Rtl66 Developer Guide 0.1.0

Chris Ahlstrom
(ahlstromcj@gmail.com)

April 27, 2024

Rtl66 Logo

Contents

List of Figures

List of Tables

1 Introduction

This document describes "Rtl66", a MIDI library based on the *RtMidi* ([6])library, but with many changes and extensions. The following project supports *Rtl66* (and documentation):

- `https://github.com/ahlstromcj/rtl66.git`
- https://ahlstromcj.github.io/

Feel free to clone or fork it!

1.1 Rtl66: What!?

The *Rtl66* library is meant to support a potential *Seq66v2* application. It is based on the *RtMidi* ([6])library, but with the following changes and additions:

- New naming conventions. Not a big fan of camel cases, so class and function names have been converted to use lower case and underscores.
- Modularization. Rather than a monolithic C++ file, `RtMidi.cpp/h` has been broken into a couple dozen separate files so that it is easier to zero in on a particular class or API.
- Additional support functions. For example, a simple command-line parser has been added to support the test applications in a uniform way.
- Pipewire. In progress, this API will be added and will be supported in *Linux*.
- Library code based on *Seq66*. This includes simplied versions of MIDI busses, calculations, port information, JACK transport, auto-connect,and more. This code can be adapted more easily for use by other developers.

The original (but refactored) *RtMidi* code resides in the `include/rtl` and `src/rtl` directories. The "C" interface has been preserved to a large extent, but the "C++" uses namespaces and modified names.

The Seq66-based additions to the library are in the `include/midi` and `src/midi` directories. In the future, there will be a *Seq66* library that relies on the *Rtl66* library and adds concepts like performer, triggers, playlists, etc.

There is also a `transport` directory, currently containing only a JACK transport class. Perhaps we can add *Ableton Link* ([3]) transport later.

At some point we might add rudimentary audio support, for playing short clips.

1.2 History

Well into the development of *Sequencer64* ([8]), we decided to add support for JACK MIDI. After a quick look around, *RtMidi* seemed to be the best basis for JACK MIDI. However, after getting well into development, it was found that the *RtMidi* model and the *Sequencer64* models did not align well.

The result was a butchery of the *RtMidi* library, much added complexity to the application libraries, and a difficult-to-extend application. Furthermore, the Mac/Windows support was dropped, and instead the existing *PortMidi* ([5]) support was used for those operating systems.

With a big laundry list of potential upgrades, and a desire to simplify *Seq66*, it was decided to adapt *Seq66* code to *RtMidi*.

2 Summary

The rest of this document will be a walk-through of the library and directory structure.

3 Walk-through

Let's start by walking through one of the test applications, `tests/qmidiin`. We run our debugger, set a breakpoint where the input object is created, and run using ALSA.

```
$ cgdb build/tests/qmidiin
(gdb) r --alsa
```

Here is what happens:

1. **Select ALSA as the API**.
   - Use the command-line to set the `-alsa` option.
   - `rt_simple_cli()` sets the API to `rtl::rtmidi::api::alsa`.
   - Retrieved the API with `rtl::rtmidi::desired_api()`.
2. **Create the MIDI input proxy**. Pass the API to the `rtl::rtmidi_in` constructor. The three possible parameters are:

- The API. For *Linux*, they are *JACK*, *ALSA*, and, later, *Pipewire*. The default is "unspecified", which would invoke the fallback sequence that detects which APIs are available (currently JACK to ALSA).
- The client-name. Empty by default. For testing, the `-client name`.
- The input-queue size. Defaults to 100 MIDI messages.

3. **Open the selected API**. `rtl::rtmidi_in::open_midi_api()` detects the compiled APIs and tries to create the desired one.
4. **Create the actual MIDI input**. `rtl::midi_alsa_in()` with default client-name "rtl66 in".
5. **Set ALSA data structure to defaults**. Initializes `rtl::midi_alsa_data` to useless defaults.
6. **Set up the ALSA client**. The call sequence is:
   - `rtl::midi_alsa_in::initialize()`
   - `rtl::midi_alsa::impl_initialize()`

   The ALSA sequencer handle (`snd_seq_t`) gets set up. Also called the "client" handle, not to be confused with "port" handles.
7. **Set the tempo and PPQN**. `rtl::midi_alsa::impl_set_tempo()`. Sets the default tempo and PPQN.
8. **Get the number of ports**. The call sequence is:
   - `rtmidi_in::get_port_count()` calls
   - Get the `midi_api` pointer returned by `rtl::rtmidi::rt_api_ptr()`.
   - Calll `midi_alsa_in::get_port_count()` via this pointer.
   - `rtl::midi_alsa::impl_get_port_count()` for MIDI input.
   - This function also gets the port capabilities via ALSA.
   - Call the static function `rtl::get_port_info()` (in `midi_alsa.cpp`) with a port value of `-1`, which indicates "all ports".
9. **Open the input port**. The call sequence is:
   - `rtl::rtmidi_in::open_port()`.
   - `rtl::midi_alsa_in::open_port()`.
   - `rtl::midi_alsa::impl_open_port()`.
   - `rtl::get_port_info()`, this time for port 0.
10. **Get port information**. `rtl::get_port_info()` iterates through all input ports, retrieving port information (wasteful!) until it gets the proper kind of port where the index of the port matches the desired port number. .
11. **Create the port**. `rtl::rtmidi_in::open_port()` then gets the sender and receiver addresses, sets the client, the port, the capabilities, type, and number of channels in the port (16), and the port's name. (EMPTY!!!!!!) Optionally sets time-stamping information.
12. **Port subscription**. `midi_alsa::impl_subscription()` is called to set up ALSA port subscription, a fancy way to connect ports.

If ALSA were not specified, then `rtl::rtmidi::fallback_api()` would be called to try JACK first, then ALSA. (Later, PipeWire will be the first choice).

Also note that of the `midi_api` functions call an "impl" function as a helper, to reduce code duplication.

3.1 Supported APIs

Rtl66 supports the following MIDI APIs:

- PipeWire. (Planned, not yet implemented).
- JACK.
- ALSA.
- Windows Multi-Media.
- Mac OSX Core.
- Web MIDI.
- Dummy.

3.2 xxxx

4 References

The *Rtl66* references 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] JACK team. *JACK Audio Connection Kit.* http://jackaudio.org/. 2015.

[3] Ableton Live Link. *Ableton Link (transport).* https://www.ableton.com/en/link/, https://github.com/Ableton/link/. 2022.

[4] JACK Audio. *New Session Manager, an offshot of Non Session Manager.* https://github.com/jackaudio/new-session-manager 2021.

[5] PortMedia team. *Platform Independent Library for MIDI I/O.* http://portmedia.sourceforge.net/. 2010.

[6] Gary P. Scavone. *The RtMIDI Tutorial.* https://www.music.mcgill.ca/~gary/rtmidi/. 2016.

[7] Seq24 Team. *The home site for the Seq24 looping sequencer.* http://www.filter24.org/seq24/download.html. 2010.

[8] Chris Ahlstrom. *A reboot of the Seq24 project as "Sequencer64".* https://github.com/ahlstromcj/sequencer64/. 2015-2021.

[9] Chris Ahlstrom. *A reboot of the Seq24 project as "Seq66".* https://github.com/ahlstromcj/seq66/. 2015-2022.

Index