track chunk following this number. That is, the marker and length are not counted in the length value.
- *track_event* denotes a sequenced track event; usually there are many track events in a track. However, some of the events may simply be informational, and not modify the audio output.

A track event consists of a delta-time since the last event, and one of three types of events.

```
track_event = v_time + midi_event | meta_event | sysex_event
```

- *v_time* is the variable length value for elapsed time (delta time) from the previous event to this event.
- *midi_event* is any MIDI channel message such as note-on or note-off.
- *meta_event* is an SMF meta event.
- *sysex_event* is an SMF system exclusive event.

### 18.3.3 MIDI Meta Events

Meta events are non-MIDI data of various sorts consisting of a fixed prefix, type indicator, a length field, and actual event data.

```
meta_event = 0xFF + meta_type + v_length + event_data_bytes
```

- *meta_type* is 1 byte, expressing one of the meta event types shown in the table that follows this list.
- *v_length* is length of meta event data, a variable length value.
- *event_data_bytes* is the actual event data.

*Timidity* reads the legacy and new formats and plays the tune. *Sequencer64* saves the "b4uacuse" tune out, in both formats, with a "MIDI divisions" value of 192, versus its original value of 120. The song plays a little bit faster after this conversion.

The *midicvt* application does not read the legacy *Seq24* file format. It expects to see the MTrk marker. Even if the *midicvt* `--ignore` option is provided, *midicvt* does not like the legacy *Seq24* format, and ends with an error message. However, as shown by table 11 "Application Support for MIDI Files" on page 178, most applications are more forgiving, and can read (or ignore) the legacy format. The *gsequencer* application has some major issues in our installation, but it is probably our setup. (No JACK running?)

## 18.4 More MIDI Information

This section goes into even more detail about the MIDI format, especially as it applies to the processing done by *Sequencer64*. The following sub-sections describe how *Sequencer64* parses a MIDI file.

Table 10: MIDI Meta Event Types

| Type | Event |
|---|---|
| 0x00 | Sequence number |
| 0x01 | Text event |
| 0x02 | Copyright notice |
| 0x03 | Sequence or track name |
| 0x04 | Instrument name |
| 0x05 | Lyric text |
| 0x06 | Marker text |
| 0x07 | Cue point |
| 0x20 | MIDI channel prefix assignment |
| 0x2F | End of track |
| 0x51 | Tempo setting |
| 0x54 | SMPTE offset |
| 0x58 | Time Signature |
| 0x59 | Key Signature |
| 0x7F | Sequencer-Specific event |

Table 11: Application Support for MIDI Files

| **Application** | **Legacy** | **New** | **Original File** |
|---|---|---|---|
| ardour | TBD | TBD | TBD |
| composite | TBD | TBD | TBD |
| gsequencer | No | No | No |
| lmms | Yes | Yes | Yes |
| midi2ly | Yes | Yes | TBD |
| midicvt | No | Yes | Yes |
| midish | TBD | TBD | TBD |
| muse | TBD | TBD | TBD |
| playmidi | TBD | TBD | TBD |
| pmidi | TBD | TBD | TBD |
| qtractor | Yes | Yes | Yes |
| rosegarden | Yes | Yes | Yes |
| superlooper | TBD | TBD | TBD |
| timidity | Yes | Yes | Yes |

### 18.4.1 MIDI File Header, MThd

The first thing in a MIDI file is The data of the header:

```
Header ID:     "MThd"         4 bytes
MThd length:     6            4 bytes
Format:        0, 1, 2        2 bytes
No. of track:  1 or more      2 bytes
PPQN:           192           2 bytes
```

The header ID and it's length are always the same values. The formats that Sequencer64 supports are 0 or 1. SMF 0 has only one track, while SMF 1 can support an arbitary number of tracks. The last value in the header is the PPQN value, which specifies the "pulses per quarter note", which is the basic time-resolution of events in the MIDI file. Common values are 96 or 192, but higher values are also common. Sequencer64 and its precursor, Seq24, default to 192.

### 18.4.2 MIDI Track, MTrk

The next part of the MIDI file consists of the tracks specified in the file. In SMF 1 format, each track is assumed to cover a different MIDI channel, but always the same MIDI buss. (The MIDI buss is not a data item in standard MIDI files, but it is a special data item in the sequencer-specific section of *Seq24/Sequencer64* MIDI files.) Each track is tagged by a standard chunk marker, "MTrk". Other markers are possible, and are to be ignored, if nothing else. Here are the values read at the beginning of a track:

```
Track ID:      "MTrk"         4 bytes
Track length:  varies         4 bytes
```

The track length is the number of bytes that need to be read in order to get all of the data in the track.

**Delta time**. The amount time that passes from one event to the next is the *delta time*. For some events, the time doesn't matter, and is set to 0. This values is a *variable length value*, also known as a "VLV" or a "varinum". It provides a way of encoding arbitrarily large values, a byte at a time.

```
Delta time:    varies         1 or more bytes
```

The running-time accumulator is incremented by the delta-time. The current time is adjusted as per the PPQN ratio, if needed, and passed along.

### 18.4.3 Channel Events

**Status**. The byte after the delta time is examined by masking it against 0x80 to check the high bit. If not set, it is a "running status", it is replaced with the "last status", which is 0 at first.

```
Status byte:   varies         1 byte
```

If the high bit is set, it is a status byte. What does the status mean? To find out, the channel part of the status is masked out using the 0xF0 mask. If it is a 2-data-byte event (note on, note off, aftertouch, control-change, or pitch-wheel), then the two data bytes are read:

```
Data byte 0:   varies         1 byte
Data byte 1:   varies         1 byte
```

If the status is a Note On event, with velocity = data[1] = 0, then it is converted to a Note Off event, a fix for the output quirks of some MIDI devices. If it is a 1-data-btye event (Program Change or Channel Pressure), then only data byte 0 is read. The one or two data bytes are added to the event, the event is added to the current sequence, and the MIDI channel of the sequence is set.

### 18.4.4 Meta Events Revisited

If the event status masks off to 0xF0 (0xF0 to 0xFF), then it is a Meta event. If the Meta event byte is 0xFF, it is called a "Sequencer-specific", or "SeqSpec" event. For this kind of event, then a type byte and the length of the event are read.

```
Meta type:     varies         1 byte
Meta length:   varies         1 or more bytes
```

If the type of the SeqSpec (0xFF) meta event is 0x7F, parsing checks to see if it is one of the Seq24 "proprietary" events. These events are tagged with various values that mask off to 0x24240000. The parser reads the tag:

```
Prop tag:      0x242400nn      4 bytes
```

These tags provide a way to save and recover Seq24/Sequencer64 properties from the MIDI file: MIDI buss, MIDI channel, time signature, sequence triggers, and (new), the key, scale, and background sequence to use with the track/sequence. Any leftover data for the tagged event is let go. Unknown tags ate skipped.

If the type of the SeqSpec (0xFF) meta event is 0x2F, then it is the End-of-Track marker. The current time marks the length (in MIDI pulses) of the sequence. Parsing is done for that track.

If the type of the SeqSpec (0xFF) meta event is 0x03, then it is the sequence name. The "length" number of bytes are read, and loaded as the sequence name.

If the type of the SeqSpec (0xFF) meta event is 0x00, then it is the sequence number, which is read:

```
Seq number:    varies         2 bytes
```

Note that the sequence number might be modified latter to account for the current *Seq24* screenset in force for a file import operation.

Anything other SeqSpec type is simply skipped by reading the "length" number of bytes.

The remaining sections simply describe MIDI meta events in more detail, for reference.

## 18.5 Meta Events

Here, we summarize the MIDI meta events.

1. `FF 00 02 ssss`: Sequence Number.
2. `FF 01 len text`: Text Event.

3. `FF 02 len text`: Copyright Notice.
4. `FF 03 len text`: Sequence/Track Name.
5. `FF 04 len text`: Instrument Name.
6. `FF 05 len text`: Lyric.
7. `FF 06 len text`: Marker.
8. `FF 07 len text`: Cue Point.
9. `FF 08 through 0F len text`: Other kinds of text events.
10. `FF 2F 00`: End of Track.
11. `FF 51 03 tttttt`: Set Tempo, us/qn.
12. `FF 54 05 hr mn se fr ff`: SMPTE Offset.
13. `FF 58 04 nn dd cc bb`: Time Signature.
14. `FF 59 02 sf mi`: Key Signature.
15. `FF 7F len data`: Sequencer-Specific.
16. `FF F0 len data F7`: System-Exclusive

The next sections describe the events that *Sequencer* tries to handle. These are:

- Sequence Number (0x00)
- Track Name (0x03)
- End-of-Track (0x2F)
- Set Tempo (0x51) (Sequencer64 only)
- Time Signature (0x58) (Sequencer64 only)
- Sequencer-Specific (0x7F) (Handled differently in Sequencer64)
- System Exclusive (0xF0) Sort of handled, functionality incomplete.

### 18.5.1 Sequence Number (0x00)

```
FF 00 02 ss ss
```

This optional event must occur at the beginning of a track, before any non-zero delta-times, and before any transmittable MIDI events. It specifies the number of a sequence.

### 18.5.2 Track/Sequence Name (0x03)

```
FF 03 len text
```

If in a format 0 track, or the first track in a format 1 file, the name of the sequence. Otherwise, the name of the track.

### 18.5.3 End of Track (0x2F)

```
FF 2F 00
```

This event is not optional. It is included so that an exact ending point may be specified for the track, so that it has an exact length, which is necessary for tracks which are looped or concatenated.

### 18.5.4 Set Tempo Event (0x51)

The MIDI Set Tempo meta event sets the tempo of a MIDI sequence in terms of the microseconds per quarter note. This is a meta message, so this event is never sent over MIDI ports to a MIDI device. After the delta time, this event consists of six bytes of data:

```
FF 51 03 tt tt tt
```

Example:

```
FF 51 03 07 A1 20
```

1. 0xFF is the status byte that indicates this is a Meta event.
2. 0x51 the meta event type that signifies this is a Set Tempo event.
3. 0x03 is the length of the event, always 3 bytes.
4. The remaining three bytes carry the number of microseconds per quarter note. For example, the three bytes above form the hexadecimal value 0x07A120 (500000 decimal), which means that there are 500,000 microseconds per quarter note.

Since there are 60,000,000 microseconds per minute, the event above translates to: set the tempo to 60,000,000 / 500,000 = 120 quarter notes per minute (120 beats per minute).

This event normally appears in the first track. If not, the default tempo is 120 beats per minute. This event is important if the MIDI time division is specified in "pulses per quarter note", which does not itself define the length of the quarter note. The length of the quarter note is then determined by the Set Tempo meta event.

Representing tempos as time per beat instead of beat per time allows absolutely exact DWORD-term synchronization with a time-based sync protocol such as SMPTE time code or MIDI time code. This amount of accuracy in the tempo resolution allows a four-minute piece at 120 beats per minute to be accurate within 500 usec at the end of the piece.

### 18.5.5 Time Signature Event (0x58)

After the delta time, this event consists of seven bytes of data:

```
FF 58 04 nn dd cc bb
```

The time signature is expressed as four numbers. `nn` and `dd` represent the numerator and denominator of the time signature as it would be notated. The denominator is a negative power of two: 2 represents a quarter-note, 3 represents an eighth-note, etc. The `cc` parameter expresses the number of MIDI clocks in a metronome click. The `bb` parameter expresses the number of notated 32nd-notes in a MIDI quarter-note (24 MIDI Clocks).

Example:

```
FF 58 04 04 02 18 08
```

1. 0xFF is the status byte that indicates this is a Meta event.
2. 0x58 the meta event type that signifies this is a Time Signature event.
3. 0x04 is the length of the event, always 4 bytes.
4. 0x04 is the numerator of the time signature, and ranges from 0x00 to 0xFF.
5. 0x02 is the log base 2 of the denominator, and is the power to which 2 must be raised to get the denominator. Here, the denominator is 2 to 0x02, or 4, so the time signature is 4/4.
6. 0x18 is the metronome pulse in terms of the number of MIDI clock ticks per click. Assuming 24 MIDI clocks per quarter note, the value here (0x18 = 24) indidicates that the metronome will tick every 24/24 quarter note. If the value of the sixth byte were 0x30 = 48, the metronome clicks every two quarter notes, i.e. every half-note.
7. 0x08 defines the number of 32nd notes per beat. This byte is usually 8 as there is usually one quarter note per beat, and one quarter note contains eight 32nd notes.

If a time signature event is not present in a MIDI sequence, a 4/4 signature is assumed.

In *Sequencer64*, the `c_timesig` SeqSpec event is given priority. The conventional time signature is used only if the `c_timesig` SeqSpec is not present in the file.

### 18.5.6 SysEx Event (0xF0)

If the meta event status value is 0xF0, it is called a "System-exclusive", or "SysEx" event.

*Sequencer64* has some code in place to store these messages, but the data is currently not actually stored or used. Although there is some infrastructure to support storing the SysEx event within a sequence, the SysEx information is simply skipped. *Sequencer64* warns if the terminating 0xF7 SysEx terminator is not found at the expected length. Also, some malformed SysEx events have been encountered, and those are detected and skipped as well.

### 18.5.7 Sequencer Specific (0x7F)

This data, also known as SeqSpec data, provides a way to encode information that a specific sequencer application needs, while marking it so that other sequences can safely ignore the information.

```
FF 7F len data
```

In *Seq24 and Sequencer64*, the data portion starts with four bytes that indicate the kind of data for a particular SeqSpec event:

```
c_midibus        ^  0x24240001  Track buss number
c_midich         ^  0x24240002  Track channel number
c_midiclocks     *  0x24240003  Track clocking
c_triggers       ^  0x24240004  See c_triggers_new
c_notes          *  0x24240005  Song data, notes
c_timesig        ^  0x24240006  Track time signature
c_bpmtag         *  0x24240007  Song beats/minute
c_triggers_new   ^  0x24240008  Track trigger data
c_mutegroups     *  0x24240009  Song mute group data
c_midictrl       *  0x24240010  Song MIDI control
```

```
c_musickey       +  0x24240011  Track key (Sequencer64 only)
c_musicscale     +  0x24240012  Track scale (Sequencer64 only)
c_backsequence   +  0x24240013  Track background sequence (Sequencer64 only)

* = global only; ^ = track only; + = both
```

In *Seq24*, these events are placed at the end of the song, but are not marked as SeqSpec data. Most MIDI applications handle this situation fine, but some (e.g. midicvt) do not. Therefore, *Sequencer64* makes sure to wrap each data item in the 0xFF 0x7F wrapper.

Also, the last three items above (key, scale, and background sequence) can also be stored (by *Sequencer64*) with a particular sequence/track, as well as at the end of the song. Not sure if this bit of extra flexibility is useful, but it is there.

### 18.5.8 Non-Specific End of Sequence

Any other statuses are deemed unsupportable in *Sequencer64*, and abort parsing with an error.

If the –bus option is in force, it overrides the buss number (if any) stored with the sequence. This option is useful for testing a setup. Note that it also applies to new sequences.

At the end, *Sequencer64* adds the sequence to the encoded tune.

## 19 Sequencer64 JACK Support

This section describes some details concerning the JACK support of *Sequencer64*. As with *Seq24*, *Sequencer64* has JACK transport support. But, if one wants to use the older version of *Sequencer64* (versions 0.9.x) with JACK MIDI, one needs to expose the ALSA ports to JACK using `a2jmidid --export-hw` and connect the resultant MIDI JACK ports oneself, using *QJackCtl*, for example.

To enable the JACK transport support at run-time, the options `-j`/`--jack-transport`, `-J`/`--jack-master`, and `-C`/`--jack-master-cond` are available.

With version 0.90, *Sequencer64* can be built to support the legacy ALSA interface, the PortMIDI interface, or, best of all, the native JACK MIDI interface, loosely based on the RtMIDI project (see [18]). This mode also supports fallback-to-ALSA if the JACK server is not running.

- *Sequencer64*. . This application is built when the `--enable-alsamidi` option is specified at "configure" time. It is basically the 0.9.x version of *Sequencer64* application. However, this build is no longer the default.
- *seq64*. . This application is built when the `--enable-rtmidi` option is specified at "configure" time. This build is now the default build.
- *seq64portmidi*. . This application is built when the `--enable-portmidi` option is specified at "configure" time. This build is *deprecated*. It works with Linux, but, for Windows support, we will instead add Windows API calls to the "rtmidi" build of the project. We won't discuss this version at all. We've tested it for playback, but nothing else.

The following sections discuss the JACK transport support and the native JACK MIDI support.

## 19.1 Sequencer64 JACK Transport

This section is just underway. Here are some of the topics to be discussed:

1. What JACK functions are supported for JACK Transport.
2. Exposing the ALSA MIDI ports to JACK, when using the legacy ALSA version of *Sequencer64*.
3. Fixes to JACK Master mode.
4. Interactions with the Klick and Hydrogen applications.
5. Patches from the new (!) version of Seq24, 0.9.3, to correct for MIDI Clock drift over long durations.

In the meantime, the text files in the project's `contrib/notes` directory provide some useful setup notes.

JACK transport support is *separate* from native JACK MIDI support. The JACK transport client is an invisible client with the name "seq64-transport", while the JACK MIDI client is visible in *QJackCtl*, and the ports created are part of the "seq64" client.

The first thing to note about JACK transport with *Sequencer64* is that the progress bars will not move unless *Sequencer64* is connected to a JACK client, such as *Hydrogen* (in JACK MIDI mode) or *Yoshimi*. Currently, *Sequencer64* will connect to a JACK client automatically only at startup, where it will connect to all JACK clients that it finds. If it can't find a JACK client, then it will fail to register a JACK port, and cannot play.

The second thing is that *Sequencer64* still has the issue where it must be JACK Master to follow transport. More on this issue later.

## 19.2 Sequencer64 Native JACK MIDI

This section discusses the new *seq64* application, which supports native JACK MIDI. It is now the *official* version of *Sequencer64*, and new bugs will be fixed mainly in this version.

The first thing to note about *seq64* is that it supports both ALSA and JACK MIDI. If one runs it to support JACK, and JACK is not present, then *seq64* falls back to ALSA support.

To run seq64 to support JACK, for now, one must add the `-t` or `--jack-midi` option. Why `-t`? We are running out of option letters. And eventually we will make the `-t` option the default. If *Sequencer64* (in its native JACK mode, using the `-t` or `--jack-midi` option) is run in JACK mode *without JACK running* on the system, it will take awhile to come up (in ALSA mode). If run from the console, one will see:

```
$ ./Seq64rtmidi/seq64 -t
[Activating native JACK MIDI]
[Reading rc configuration /home/ahlstrom/.config/sequencer64/sequencer64.rc]
[Reading user configuration /home/ahlstrom/.config/sequencer64/sequencer64.usr]
[Activating native JACK MIDI]
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
     . . .
[JACK server not running?]
connect: JACK server not running?
```

```
rtmidi_info: no compiled support for specified API     (???)
[Initialized, running without JACK sync]
9 rtmidi ports created:
Input ports (3):
  [0] 0:1 system:announce (system)
  [1] 14:0 Midi Through:Midi Through Port-0
  [2] 24:0 nanoKEY2:nanoKEY2 MIDI 1
Output ports (6):
  [0] 14:0 Midi Through:Midi Through Port-0
  [1] 24:0 nanoKEY2:nanoKEY2 MIDI 1
  [2] 128:0 TiMidity:TiMidity port 0
  [3] 128:1 TiMidity:TiMidity port 1
  [4] 128:2 TiMidity:TiMidity port 2
  [5] 128:3 TiMidity:TiMidity port 3
```

To enable native JACK MIDI, use the `-t/--jack-midi` options. When enabled, the "transport" button in the main window will show the word "JACK" (no matter whether JACK transport is also enabled or not). Please note that, if you select the JACK-MIDI option and JACK is not available, *seq64* will start in ALSA mode and *it will save* that mode when exiting.

The JACK (`-t`) and ALSA (`-A`) options are sticky options. That is, they are saved to the "rc" configuration file at exit, so one does not have to specify them in subsequent *seq64* sessions.

### 19.2.1 Sequencer64 JACK MIDI Output

By default (or depending on the "rc" configuration file), the new `seq64` version of *Sequencer64* will automatically connect the ports that it finds to `seq64`, as shown in the following figure:

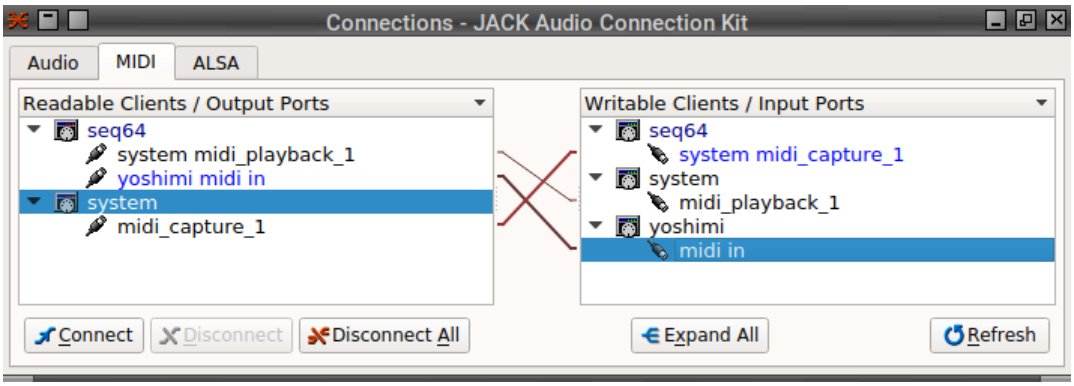



Figure 104: JACK MIDI Ports and Auto-Connect

The `seq64:system midi_playback_1` output port shown in the left panel is created by `seq64`. It connects it to the `system:midi_playback_1` port in the right panel, which is actually the input for the *Korg nanoKEY2* controller. ALSA detects the real name of this USB MIDI device, but JACK does not. Thus, the MIDI tab shows the "system" name of the USB MIDI port, while the ALSA tab does show the name.

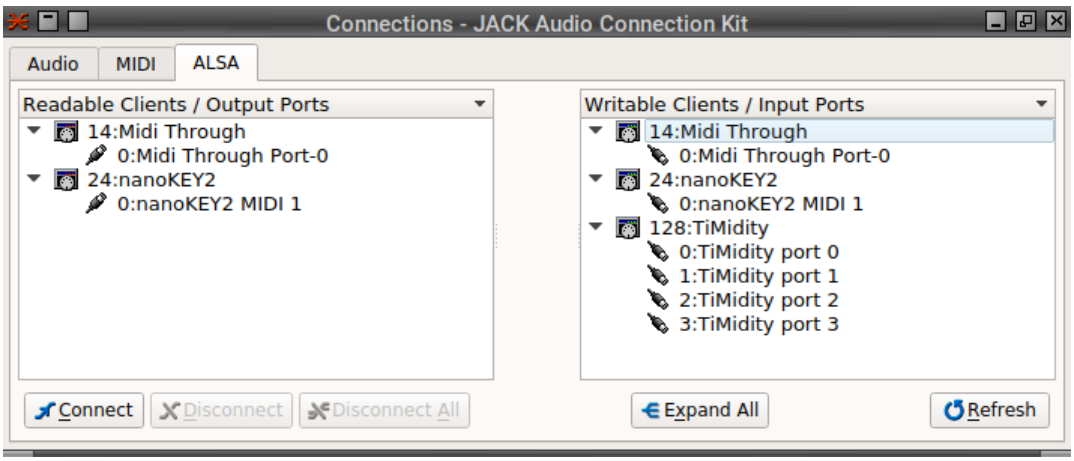



Figure 105: ALSA MIDI Ports

Note that the input connection is not useful unless `seq64` could send setup information to the *nanoKEY2*. Korg provides a configuration application for *Windows*. For *Linux*, a Python application called *Nano-Basket* ([16]) is available.

More useful is the automatic connection between `seq64:yoshimi midi in` in the left (output) panel and `yoshimi:midi in` in the right (input) panel. With it it, *Sequencer64* patterns with the proper output-buss setting can play to the *Yoshimi* software synthesizer.

The output ports available are shown in *seq64*'s **File / Options / MIDI Clock** tab, shown here:

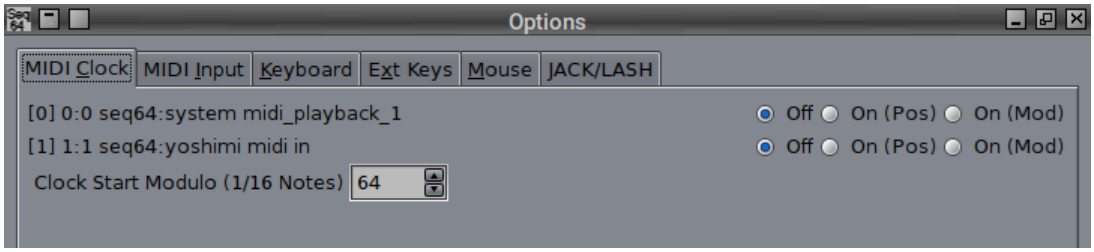



Figure 106: JACK MIDI Ports in Seq64

Note that the index, client, and buss numbers are all the same. There's actually a bug here, since all `seq64` ports should have the same client number. However, in JACK, clients and ports are referred to by name, not number, and so functionality is not affected.

Entry 0 (`seq64:system midi_playback_1`) is, as already noted, not useful unless the *nanoKEY2* can accept input control. Entry 1 allows `seq64` to send MIDI to *Yoshimi*. A pattern must specify output buss 1 in order for the MIDI to reach *Yoshimi*. Another option, normally for testing only, is to specify the "bus" option on the command line:

```
$ seq64 --jack-midi --bus 1
```

With that option, all patterns send to buss 1.

### 19.2.2 Sequencer64 JACK MIDI Input

One more connection to note is the input connection to `seq64`. Referring back to Figure 104 "JACK MIDI Ports and Auto-Connect" on page 186we see that `system:midi_capture_1` in the left (output) panel is connected to `seq64:system midi_capture_1` in the right (input) panel. This allows the *nanoKEY2* MIDI output port to feed `seq64`, which can then record the input notes, and also forward them to *Yoshimi* so that they can be heard.

This input port is also shown in the **File / Options / MIDI Input** tab, shown here:

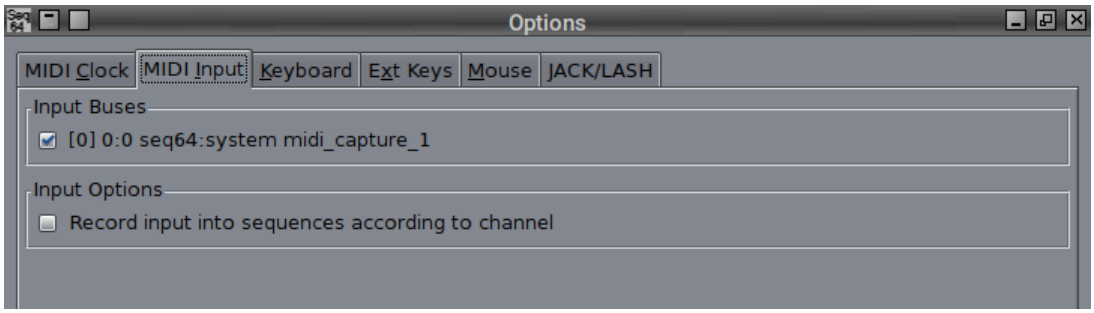



Figure 107: JACK MIDI Input Ports

When the check-box for that buss is selected, the input can be captured by `seq64`.

### 19.2.3 Sequencer64 JACK MIDI Virtual Ports

The manual-versus-normal port support for JACK MIDI is essentially the same as that for ALSA. Currently, the same option name is used (we will provide a more generic option-name soon). The `-m`/`--manual-alsa-ports` option actually provides what are known as "virtual" ports. These are ports that do not represent hardware, but are created by applications to allow them to connect to other applications or MIDI devices.

The difference between manual/virtual ports and normal ports is that, while normal ports are automatically connected to the remote ports that exist in the system, the manual/virtual ports are just created, and one must manually connect them via, for example, the *QJackCtl* connections dialog.

So, if one wants *seq64* to automatically connect to all existing JACK MIDI ports, *do not* use the `-m`/`--manual-alsa-ports` option... use the `-a`/`--auto-alsa-ports` option. Both options apply to both ALSA and JACK, but we do not want to change the option-names at this time.

If one wants the freedom to make the connections oneself, or with a session manager, then use the manual/virtual option. Here are the ports created in manual/virtual mode:

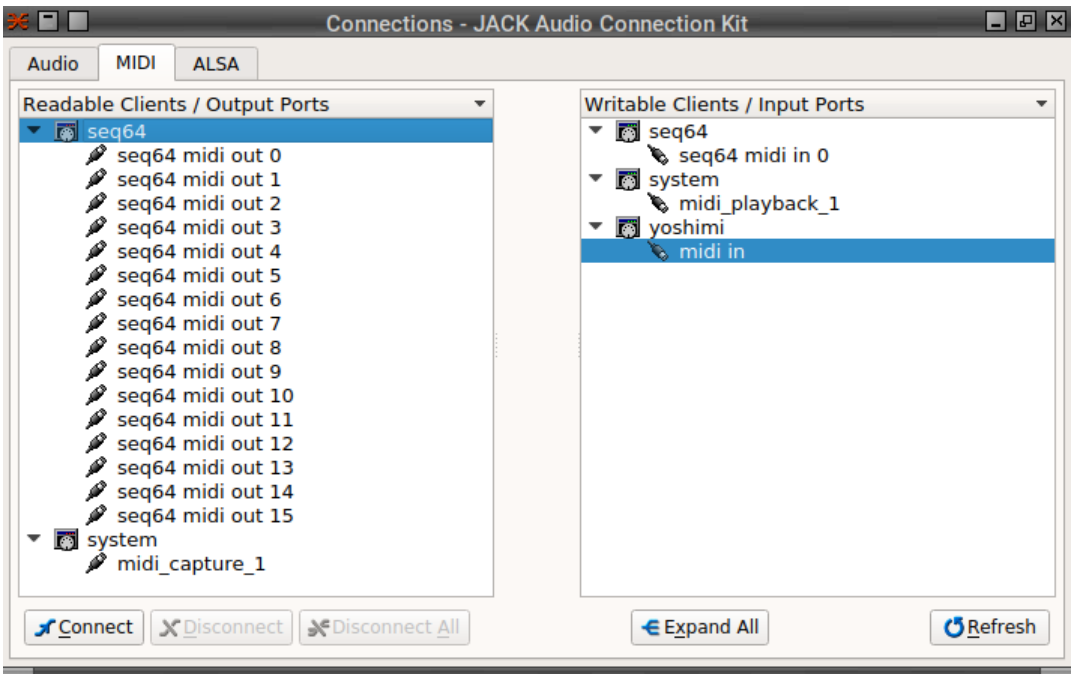



Figure 108: JACK MIDI Manual Ports

One sees that `seq64` creates 16 output ports (busses), and one input port (buss). One also sees that `seq64` *does not* connect the ports automatically. The user or the session manager will have to make those connections.

The **MIDI Clock** and **MIDI Input** tabs reflect in an obvious manner what is seen in *QJackCtl*, so we won't bother to show those tabs.

### 19.2.4 Sequencer64 JACK MIDI and a2jmidid

One more thing to show is that `seq64` can deal with the odd naming of JACK ports created by the *a2jmidid* application.

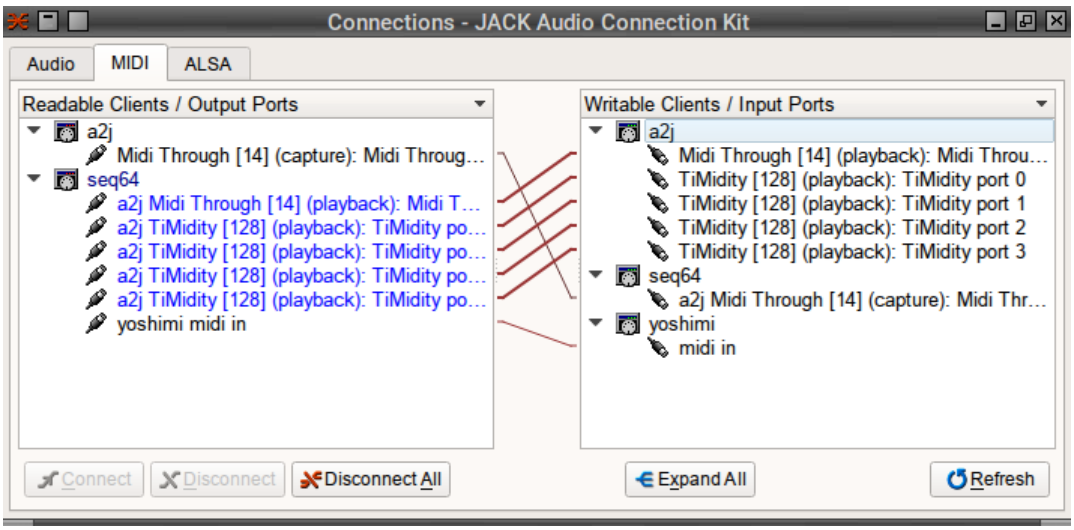



Figure 109: JACK MIDI a2jmidid Ports

One can see in the right (input) panel that that the `a2j` client creates 5 entries, one for "Midi Through", and four for the *TiMidity* client. In the left (output) panel, one sees (in blue) the output ports that `seq64` creates to connect to the ports created by *a2jmidid*.

Also note the true JACK output port, `seq64:yoshimi midi in` to connect to the input port `yoshimi:midi in`.

Again, if these automatic connections get in the way, run `seq64` in manual/virtual mode.

When recording, do not forget the step option. If one paints notes with the mouse, the note is previewed, and the note position advances with each click. If one paints notes via an external MIDI keyboard, the notes are painted and advanced, but they are not previewed. To preview them, click the "pass MIDI in to output" button in the pattern editor window to activate so that they will be passed to your sound generator. Be careful of MIDI loops!

# 20 Kudos

This section gives some credit where credit is due. We have contributors to acknowledge, and have not caught up with all the people who have helped this project:

- *Tim Deagan (tdeagan)*: fixes to the mute-group support.
- *0rel*: an important fix to add and relink notes after a paste action in the pattern editor.
- *arnaud-jacquemin*: a bug report and fix for a regression in mute-groups support.
- *Stan Preston (stazed)*: ideas for some upcoming improvements based on his *seq32* project. A lot of ideas. And a lot of code!
- *Animtim*: a number of bug reports and a new logo.
- *jean-emmanuel*: scrollable main-window support, other features and reports.
- *Olivier Humbert (trebmuh)*: French translation for the desktop files.
- *Oli Kester*: The creator of *Kepler34*, from which we are getting clues on porting the user-interface to Qt 5 and Windows.

Also some bug-reporters and testers:

- *F0rth*: a request for scripting support, a possible future feature.
- *gimmeapill*: testing, bug-reports, and, um, "marketing".
- *georgkrause*: a number of helpful bug reports.
- *goguetchapuisb*: found that Seq64 native JACK did not properly handle the copious Active Sensing messages emitted by Yamaha keyboards.
- *milkmiruku*: mainwids issues and ideas.
- *muranyia*: feature request for numbered piano keys and bug-reports.
- *pixelrust*: reports of issues with "fruity" interaction.
- *simonvanderveldt*: issues with window sizing and more.
- *ssj71*: a request for an LV2 plugin version, a possible future feature.
- *triss*: a request for OSC support, a possible future feature.
- *layk*: some bug reports, and, we are pretty sure, some nice videos that demonstrate *Sequencer64* on *YouTube*. See [10].
- *matt-bel*: reported a regression from *Seq24*, which could use a MIDI control event to mute/unmute multiple patterns at once, a cool feature!

... and there are more to add to this list....

There are a number of authors of *Seq24*. ideas from other *Seq24* fans), and some deep history, as one can see in Figure 21 "Help Credits" on page 34, and in Figure 22 "Help Documentation" on page 34. All of these authors, and more, have contributed to *Sequencer64*, whether they know it or not. The original author is Rob C. Buse; where the word "I" occurs, that is probably him. Without his work, we would never have started *Sequencer64*.

From the original author:

> *Seq24* is a real-time MIDI sequencer. It was created to provide a very simple interface for editing and playing MIDI 'loops'. After searching for a software based sequencer that would provide the functionality needed for a live performance, there was little found in the software realm. I set out to create a very minimal sequencer that excludes the bloated features of the large software sequencers, and includes a small subset of features that I have found usable in performing.
>
> Written by Rob C. Buse. I wrote this program to fill a hole. I figure it would be a waste if I was the only one using it. So, I released it under the GPL.

This project deserves to stay alive! (And it is alive! A new version, 0.9.3, has come out from the LaunchPad group! It corrects a problem with MIDI Clock drift). Taking advantage of Rob's generosity, we've created a reboot, a refactoring, an improvement (we hope) of *Seq24*. It preserves (we hope) the lean nature of *Seq24*, while adding a few features we've found useful, to make it the "vi of sequencers".

Always remember that, without *Seq24* and its authors, *Sequencer64* would never have come into being.

# 21 Summary

In summary, we can say that you will find *Sequencer64* intriguing.

Contact: If you have ideas about *Sequencer64* or a bug report, please email us (at mailto:ahlstromcj@gmail.com). If it's a bug report, please add **[BUG]** to the Subject, or use the GitHub bug-reporting interface.

## 22 References

The *Sequencer64* reference list.

## References


[1] ALSA team. *Advanced Linux Sound Architecture (ALSA) project homepage.* http://www.alsa-project.org/. ALSA tools through version 1.0.29. 2015.

[2] amSynth team, Nick Dowell. *amSynth and Demos with Calf Effects.* http://amsynth.com/amsynth.html. Includes links to demos and the source code. 2015.

[3] Bristol team, Nick Copeland. *Bristol: A Vintage Synthesizer Emulator.* http://www.linuxsynths.com/BristolPatchesDemos/bristol.html. 2014.

[4] Coolsoft. *Coolsoft MIDIMapper (Windows).* https://coolsoft.altervista.org/en/midimapper. 2018.

[5] Coolsoft. *Coolsoft VirtualMIDISynth (Windows).* https://coolsoft.altervista.org/en/virtualmidisynth. 2018.

[6] FluidSynth team. *FluidSynth: A SoundFont Synthesizer.* http://www.fluidsynth.org/. 2014.

[7] Jay Capela Music. *"Combine": A Seq24 Demonstration.* https://www.youtube.com/watch?v=fUiXbVT0bJQ. 2010.

[8] JACK team. *JACK Audio Connection Kit.* http://jackaudio.org/. 2015.

[9] Oli Kester. *Kepler34: Seq24 for the 2010s.* https://github.com/oli-kester/kepler34. 2010-2016.

[10] Lassi Ylikojola. *Many demo videos of Sequencer64* https://www.youtube.com/watch?v=YStYVjFv1TM, https://www.youtube.com/watch?v=GBlEP8Ffqss, https://www.youtube.com/watch?v=4gG8SvJxJkA&t=28s, https://www.youtube.com/watch?v=n4Z4WPK6FpA. 2010-2017.

[11] LinuxSynths team, briandc@linuxsynths.com. *A Sonic Palette on the Linux Platform.* http://www.linuxsynths.com/. 2015.

[12] Dave Phillips. *At the Sounding Edge: Introducing seq24.* http://www.linuxjournal.com/article/8304. Linux Journal, May 12, 2005.

[13] Chris Ahlstrom. *Extension of midicomp/midi2text to convert between MIDI and ASCII text format.* https://github.com/ahlstromcj/midicvt. 2015-2016.

[14] linuxaudio.org. *seq24: toggle sequences with a MIDI controller.* http://wiki.linuxaudio.org/wiki/seq24togglemiditutorial. 2013.

[15] midi.org. *Summary of MIDI Messages.* https://www.midi.org/specifications/item/table-1-summary-of-midi-message#2. Year unknown.

[16] Roy Vegard. *Configurator software for the Korg nanoSERIES of MIDI controllers.* https://github.com/royvegard/Nano-Basket. 2015.

[17] PortMedia team. *Platform Independent Library for MIDI I/O.* http://portmedia.sourceforge.net/portmidi/. 2010.

[18] Gary P. Scavone. *The RtMIDI Tutorial.* https://www.music.mcgill.ca/~gary/rtmidi/. 2016.

[19] Seq24 Team. *The home site for the Sequencer64 looping sequencer.* http://www.filter24.org/seq24/download.html. 2010.

[20] pneumanlsd. *Linux audio demo: Live sequencing with seq24.* https://www.youtube.com/watch?v=f8zLV0vlSpY. 2010.

[21] synthWF. *Misty Corridor - Seq24 (with QSynth).* https://www.youtube.com/watch?v=RH99zHvffGQ. 2012.

[22] pneumanlsd. *Linux music tutorial: seq24, part 1.* https://www.youtube.com/watch?v=J2WDHS1wYeM. 2010.

[23] pneumanlsd. *Linux music tutorial: seq24, part 2.* https://www.youtube.com/watch?v=i3Vpi3oxdqk. 2010.

[24] Seq24 Team. *The home site for the Sequencer64 looping sequencer.* https://launchpad.net/seq24. 2016.

[25] Excds. *A simple mapping for toggling the LEDs on the Novation launchpad together with seq24.* https://github.com/Excds/seq24-launchpad-mapper. 2013.

[26] Stan Preston (stazed). *The home site for the Seq32 looping sequencer.* https://github.com/Stazed/seq32. 2016.

[27] sbrauer. *The home site for the original Seq42 looping sequencer.* https://github.com/sbrauer/seq42. 2016.

[28] Stan Preston (stazed). *A fork of the Seq42 looping sequencer.* https://github.com/Stazed/seq42. 2016.

[29] Chris Ahlstrom. *A reboot of the Seq24 project as "Sequencer64".* https://github.com/ahlstromcj/sequencer64/. 2015-2017.

[30] Chris Ahlstrom. *The Sequencer64 User Manual.* https://github.com/ahlstromcj/sequencer64-doc/. 2015-2017.

[31] Chris Ahlstrom. *Sequencer64 Debian Packages.* https://github.com/ahlstromcj/sequencer64-packages/. 2017.

[32] Kevin at subatomicglue.com. *Subatomic Mods for Seq24 Win32.* http://www.subatomicglue.com/seq24/. 2010.

[33] Timidity++ Team. *Download site for Timidity++ source code.* http://sourceforge.net/projects/timidity/. 2015.

[34] VMPK Team. *Virtual MIDI Piano Keyboard.* http://vmpk.sourceforge.net/. 2015.

[35] The Wootangent man. *Sequencer64 Tutorial Video, Part 1.* http://wootangent.net/2010/10/linux-music-tutorial-seq24-part-1/. 2010.

[36] The Wootangent man. *Sequencer64 Tutorial Video, Part 2.* http://wootangent.net/2010/10/linux-music-tutorial-seq24-part-2/. 2010.

[37] Yoshimi team abrolag@users.sourceforge.net. *The download site for the Yoshimi software synthesizer.* http://yoshimi.sourceforge.net/. 2015.

[38] Yoshimi team. *The alternate location for the Yoshimi source-code.* https://github.com/abrolag/yoshimi/. 2015.

[39] Chris Ahlstrom. *A Yoshimi User Manual.* https://github.com/ahlstromcj/yoshimi-doc/. 2015-2017.

[40] Chris Ahlstrom. *A Yoshimi Cookbook.* https://github.com/ahlstromcj/yoshimi-cookbook/. 2015.

[41] Mark McCurry, Paul Nasca (ZynAddSubFX team). *The download site for the ZynAddSubFX software synthesizer.* http://zynaddsubfx.sourceforge.net/. 2015.

# Index