A Yoshimi 1.3.9 User Manual

Chris Ahlstrom (ahlstromcj@gmail.com) with Will J. Godfrey

May 22, 2016

Contents

List of Figures

List of Tables

1 Introduction

This document was inspired by finding a fairly thorough wiki version of a *ZynAddSubFX* manual (see reference [29]). That wiki, and the Open Document Format version of it, showed a lot of screen shots and a detailed survey of the settings and parameters of *ZynAddSubFX*. It inspired us to be even more thorough in describing *Yoshimi*, and then Will got involved to provide even more details. Please note that this document also owes a lot to the descriptions and diagrams provided by the original *ZynAddSubFX* author, Paul Nasca, as well as some others whose names we do not have.

1.1 Yoshimi Versus ZynAddSubFX

This document describes how to use *Yoshimi* [17], the software synthesizer derived from the great *ZynAddSubFx* [21] software synthesizer. Because of their common origin, much of this document also applies to *ZynAddSubFx* and depends upon some *ZynAddSubFX* documentation and diagrams.

Yoshimi is an algorithmic MIDI software synthesizer for Linux. It synthesizes in real time, can run polyphonic or monophonic, with multiple simultaneous patches on one or more MIDI channels, and has broad microtonal capability. It includes extensive addititive, subtractive, and pad synth capabilities which can be run simultaneously within the same patch. It also has eight audio effects modules.

Originally based on the 2.4.0 version of *ZynAddSubFX* (Copyright 2002-2009 Nasca Octavian Paul), development of *Yoshimi* has continued for quite a while now in its own direction. These include major optimizations for audio and MIDI performance, and more recently progressive development of user-level and command-line access to all controls. At the same time, refinement continues, both visually and within the code.

What are the advantages of *Yoshimi* versus *ZynAddSubFX*? At one time *Yoshimi* had better JACK support than *ZynAddSubFX*, but that is no longer true. Both projects are now in active development, and both are progressing along their respective roadmaps. *Yoshimi* currently has a slight revamped graphical user interface, while *ZynAddSubFX* has a major user-interface upgrade planned. Each project has some features and capabilities that the other lacks. There may be internal differences of importance to a developer, as well as features that aren't so easy to discover. So, really, it is not clear that there

is really a big advantage to one over the other... use them both and make your own decision! Or just continue using both of them!

That being said, please note that the references to *ZynAddSubFx* in this manual apply primarly to versions of *ZynAddSubFx* prior to version 3.0 – the *ZynAddSubFx* project is planning a major user-interface change for version 3.0, and many of the file formats will change as well. Of course, one can still continue to use the old "Zyn" as well.

1.2 New Features

This section provides an *ad hoc*, catch-as-catch-can survey of the new features of *Yoshimi*, in no particular order.

1.2.1 LV2 Plugin

Yoshimi can now run as an LV2 plugin. Supported features:

1. Sample-accurate MIDI timing.
2. State save/restore support via *LV2_State_Interface*.
3. Working UI support via *LV2_External_UI_Widget*.
4. Programs interface support via *LV2_Programs_Interface*.
5. Multi-channel audio output. 'outl' and 'outr' have lv2 index 2 and 3. All individual ports numbers start at 4.

1.2.2 Control Automation

A planned feature, already well underway, is controls automation support. This is a part of a common controls interface. There are significant extensions to the NRPNs that *Yoshimi* handles.

Sensitivity to MIDI volume change (CC #7) is now variable in 'Controllers' in the same way as pan width, etc. The numeric range is 64 to 127; the default at 96 gives the same sensitivity as before at -12dB relative to the GUI controls. 127 gives 0dB and 64 gives -26dB.

In parallel with this there are more NRPNs supported so that one can perform some of these controls via automation. The arrangment looks positively steam-punk, but is actually very easy to use, requiring only a utility that can send MIDI CCs. NRPNs aren't special. They are simply a specific pattern of CCs. *Yoshimi*'s implementation is very forgiving, doesn't mind if one stops halfway through (will just get on with other things while it waits) and will report exactly what it is doing. See the new NRPN sections.

1.2.3 MIDI CC

To help when things don't seem to go right, one can now show 'raw' incoming CCs. This is enabled from the 'MIDI' tab in 'Settings'. These are the values before *Yoshimi* does any processing.

MIDI program changes have always been pretty clean from the time Cal first introduced them, but now GUI changes are just as clean. While it is generally best to change a program when the part is silent, even if a part is sounding there is usually barely a click. There is no interference at all with any other sounding parts.

Sometimes MIDI CCs don't seem to give the results one expects. There is now a setting that will report all incoming CCs so that one can discover what *Yoshimi* actually sees (which may not be what was expected).

At the request of one user, *Yoshimi* now has an implementation of CC2, Breath Control. This feature combines volume with filter cutoff.

1.2.4 Vectors

It's probably best to more clearly separate the concept of *parts* versus *channels* these days. *Yoshimi* now can provide up to 64 parts, in blocks of 16. One can now decide how many one wants to have available using the spin-box alongside the channel number. One can have 16, 32 or 64 parts. By default, all the upper parts are mapped to the same MIDI channel numbers as the lowest ones, but have independent voice and patch set values. They can not normally receive independent note or control messages. However, vector control will intelligently work with however many are set, as will all the NRPN direct part controls.

1.2.5 Bank Support

Bank root directories are better identified, with IDs that can be changed by the user in the GUI. This is also made available for selecting over MIDI. MIDI only sees banks in the *current* **root** directory, but all banks are accessible to the GUI.

It is now possible to set up a new bank root path when starting from the command line. This takes the form:

```
$ yoshimi -D /home/(username)/(directory)/(subdirectory)/bank
```

Yoshimi will then immediately scan this path for new banks, but won't make the root (or any of its banks) current. The final directory doesn't in fact have to be 'banks', but, traditionally, we have always done this. Also, when running from the command line there is now access to many of the system and root, bank, and other settings.

Yoshimi now splits out roots and banks from the main configuration file, and also creates a new history file. The separation means that the different functions can be implemented, saved, and loaded at the most appropriate time. These files have "yoshimi" as the doctype, as they are in no way relevant to *ZynAddSubFX*. See the new Banks sections.

1.2.6 Accessibility

One of the main features of recent releases of *Yoshimi* is improved accessibility. The effectiveness (and indeed usefulness) of this will help shape future complimentary interfaces. Also, a number of first-time defaults have been changed to make this easier.

It has always been possible to run *Yoshimi* headless, but now real control is available. In the first place, when starting from the command line, an argument can be included for a new root path to be defined to point to a set of banks fetched from elsewhere. This will be given the next free ID.

Once running, almost all dynamic setup (i.e. doesn't require a restart) can now be done within the terminal window. There is also extensive control of roots, banks, parts and instruments including the ability to list and set all of these. One can now do things like:

```
path add /home/music/yoshimi/banks
set part 4 program 130
```

Additional controls that are frequently taken forgranted in the GUI but otherwise get forgotten are master key shift and master volume. We have the most important parts of vector control exposed to the command line. For all of this there is extensive error checking and feedback, which can be rendered aurally using text-to-speech software.

There is one partially-sighted persion we very occasionally hear from. There is also a totally blind person (working with a Braille reader/writer) who has offered a lot of suggestions, and very much likes vector control. So accessibility is an important feature of *Yoshimi*.

1.2.7 Command Line

There is greater control of one's working environment. One can have just the GUI, just a CLI or both, and these settings can be saved. If one tries to disable both, one will get a polite warning and will be left with just the CLI.

The CLI can now access almost all top level controls as well as the 'main page' part ones and can select any effect and effect preset, but can't yet deal with the individual effects controls. It can be used to set up Vector Control much more quickly and easily than using NRPNs. The CLI is also context sensitive, which, along with careful choice of command names and abreviations, allows very fast access with minimal typing.

Yoshimi's parser is case-insensitive for commands (but not for filenames), and accepts the shortest unambiguous abbreviation. However it is quite pedantic, and expects spelling to be correct regardless of length. Apart from the 'back' commands, it is word-based so spaces are significant.

Some examples:

"`s p 4 pr 6`" (set part 4 program 6): This command sets part 4 to the instrument with ID 6 from the current bank. It also then leaves one at the part context level and pointed to part 4. Additionally, it will activate that part if it was off (and the configuration setting is checked). In most cases the words 'program' and 'instrument' are interchangable.

"`s ef 1 ty rev`": This command moves one up to part effects context level and sets that part's effect number 1 to effect type 'reverb'.

"`s pre 2`": This command sets preset number 2 (we use numbers here as most preset names repeat the effect type).

"`..s 6 v 80`": This command drops one back up to part level, switches one to part 6, and sets its volume to 80 (but doesn't actually enable it).

"`/s ve cc 93`": This command drops back up to the top level, and sets vector control for channel 0, X axis to respond to CC 93 leaving one in the vector context.

Whenever intermediate values are omitted, the default or last-used value will be assumed, and all counting starts from zero. The CLI prompt always shows what level one is on, and the help lists are also partly context-sensitive, so one doesn't see a lot of irrelevent clutter. There is more, and a lot more to come! While doing all this work, we've also ensured that *Yoshimi* instrument patches are still fully compatible with *ZynAddSubFX* patches, and have now ported across the new refinements with thanks.

1.2.8 Audio Support

The preferred JACK/ALSA MIDI and audio connections are no longer fixed at compile time. There are now checkboxes on 'Settings' to change them. One can also set 'preferred' startup ALSA/JACK/MIDI and audio devices. These selections will be remembered on the next run.

Yoshimi will now always start even if the audio/MIDI backend called for doesn't exist. In this situation it will try all combinations in the order JACK, ALSA, and null. This enables one to then change the settings and try again.

Another significant improvement is to the handling of ALSA audio, which is still very important for some people. Up until now, *Yoshimi* has insisted on 2-channel 16-bit format. Tests have shown that virtually all motherboard sound chipsets will handle this setting, but many external ones don't. So now we initially request 32 bit 2 channel and work towards a compromise with the hardware. With external sound modules in mind, endian swaps are also implemented.

Yoshimi is now verified as being able to use the 192000 Hz sample rate in both ALSA and JACK... if one has a suitable soundcard!

1.2.9 Miscellany

Yoshimi now stamps instrument and patch set XML files with its own major and minor version numbers so it is possible to tell which version created the files, or whether they were created by *ZynAddSubFX*.

It is now possible to direct messages to either `stderr` (the error output of a terminal console) or the report window on the fly. If one chooses `stderr`, the **Reports** button is greyed out.

One can now use the mouse scroll wheel to adjust rotary controls. Holding down the Ctrl key gives access to finer adjustment. Also, horizontal as well as vertical mouse movement will adjust the knob.

Part-editing windows carry the part number and voice name in the title bar. For the AddSynth oscillator window this also includes the voice number.

When opening an instrument bank, one can now tell exactly which synth engines are used by each instrument. This is represented by three pale background colours: Red = AddSynth; Blue = SubSynth; and Green = PadSynth.

If the instruments are kits they scanned to find out if any member of the kit contains each engine, so that the colors above can be applied. This feature is duplicated in the current part, the mixer panel for the currently loaded instruments, and in the Instrument Edit window. The same colors highlight the engine names when they are enabled with the check boxes.

Yoshimi now remembers where major windows were last placed (per instance), and if any were left open at shutdown, they will be reopened at the same location on the next run.

Thanks to the *ZynAddSubFX* developers, *Yoshimi* now has pink as well as white noise available on Addsynth voices. Pink noise sounds 'softer'. With the latest 'depop' port from *ZynAddSubFX*, *Yoshimi* is now fully compatibly with all instrument files.

The 'Humanise' feature has had more interest so it's been upgraded. It's now a slider and it's setting can be saved in patch sets. It provides a tiny per-note random detune to an *entire* part (all engines in all kits), but only for that part.

Audio and MIDI preferences have been improved. If one sets (say) ALSA MIDI and JACK audio, either from the GUI or the command line, the setting can be saved and will be reinstated on the next run. These settings are per-instance, so if one has multiple sound cards, one can make full use of them.

Barring major system failures, there are now no circumstances where Yoshimi will fail to start.

We have tested *Yoshimi* in 'recovery' mode, logged in as root, with no X server. Using the command `/usr/local/share/bin/yoshimi -A -i` worked perfectly and auto-connected the keyboard so we could prove everything worked. We still need the X11 libraries to compile *Yoshimi*, but we don't know if it is practical to provide a compile time option to build a purely headless version.

1.3 Document Structure

The structure of this document is a struggle. No matter which route is taken, there's no way to avoid jumping all over this document to adequately cover a topic. Therefore, the sections are basically provided in the order their contents appear in the user interface of *Yoshimi*. To help the reader jump around this document, multiple links and references are supplied.

Usage tips for each of the functions provided in *Yoshimi* are sprinkled throughout this document. Each tip occurs in a section beginning with "Tip:". Each tip is provided with an entry in the Index, under the main topic "tips".

Bug notes for some of the oddities found in *Yoshimi* are sprinkled throughout this document. Each bug occurs in a sentence beginning with "Bug:". Each bug is provided with an entry in the Index, under the main topic "bugs".

New features since the last version are flagged with "New:" We cannot pretend to have marked all new developments, as *Yoshimi* is advancing fast.

TODO items are also present, in the same vein. This document currently has a number of them!

1.4 Yoshimi Mailing List

The *Yoshimi* project used to have an email listserv at SourceForge, but the unreliability of the site has prompted a move to a new mailing list. See reference [19]. The team have managed to port across all the old yoshimi-user archives to this new site. See reference [20].

Subscribe to the *Yoshimi* mailing list with an e-mail to: `yoshimi-request@freelists.org` or by visiting `http://www.freelists.org/list/yoshimi`.

To post to the list, send an email to: `yoshimi@freelists.org`. The news archive is at: `https://www.freelists.org/archive/yoshimi`.

1.5 Yoshimi Licensing

Before we get started, one more thing.

> Software licenses are are something I *really* don't want to get involved in - I have much better things to do with my time - but I found I was obliged to do so.
>
> It is possible I'm the only person who knows all the following events, as I was the one that instigated them!
>
> The first time I saw ZynAddSubFX source files they were licensed as GPL V2. At that time Zyn had a number of very serious problems, and not much was being done about them. Somewhat naively I asked Lars Luthman if he would help, as he had offered a couple of small patches previously. His response was that he would not do any significant work, as he did not agree with the GPL V2 only license.
>
> I then contacted Paul, explaining the situation and asking if he would consider a change in the license to V2 or later. I was actually a bit surprised that he immediately agreed. When I next looked at the sources, the licenses on the files had indeed been updated, so I passed this information on.

Unfortunately Paul forgot to update the website, but I wasn't especially concerned as it was only the files themselves that really mattered.

While developing Yoshimi after the initial fork, Cal queried the license situation. I told him of the conversations I'd had, and passed him a copy of the email I'd got from Paul. Later on, Cal - in good faith - wrote new sources and placed them under GPL V3. This would be quite compatible with V2 or later, but not with V2 strict.

What I didn't notice until very much later was that Paul had only updated half of the text in the sources, leaving the actual licence in an ambiguous state.

To the best of my knowledge, V3 is not compatible with V2 strict, but V2 or later is. However the *complete* project then becomes downgraded to V2 strict - although the V2 or later sources (such as all the new root/bank code) can independently be freely merged into V3 code.

I doubt anyone would actually make an issue of this. However, to safeguard Yoshimi as a whole, I took it upon myself to change Cal's code to V2 or later. I believe it retains the spirit of his wishes, and the only person with standing to object – his daughter - has been totally supportive of the work currently being done on Yoshimi.

Any source code I add will be GPL V2 or later.

Update.

The original change discussion has now been located and the license for both Zyn and Yoshi is confirmed as GPL V2 or later.

Anyone wanting to confirm this should look at the Zyn user list archives August 2007 and September 2007.

1.6 Let's Get Started with Yoshimi!

Let us run *Yoshimi*, but run it without using *JACK*, which complicates the discussion of *Yoshimi*. The first thing to do is make sure one has no other sound application running (unless one wants to risk blocking *Yoshimi* or hearing two sounds simultaneously, depending on one's sound card and ALSA setup). Then start *Yoshimi* so that it uses ALSA for audio and ALSA for MIDI. See section 19 "Yoshimi Man Page" on page 198.

```
$ yoshimi -a -A
```

Figure 1: Yoshimi Splash Screen!

One sees a brief message, and then the splash screen. We show the splash screen, figure 1 "Yoshimi Splash Screen!" on page 16, here because it goes away too fast when one runs *Yoshimi*! What fun is that?

Next shown is the *Yoshimi* main window, as shown in figure 2 "Yoshimi Main Screen, 1.3.8 and Above" on page 17, and it persists, of course:

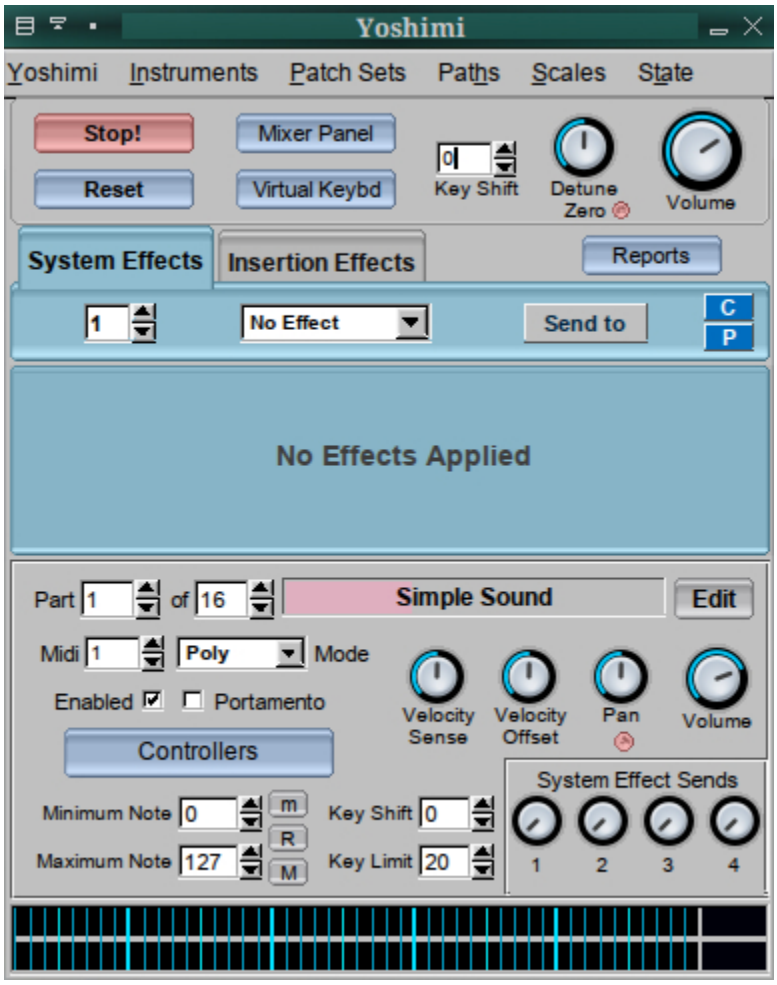

Figure 2: Yoshimi Main Screen, 1.3.8 and Above

For this manual, we describe the main window as being composed of the following sections:

1. **Menu**.
2. **Top Panel**.
3. **Effects Panel**.
4. **Bottom Panel**. Includes the VU-meter at the bottom.

There's a lot going on with *Yoshimi*, and there's no way to describe it in linear order. This manual will describe how to do useful things in each of the sections noted above, while leaving some of the details to be described in later sections, to which reference can be made for the details. This document depends heavily on index entries and references. There is also a "cookbook" at [2], but it is a long way from being comprehensive. Check it out anyway.

Finally, if one grabs the latest source code for *Yoshimi*, in the 'examples' directory there is a complete song set, 'OutThere.mid' and 'OutThere.xmz'. Together these produce a fairly complex 12 part tune that makes *Yoshimi* work quite hard.

2 Concepts

Before we start with the user-interface, let's cover some concepts and terms. *Yoshimi* requires the user to understand many concepts and terms. Understanding them makes it easier to configure *Yoshimi* and to drive it from a sequencer application.

Significant portions of this section are shamelessly copied (and tweaked) from Paul Nasca's original *ZynAddSubFX* manual [26] or [27]. One can discern such sections by the usage of the term *ZynAddSubFX* instead of *Yoshimi*. However, even the *Yoshimi* developers sometimes refer to *ZynAddSubFX* or *Zyn*.

Note that there are some audio/electrical concepts discussed in greater detail in section 6 "Stock Settings Elements" on page 69. Perhaps they belong in this "concepts" section, but they are directly tied to user-interface items.

2.1 Concepts / Terms

This section doesn't provide comprehensive coverage of terms. It covers mainly terms that might puzzle one at first, or have a special meaning in *Yoshimi*.

2.1.1 Concepts / Terms / Cent

The **cent** is a logarithmic unit of measure for musical intervals. Twelve-tone equal temperament divides the octave into 12 semitones of 100 cents each. Typically, cents are used to measure extremely small pitch intervals, or to compare the sizes of comparable intervals in different tuning systems. The interval of one cent is much too small to be heard between successive notes.

2.1.2 Concepts / Terms / Frame

The audio **frame** is a single sample of however many channels an application is handling. If one is using JACK, a mono signal will have frames of 1 float, 2 floats for stereo, etc. A six-channel device will have six samples in a single frame. An audio or JACK buffer will contain more than one frame of data. Buffers generally range in size from 16 to 1024 frames. Low values provide less latency, but make the CPU work harder.

2.1.3 Concepts / Terms / Instrument

In *Yoshimi*, an *instrument* is a complex sound that can be constructed using ADDsynth, SUBsynth, PADsynth, and kits. Each instrument is loaded into a *part* (see section 2.1.4 "Concepts / Terms / Part" on page 18).

In our documentation, we will sometimes use the terms "instrument", "patch", and even "program" interchangeably and loosely. However, "part" now has a different meaning, as seen in the next term.

2.1.4 Concepts / Terms / Part

In *Yoshimi*, a *part* is one of 16, 32, or 64 "slots" into which one can load an instrument (see section 2.1.3 "Concepts / Terms / Instrument" on page 18). Each part can be enabled or disabled, and assigned to a particular MIDI channel, one of the 16 MIDI channels. Note that the previous *Yoshimi* limit on parts was 16. Since 1.3.5, this limit has been raised to 64.

2.1.5 Concepts / Terms / Patch

In MIDI jargon, a *patch* is a sound played on one of 16 channels in a MIDI device. Many synthesizers can handle several waveforms per patch, mixing different instruments together to create synthetic sounds. Each waveform counts as a MIDI voice. Some sound cards can support two or more waveforms per patch. *Yoshimi* has some ability to combine waveforms ("voices") into one instrument (section 2.1.3 "Concepts / Terms / Instrument" on page 18), which can then be loaded into a *Yoshimi* part (section 2.1.4 "Concepts / Terms / Part" on page 18).

Before General MIDI, which standardized patches, MIDI vendors assigned patch numbers to their synthesizer products in an arbitrary manner.

2.1.6 Concepts / Terms / Patch Set

A patch set (also known as "patchset") is basically a group of instruments related simply by the user wanting to have them all loaded at once into *Yoshimi*. A patch set is stored in a `.xmz` file. A patch set is akin to a preset, in that it stores a combination of items, that took awhile to set up, for easy retrieval later.

As with most applications, *Yoshimi* and *ZynAddSubFX* allow for one to save one's work and reload it. *Yoshimi* has a number of different files that make up the current configuration. Together, they make up the concept of a *patch set* (also called a *patchset*). See section 3 "Configuration Files" on page 28.

2.1.7 Concepts / Terms / Presets

Presets allow one to save the settings for any of the components which support copy/paste operations. This is done with preset files (`.xpz`), which get stored in the folders indicated by **Paths / Preset Dirs...**. Note that the number of preset directories that can be set is limited to 128 (the same as for roots and banks).

In MIDI jargon, a *preset* is an instrument that can be easily loaded. It is also called a *program* or a *patch*. A program is selected via a "program-change" message. A *preset* is any collection of settings that can be saved to the clipboard or to a file, for later loading elsewhere.

2.1.8 Concepts / Terms / Program

In MIDI jargon, a *program* is the same as a *preset* (2.1.7).

2.1.9 Concepts / Terms / Voice

In MIDI jargon, a *voice* is the same as a *preset* or a *program*. In *Yoshimi*, a *voice* is a single configurable waveform that is just one of up to eight waveforms in an ADDsynth setting. Such voices can also be used as modulators for other voices.

2.2 Concepts / ALSA Versus JACK

Some discusson from the *Yoshimi* wiki. Here for eventual clarification.

A bit of a question mark was raised over ALSA MIDI support. A lot of people seem to be giving this up and relying on bridges like *a2jmidi* for legacy software and hardware inputs. JACK MIDI is already synchronous so should be jitter-free whereas ALSA MIDI runs on a 'best effort' basis. Added to which JACK is available for OS X and Windows so concentrating on this could make a possible port to other platforms more attractive – not to me I (Will J. Godfrey) hasten to add!

Seq24 (a nice, if old, sequencer) uses ALSA MIDI. To connect applications that exclusively support JACK MIDI, *a2jmidid* will do the translation. (Jack v. 1 has this integrated in recent versions, apparently).

ALSA is more complex as it handles the sound card's format, commonly 16-bit integers, 24 bit integers (low byte ignored), and short integers. Less commonly it may be floats or the weird 24-bit long integers. We're still not sure if these are packed or low-aligned (top byte ignored). We've assumed they are low-aligned, but we don't know anyone who has such a card, in order to prove it. The only ALSA format *Yoshimi* doesn't support is float.

Something that's not obvious is the way that ALSA audio is controlled and who takes command. If one sets a specific destination, then *Yoshimi* says what it wants. It's often a negotiation on bit depth and channel count, but *Yoshimi* nearly always gets to decide the buffer size (it will be set to the internal buffer size). However, if the destination is "default", then ALSA decides on the sound card, bit depth, number of channels and the buffer size, and *Yoshimi* will set it's internal buffer size to match. On most machines this always seems to be 1024.

Yoshimi is now verified as being able to use the 192000 Hz sample rate in both ALSA and JACK... if one has a suitable soundcard!

If *Yoshimi* is configured for JACK, but cannot find the JACK server, it will try for ALSA. If neither JACK nor ALSA works, it will run with a null client so one can at least try to work out what went wrong. This status is reported to the command line or Reports window.

For JACK, if one has started *Yoshimi* from the command-line with the `-K` option, it will auto-connect the main left and right outputs. On some machines, using the `-k` argument to also start JACK ends up running *jackdbus* which seems to route all JACK audio to `/dev/null`.

2.3 Concepts / Banks and Roots

In *Yoshimi*, a *root* is a location in which banks can be stored. It is basically a directory, though it ultimately is assigned a number by *Yoshimi*, to be able to access it in an automated way. By choosing a root, one can hone in on a smaller collection of banks.

Sometimes, one will see the term *path*. In *Yoshimi*, a *path* is simply the directory location of a root. This change is reflected in the user-interfaces, both graphical and command-line. Note that there are other file categories, such as presets, that are located via paths.

Another important concept in *Yoshimi* is *banks*. Instruments can be stored in banks. These are loaded and saved automatically by the program. On program start, the last used bank is loaded. A single bank can store up to 128 instruments normally, and 160 using extended programs. A bank isn't a file... it is a directory, managed by *Yoshimi*, which contains instrument (`.xiz`) files.

These concepts are discussed in great detail in section 4 "Banks and Roots" on page 32.

2.4 Concepts / Basic Synthesis

This section describes some of the basic principles of synthesis, and contains suggestions on how to make instruments that sound like they have been made with professional equipment. This applies to *Yoshimi*

or to any synthesizer (even if one wrote it oneself with a few lines of code). All the ideas from *Yoshimi* are derived from from the principles outlined below.

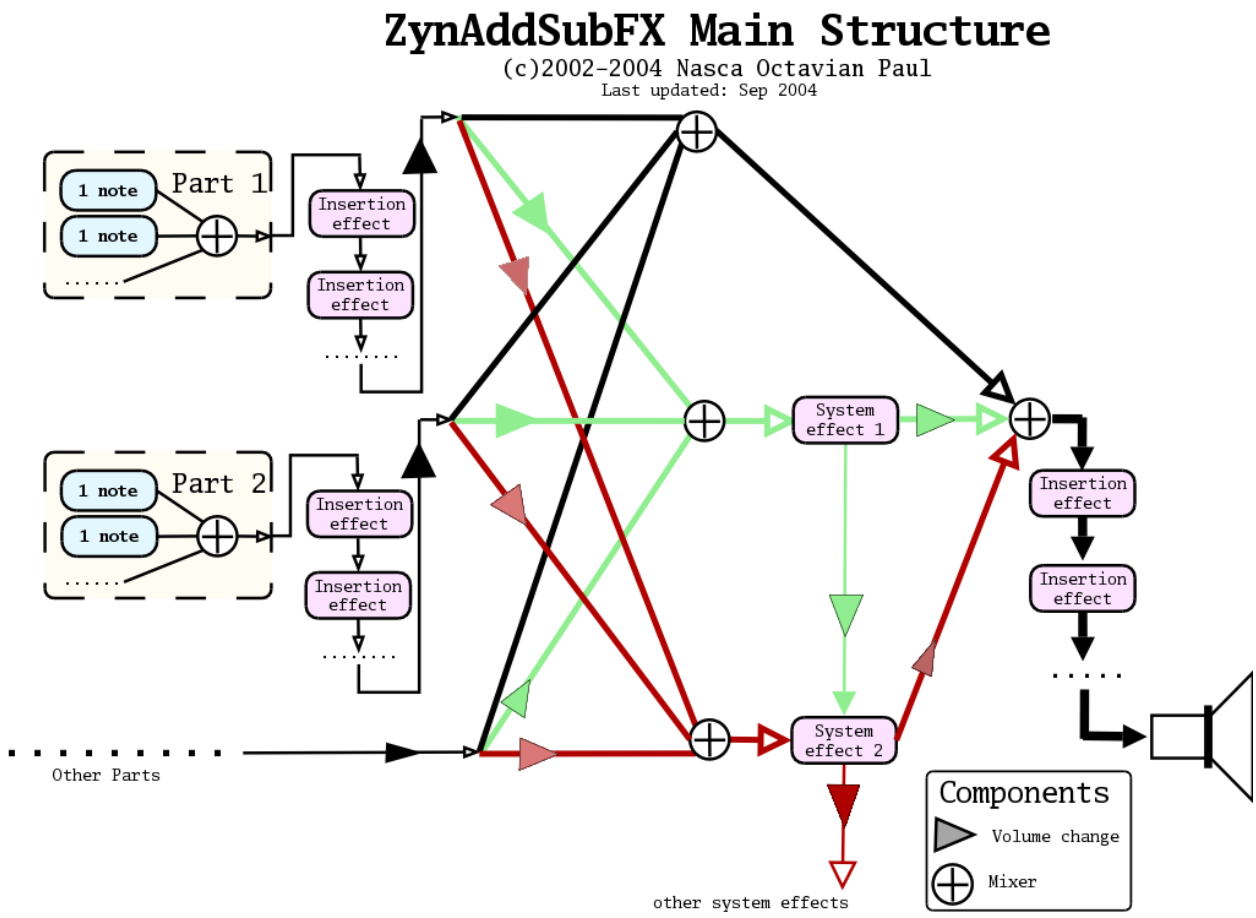

Figure 3: ZynAddSubFX/Yoshimi Main Structure

For a given part, the synthesizer first creates a note. Each note's waveform (for example, in a chord) is summed (mixed). This complex waveform is then send to the series of Insertion effects (if any) that are defined. Each part is then sent to a System effect and (depending on the wetness of the mix) directly to a mixer. Additional Insertion effects (if any) are then applied. The result is the final output of the synthesizer.

The synthesizer has three major types of parameters:

1. **Master settings/parameters**. Contains all parameters (including effects and instruments).
2. **Instrument parameters**. Contains ADDnote/SUBnote/PADnote parameters for a part.
3. **Scale settings**. Contains the settings of scales (*Yoshimi* is a micro-tonal synth) and few other parameters related to tunings.

2.4.1 Concepts / Basic Synthesis / Panning

Pan lets one apply panning, which means that the sound source can move to the right or left. Set it to 0.0 to only hear output on the right side, or to the maximum value to only hear output on the left side.

2.4.2 Concepts / Basic Synthesis / Wetness

Wetness determines the mix of the results of the effect and its input. This mix is made the effects output. If an effect is wet, it means that none of the input signal is bypassing the effect. If it is dry, then the effect is bypassed completely, and has no effect.

2.4.3 Concepts / Basic Synthesis / Single Note

The idea of this synthesis model is from another synthesizer Paul Nasca wrote years ago, released on the Internet as "Paul's Sound Designer". The new model is more advanced than that project (adding SUBsynth, more LFO's/Envelopes, etc.), but the idea is the same.

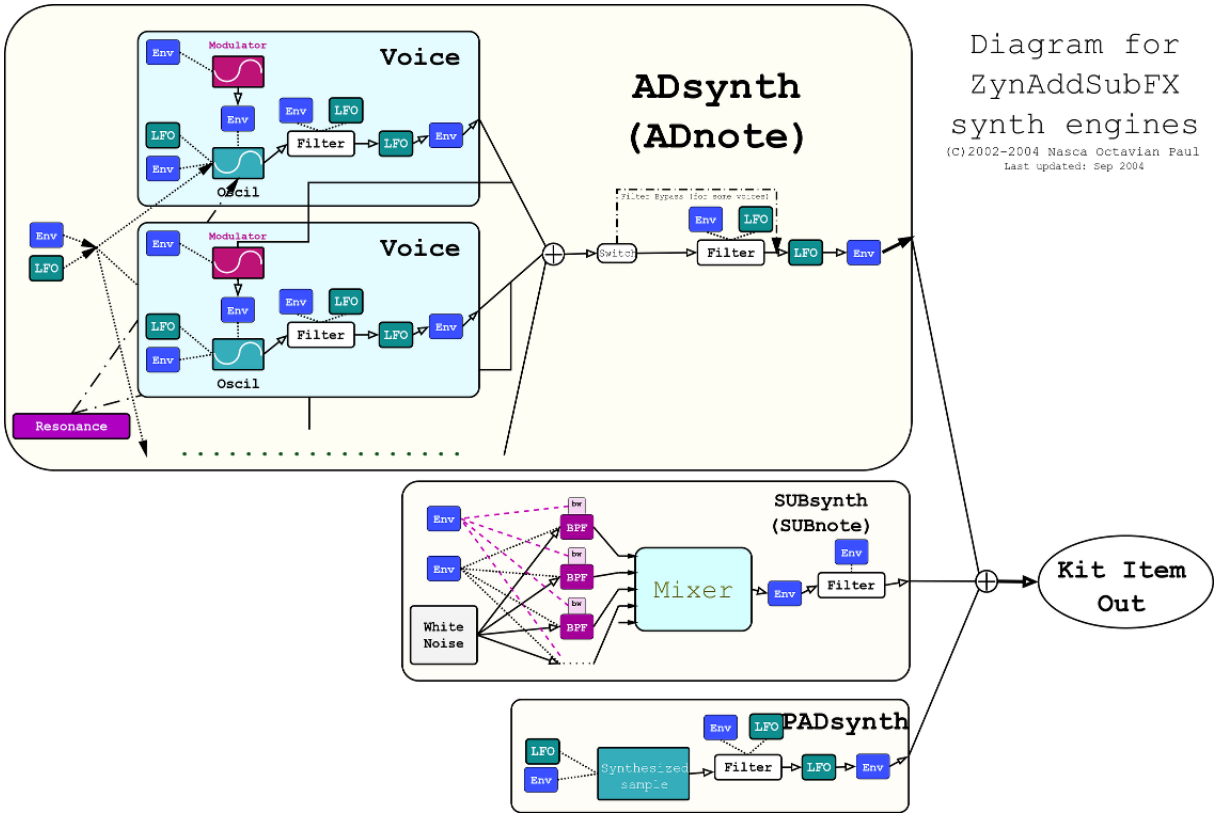

Figure 4: ZynAddSubFX/Yoshimi Note Generation

The figure represents the synthesizer module components. The continuous lines are the signal routing, and the dotted lines are frequency controlling signals (they control the frequency of something). The dashed lines control the bandwidths of bandpass filters. "Env" are the envelopes, "LFO" the Low Frequency Oscillators, "BPF" are band pass filters, "bw" are the bandwidth of the BPF's. If one uses instrument kits, the "note out" represents the output of the kit item.

2.4.4 Concepts / Basic Synthesis / Harmonics

Harmonics are sine waves that are multiple of the base frequency of a note. *Yoshimi* and *ZynAddSUbFX* introduce the concept of increasing the bandwidth of a harmonic so that it is not quite a sine wave.

2.4.4.1 Harmonic Bandwidth

"Harmonic bandwidth" does not refer to sample-rate, it refers to the frequency "spread" of each harmonic. This is the most important principle of making instruments that sound good. Unfortunately there is very little documentation about it.

Often it is believed that the pitched sounds (like piano, organ, choir, etc.) for a single note have a frequency, but it's actually harmonics and nothing more. Many people try to synthesize a sound using an exact frequency plus the harmonics, and observe that the result sounds too "artificial". They might try to modify the harmonic content, add a vibrato, tremolo, but even that doesn't sound "warm" enough. The reason is that the natural sounds don't produce an exact period; their sounds are quasi-periodic. Please note that not all quasi-periodic sounds are "warm" or pleasant. (Nasca's discussion of periodic vs. quasi-periodic, and the figures he shows, are not included here.) Basically, by slightly increasing the bandwidth of a periodic sound, it is possible to make it quasi-periodic.

A very important thing about bandwidth and natural sounds is that the bandwidth has to be increased if one increases the frequency of the harmonic. If the fundamental frequency is 440 Hz and the bandwidth is 10 Hz (that means that the frequencies are spread from 435 to 445 Hz), the bandwidth of the second harmonics (880Hz) must be 20 Hz. A simple formula to compute the bandwidth of each harmonic if one knows the bandwidth of the fundamental frequency is $BWn = nbw1$, where `n` is the order of the harmonic, `bw1` is the bandwidth of fundamental frequency and `BWn` is the bandwidth of the n'th harmonic. If one does not increase the bandwidth according the frequency, the resulting instrument will (usually) sound too 'artificial' or 'ugly'. There are at least three methods of making good sounds with the above considerations:

1. **Detuning**. By adding slightly detuned sounds (in *Yoshimi* it is called "ADDsynth"). The idea is not new: it has been used for thousands of years in choirs and ensembles. That's why choirs sound so beautiful.
2. **Noise sculpting**. By generating white noise, subtracting all harmonics with band-pass filters and adding the results (in *Yoshimi* it is called "SUBsynth").
3. **Generation by spectrum**. By "drawing" the above graph that represents the frequency amplitudes on a large array, put random phases and do a single IFFT for the whole sample.

2.4.4.2 Harmonic Amplitude

An important principle of natural harmonics is to decrease the amplitude of higher harmonics on low-velocity notes.

All natural notes have this property, because on low-velocity notes there is not enough energy to spread to higher harmonics. On artificial synthesis one can do this by using a low-pass filter that lowers the cutoff frequency on notes with low velocities or, if one uses FM (frequency modulation), by lowering the modulator index. The spectrum of the sound should be almost the same according to the frequencies and not the harmonics.

This means that, for example, the higher the pitch is, the smaller the number of harmonics it will contain. This happens in a natural instrument because of the resonance. In this case there are many instruments that don't obey this, but sound quite good (example: synth organ). If one records the C-2 note from a piano and one plays it at a very high speed (8 times), the result will not sound like the C-5 key from the piano. The pitch is C-5, but the timbre is very different. This is because the harmonic content is preserved (the n-th harmonic will have the same amplitude in both cases) and not the spectrum (eg. the amplitudes of the harmonics around 1000 Hz are too different from one case to another).

In artificial synthesis one can use filters to add resonance or FM synthesis that varies the index according to the frequency. In *Yoshimi* one can add the resonance:

1. **ADDsynth**: Use the Resonance, a high harmonics sound content, and filters or FM.
2. **SUBsynth**: Add some harmonics and use the Global Filter.

2.4.5 Concepts / Basic Synthesis / Randomness

The main reason why the digital synthesis sounds too "cold" is because the same recorded sample is played over and over on each key-press. There is no difference between a note played the first time and second time. Exceptions may be the filtering and some effects, but these are not enough. In natural or analog instruments this doesn't happen because it is impossible to reproduce exactly the same conditions for each note. To make a warm instrument one must make sure that it sounds slightly different each time. In *Yoshimi* one can do this:

1. **ADDsynth**: Set the "Randomness" function from Oscillator Editor to a value different than 0, or change the start phase of the LFO to the leftmost value.
2. **SUBsynth**: All notes already have randomness because the starting sound is white noise.
3. **PADSynth**: The engine starts the sample from random positions on each keystroke.

In setting the randomness of the oscillator output, there are two types of randomness. The first is *group randomness*, where the oscillator starts at a random position. The second is *phase randomness*: from -64 (max) to -1 (min) and each harmonic (the oscillator is phase distorted) is from 1 (min) to 63 (max). 0 is no randomness. One could use this parameter for making warm sounds like analog synthesizers.

See the ADDSynth oscillator editor, section 10.4.5 "ADDsynth / Voice Parameters / Voice Oscillator" on page 145, for this kind of control, named **Ph.rnd** or **rnd**.

There is now the possibility to add a 'naturalising' random pitch element to a part. This is found in the part-edit window. The settings are not currently saved, but will be once the control values are settled, and there has been enough experience to decide whether it should be a part or voice setting. (In the newer versions of *Yoshimi*, see the **Humanise** setting in the part-edit window.

2.4.6 Concepts / Basic Synthesis / Components

Important: All indexes of MIDI Channels, Parts, Effects starts from 0, so, for example, the first Part is 0. However, in other discussions of MIDI, part numbers, programs, or channels are often described as starting from 1.

Yoshimi components:

1. **Parts**. They receive the note messages from MIDI Channels. One may assign a part to any channel. A part can store only one instrument. "Add.S" represents ADDsynth and "Sub.S" is SUBsynth. In recent versions of *Yoshimi*, the number of parts available has been increased from 16 to 64.
2. **Insertion Effect**. This effect applies only to one part; one can have any number of insertion effects for one part, but the number of these cannot be bigger than NUM.INS.EFX.
3. **Part Mixer**. Mixes all parts.
4. **System Effects**. Applied to all parts, one can set how much signal is routed through a system effect.
5. **Master mixer**. Mixes all outputs of Parts Mixers and System Effects.

2.4.7 Concepts / Basic Synthesis / Filters

Yoshimi offers several different types of filters, which can be used to shape the spectrum of a signal. The primary parameters that affect the characteristics of the filter are the cutoff, resonance, filter stages, and the filter type.

Cutoff: This value determines which frequency marks the changing point for the filter. In a low pass filter, this value marks the point where higher frequencies are attenuated.

Resonance: The resonance of a filter determines how much excess energy is present at the cutoff frequency. In *Yoshimi*, this is represented by the Q-factor, which is defined to be the cutoff frequency divided by the bandwidth. In other words higher Q values result in a much more narrow resonant spike.

Stages: The number of stages in a given filter describes how sharply it is able to make changes in the frequency response. The affect of the order of the filter is roughly synonymous with the number of stages of the filter. For more complex patches it is important to realize that the extra sharpness in the filter does not come for free as it requires many more calculations being performed. This phenomena is the most visible in SUBsynth, where it is easy to need several hundred filter stages to produce a given note.

The **Q**: value of a filter affects how concentrated the signals energy is at the cutoff frequency. For many classical analog sounds, high Q values were used on sweeping filters. A simple high Q low pass filter modulated by a strong envelope is usually sufficient to get a good sound.

Filter Type: There are different types of filters. The number of poles define what will happen at a given frequency. Mathematically, the filters are functions which have poles that correspond to that frequency. Usually, two poles mean that the function has more "steepness", and that one can set the exact value of the function at the poles by defining the "resonance value". Filters with two poles are also often referenced as Butterworth filters.

For the interested reader, functions having *poles* means that we are given a quotient of polynomials. The denominator has degree 1 or 2, depending on the filter having one or two poles. In the file `DSP/AnalogFilter.cpp`, `AnalogFilter::computefiltercoefs()` sets the coefficients (depending on the filter type), and `AnalogFilter::singlefilterout()` shows the whole polynomial (in a formula where no quotient is needed).

Filters are thoroughly described in section 6.2 "Filter Settings" on page 71.

2.4.8 Concepts / Basic Synthesis / Envelopes

Envelopes are long-period wave forms that are applied to frequency, amplitude, or filters. Envelopes generate effects such as tremolo and vibrato, as well as effects that occur when a sound-generating physical component changes shape. Envelopes are thoroughly described in section 6.5 "Envelope Settings" on page 84.

2.5 Concepts / MIDI

It is useful to discuss some of the details of MIDI in order to understand *Yoshimi*. Obviously, we assume some knowledge already, or one wouldn't be running *Yoshimi*.

Table 1: ZynAddSubFX/Yoshimi MIDI Messages

0 or **32**	Bank Change (user selectable, does *not* force a program change)
1	Modulation Wheel
2	Breath Control
6	Data MSB
7	Volume
10	Panning
11	Expression
38	Data LSB
64	Sustain pedal
65	Portamento
71	Filter Q (Sound Timbre)
74	Filter Cutoff (Brightness)
75	BandWidth (different from GM spec)
76	FM amplitude (different from GM spec)
77	Resonance Center Frequency (different from GM spec)
78	Resonance Bandwith (different from GM spec)
96	Data Increment
97	Data Decrement
98	NRPN LSB
99	NRPN MSB
120	All Sounds OFF
121	Reset All Controllers
123	All Notes OFF
192	Program Change (voices 1-128; see notes below)
224	Pitch Bend (see notes below)

2.5.1 Concepts / MIDI / Messages

Yoshimi responds to the following MIDI controller messages (1). This list seems to be more extensive than that presented in the online *ZynAddSubFX* documentation.

For the controllers (numbers 75 to 78) that are not defined in GM:

- **Bandwidth** control (75) increases or decreases the bandwidth of instruments. The default value of this parameter is 64.
- **Modulation amplitude** (76) decreases the amplitude of modulators on ADDsynth. The default value of this parameter is 127.
- **Resonance Center Frequency** control (77) changes the center frequency of the resonance.
- **Resonance Bandwidth** control (78) changes the bandwidth of the resonance.

Some standard MIDI messages are handled a bit differently by *Yoshimi*:

- **Program Change (192)** also provides user selectable CC for voices 128-160. There is now an option to make Program Change enable a part if it's currently disabled.
- **Key pressure (aftertouch)** is internally translated as CC 900.
- **Channel Pressure** is internally translated as CC 901.

- **Pitch Bend** is internally translated as CC 1000.
- **Modulation** The wheel only affects AddSynth and PadSynth, and then only the frequency LFO depth. Just to make it more confusing, it changes the level from 0 up to it's current (GUI) setting only. Therefore, if the LFO depth is set to zero, the Mod Wheel will have no effect.
- **User selectable CC for Bank Root Path change** For more details of bank changes see section 2.3 "Concepts / Banks and Roots" on page 20.

Instruments inside banks should *always* have file-names that begin with four digits, followed by a hyphen. Otherwise the results can be rather unpredictable.

2.5.2 Concepts / MIDI / NRPN

NRPN stands for "Non Registered Parameters Number". NRPNs can control all System and Insertion effect parameters. Using NRPNs, *Yoshimi* can now directly set some part values regardless of what channel that part is connected to. For example, one may change the reverb time when playing to keyboard, or change the flanger's LFO frequency.

NRPNs are described in greater detail in section section 15 "Non-Registered Parameter Numbers" on page 177.

2.5.2.1 Concepts / MIDI / NRPN / Vector Control

Vector control is a way to control more than one part with the controllers. Vector control is only possible if one has 32 or 64 parts active in *Yoshimi*. In vector mode, parts will still play together but the vector controls can change their volume, pan, filter cutoff in pairs, controlled by user defined CCs set up with NRPNs.

Vector control is described in greater detail in section section 16.2 "Vector / Vector Control" on page 184.

2.5.2.2 Concepts / MIDI / NRPN / Effects Control

NRPNs are very useful in modifying the parameters of the *Yoshimi* effects.

Effects control is described in greater detail in section section 15.2 "NRPN / Effects Control" on page 179.

2.6 Concepts / Command Line

This section covers a few terms useful in discussing the command line.

2.6.1 Concepts / Command Line / level

The term **level** is used in the description of the new command-line facility of *Yoshimi*. A **level** is simply a related group of parameters or a location where one can go to for making changes. Important levels are: the top level; system effects; part; and more.

2.7 Concepts / LV2 Plugin

Yoshimi now runs as an LV2 plugin.

TODO: Describe LV2 at a high-level.

3 Configuration Files

Let's cover the configuration files, which have expanded in utility in recent versions of *Yoshimi*. Understanding these configuration file makes it easier to use *Yoshimi*.

As with most applications, *Yoshimi* and *ZynAddSubFX* allow for one to save one's work and reload it. *Yoshimi* has a number of different files that make up the current configuration. Together, they make up the concept of a *patch set* (also called a *patchset*). Sometimes one will see reference to a "session", but that term is too easy to confuse with the "session" in "JACK session manager". Here are the file extensions used for saving the *Yoshimi* patch-set data:

Here is a summary of the files. Please note that the names all start with `yoshimi`. For example, `.banks` is really `yoshimi.banks`.

- `.banks` Contains information on the accessible instrument banks.
- `.config` Contains information to translate between bank directory names and bank ID values.
- `.history` Recent patch sets are now stored in the `.history` file. (Does this include command history?)
- `.instance(n)` Contains the current root/bank, MIDI settings, and preferred engines.
- `.state` Contains the state information needed to duplicate a *Yoshimi* session that was saved.
- `.windows` Contains the current layout of windows for reinstantiation at the next startup of *Yoshimi*. If there is no such file (`~/.config/yoshimi/yoshimi.window`) at *Yoshimi*, then the keyboard is also opened, alongside the main window, as a help to those new to *Yoshimi*. And of course that state will be saved, if present, when *Yoshimi* exits.
- `.xiz` An Instrument file.
- `.xmz` A full *Yoshimi* configuration; everything. This file is also called a *parameter set*.
- `.xpz` Presets. A preset is a *Yoshimi* sub-setting file.
- `.xsz` Scale Settings.
- `.xvy` Vector settings. The extension stands for "Xml Vector Yoshimi". It will eventually be integrated with the saved states. For a good example, see section 16.3 "Vector / Command Line" on page 186.

The entire config set should then be (ignoring the prepended `yoshimi`):

- `.config`
- `.instance[n]`
- `.history`
- `.banks` (this is currently per instance)

The `.windows` file is specific to the GUI, so doesn't figure in this scheme at all.

3.1 Configuration Files / Patch Set

A patch set is basically a group of instruments related simply by the user wanting to have them all loaded at once into *Yoshimi*. A patch set is stored in a `.xmz` file. A patch set is akin to a preset, in that it stores a combination of items, that took awhile to set up, for easy retrieval later.

This is the full *Yoshimi* configuration. Everything. It's format is either XML or compressed XML, as explained elsewhere. The **Parameters / Save** menu entry saves files with this extension. This file is also called a *parameter set*.

One of the simplest ways to save one's work is to save the entire *Yoshimi* configuration. This saving can be done through the **Patch Sets** menu (the **File** menu in *ZynAddSubFX*), and will result in the creation of a `.xmz` file. Once created, this file will hold the settings for all settings within that setup, such as microtonal tunings, all patches, system effects, insertion effects, etc. See section 5.1.4.1 "Menu / Yoshimi / Settings / Main Settings" on page 36.

In many cases saving everything in a part is not what is desired. Saving a patch later on in an editing session is one such example. In order to save a patch, one can either save it from the **Instruments** menu, or through the **Bank** window.

3.2 Configuration Files / Config

Often, one will see the extension `.config` used in the `$HOME/.config/yoshimi` directory. This file once contained information to translate between bank directory names and bank ID values. In recent versions of *Yoshimi*, this file is much reduced in size, and its "doctype" is no longer "ZynAddSubFX".

The `.config` file is always going to be specific to one machine and working modes, so no one will ever want to copy it across even to another *Yoshimi* environment. Recent patch sets are now no longer stored in the main `.config` file, but in a new `.history` file. The `.config` file is now a much reduced common settings – interfaces, sample rate – file. It is a single file that every instance can read, but only the first one can write.

`.config` hasn't yet been separated from `.instance(n)` and all files are still per instance, as the *Yoshimi* team haven't had time to work out exactly how to manage common files and memory locations for those that should be shared. The `.config` file is saved only when the user explicitly calls for it to be saved. *Yoshimi* will still mention its absense:

```
$ yoshimi -a -A
Yoshimi is starting
ConfigFile /home/ahlstrom/.config/yoshimi/yoshimi.config not found, will
   use default settings ...
```

The `.config` file will be readable by all instances of *Yoshimi*, but writeable only by the main instance. The relevant controls will be greyed out in the other instances. The `.config` and `.banks` data now reside in separate configuration files. The banks file is saved every time there is a normal exit, so the last-used root and bank IDs will always match what that instance thinks is there. Conversely, the main `.config` file *doesn't* get saved when one starts a new (unkown) instance of *Yoshimi*, but the config-changed flag is set, so one has control over whether any settings are saved. So now, if anything goes wrong with the config files they won't corrupt one's carefully organised bank files, and vice-versa.

3.3 Configuration Files / State

Sometimes one will see the extension `.state` used in the `$HOME/.config/yoshimi` directory. These files contain a lot more information, that needed to duplicate a *Yoshimi* session that was saved. Seems to be a superset of an `.xmz` file, saving everything. This is a facility that is peculiar to *Yoshimi*. The state file is accessed from the **State** menu item in the main window. Its default name is `~/.config/yoshimi/yoshimi.state`.

The *Yoshimi* 'state' file consists of the entire setup, from basic configuration settings to currently-loaded instrument sets. However, upon investigating some JACK session managers, it looks like they don't want

(or can't use) most of the configuration information because they are expecting to be able to change the state in *running* instances.

If and when *Yoshimi* gets around to splitting the 'instance' data from the main configuration, then a lot of this session issue can be resolved buy saving only the 'true' configuration locally, and to the state save. However, the 'instance' data will include things like ALSA/JACK settings. Currently we can't change these live (although it would be nice if we could), but would anyone want to do so from a JACK session manager?

3.4 Configuration Files / Instrument

An Instrument. These files can have two formats, compressed and uncompressed. With the **Instrument** menu, one can just save the file to any given location with the `.xiz` extension.

3.5 Configuration Files / Scale

Scale Settings. These files store microtonal settings that *Yoshimi* can use to produce non-standard musical scales. Recent scales settings are saved and recorded.

3.6 Configuration Files / Presets

Have a favorite setting for an envelope, or a difficult-to-reproduce oscillator? Then presets are for you! Presets allow for one to save the settings for any of the components which support copy/paste operations. This is done with preset files (`.xpz`), which get stored in the folders indicated by *Paths / Preset Dirs...*. The key thing about using presets is that one must first specify a presets directory! Otherwise, who knows where they go? A good choice for a preset directory is `~/.config/yoshimi/presets`.

In *Yoshimi*, a *preset* is any collection of settings that can be saved to the clipboard or to a file, for later loading elsewhere.

A preset is canned version of a *Yoshimi* sub-setting. Presets can be copied and pasted using the **C** and **P** user-interface buttons associated with many of the *Yoshimi* dialog windows. They make it easy to save portions of the current settings for later use. For example, resonance settings can be saved.

The naming convention for a preset file is `presetname.presettype.xpz`, where *presename* is the name one types into the **Copy to Preset** name field, *presettype* is the name that appears in the **Type** field, and *xpz* is the file-extension for compressed XML preset files.

3.7 Configuration Files / Instance

A new feature of the *Yoshimi* configuration. It contains the current root/bank, MIDI settings, and preferred engines. These instance files are totally independent files, distinguished by a number in the file-name.

3.8 Configuration Files / History

A new feature of the *Yoshimi* configuration. Recent patch sets are now stored in the `.history` file. For example, if the **XML Compression** option is set to 0, and one exits *Yoshimi*, then the file `~/.config/yoshimi/yoshimi.history` might contain the following items (ignoring the XML markup):

```
/home/me/yoshimi-cookbook/sequencer64/b4uacuse/yoshimi-b4uacuse-gm.state
/home/me/sequencer64/contrib/yoshimi/horse.state
```

`.history` is a single file that every instance can read and write. The `.history` file is saved only upon a normal exit, as it is comparatively unimportant.

The history is a single buffer and file, readable and writeable by all instances. This is actually quite interesting as there can never be a conflict. It is impossible to have two browser lists open at the same time (try it!) and the lists are always rebuilt from memory every time they are opened. Similarly, the histories are added too every time a new recognised file is loaded or saved and one can't physically do two at the same time – even if one could it would simply mean that one very briefly waited for the other, which is not an issue as they are not in the realtime thread.

Now, there is also another "history" file apparently created by *Yoshimi*: `~/.yoshimi_history`. This file seems to be a history of commands typed into the command-line interface of *Yoshimi*. TO BE DETERMINED.

3.9 Configuration Files / Banks

A new feature of the *Yoshimi* configuration. Currently each *Yoshimi* instance takes its own copy of the actual files as it starts up. However, they can all save, delete, or rename the actual files without talking to the other instances, so one can move a file in one instance, and then try (and fail) to access it from another.

With the **Banks** menu, one can assign a patch to a given slot with a bank. This instrument will remain in that slot for future use until it is deleted. To see the physical location of the `.xiz` file, one should check the **Yoshimi / Settings / Banks / Root Dirs** (*FileSettingsBank_Root_Dirs*) window to see the paths for banks.

At startup, after all the configuration is complete, the banks are loaded and installed. On a per instance basis, the first thing this process does is look for a `yoshimi(-n).banks` file, if it can't, find that it then hunts for a `yoshimi(-n).config` file, and if that fails it does a rescan for banks. In this way it should be completely backward compatible with any previous config files.

The `.banks` file is *saved* every time roots, banks, or instruments are changed, and again on a normal exit to catch the current root and bank (which don't otherwise trigger a save). This allows the last-used root and bank IDs to always match what that instance thinks is there. Note that one needs to have write permissions to add instruments to the bank.

Banks are more thoroughly described in section 2.3 "Concepts / Banks and Roots" on page 20.

3.10 Configuration Files / Windows

No, this term isn't a reference to "that other operating system".

A new feature of the *Yoshimi* configuration. It saves the current layout of windows for reinstantiation at the next startup of *Yoshimi*.

3.11 Configuration Files / Format

The Unix `file` command indicates that the XML files are one of two types:

- *exported SGML document, ASCII text.* These files are unindented XML data with an encoding of UTF-8 and a DOCTYPE of "ZynAddSubFX-data".
- *gzip compressed data, from Unix.* These files can be renamed to end in ".gz", and then run through the `gunzip` program to yield the XML file (but without an `.xml` extension).

The format depends on the "XML compression level" option discussed in section 5.1.4.1 "Menu / Yoshimi / Settings / Main Settings" on page 36.

Saving settings or not: If one changes settings, and closes without saving, that means the settings remain in place only for the current session. If one has changed anything, when one closes *Yoshimi*, one will be given a second chance to save them. If one responds 'No', the next time *Yoshimi* starts, the old settings will be restored. An 'undo' feature would get pretty crazy very quickly.

In the **Settings** window, **Save Settings** refers to the entire window, not just individual tabs. The close buttons are actually outside the frame of the tabs.

Close without saving doesn't mean revert to previous settings; it means to use the changes, but don't immediately store them to the filesystem.

In general, the contents are structured a lot like the user-interface elements that are used to set them.

4 Banks and Roots

In recent versions of *Yoshimi*, the concepts of banks and roots have undergone a fair amount of change, including new features to make them easier to manage and easier to automate. There are a lot of details to understand, too many to include along with the descriptions of the user-interface elements that control them. Therefore, this new section is devoted to banks and roots. It is an elaboration of material originally presented in section 2.3 "Concepts / Banks and Roots" on page 20.

At one time, one could in theory have 1000 roots, 1000 banks and 1000 presets. However, now roots and banks were trimmed to what can be addressed from MIDI. One can supposedly still have 1000 presets, though. Anyway, 128x128x160 = 2621440 instruments should be enough for anyone.

All root, bank, and instrument IDs are used by MIDI controls, and as of version 1.3.6 will also be accessible to the command line.

The file `Banks.txt` in the *Yoshimi* source-code bundle makes an important point about a transition (in newer versions) to tagging roots (directories) and banks with an ID code:

> One no longer has the concept of a default root directory, but a current one. This can by changed at any time without requiring a restart, so there is no longer a need to display the (confusing) contents of all roots. Also, roots now have ID numbers associated with them, but no changes have been made to the actual directores to achieve this. Instead the IDs are stored in the config file. The same ID system is used for banks, again without making any file system changes.
>
> At first run (and whenever new root directories are set) unknown roots and banks are given these IDs. Once set they will not change no matter how many more roots and banks are later added. One can however, manually change root directory IDs in the 'Bank Root Paths window' accessible from the 'Paths' item in the main window. Also, there is a new Banks window so that these can be set up, moved and renamed in exactly the same way as instruments can. With these IDs, roots and banks can be grouped/ordered by function instead

of alphabetically. When using the GUI, one will always know exactly which root and bank one fetches an instrument from.

One can quickly step between roots, banks and instruments with the so marked buttons in each of these, and if one right-clicks on them one will close one window as one opens the other.

The significance of all this is that one's MIDI sequencer can now reliably use these ID numbers to select roots, banks and (already available) instruments. That Rosegarden or Muse file one saves today will be just as valid in the future, unless one makes the deliberate choice to change some IDs. Indeed, one can now start with an 'empty' Yoshimi, and via MIDI, set roots, banks and load instruments into parts (enabling the parts as one does so) swapping banks and roots as necessary. While the MIDI file runs it can silently pull instruments from any root/bank into any non-sounding part without disturbing the playing ones.

In Yoshimi / Settings / MIDI one can enable or disable all these features, and can define which CCs one wants to use. Bank can be either MSB or LSB (as before). Root can be any non-reserved CC but including the one not in use for Bank. Also, Extended Program Change now has the same restrictions as Root, and these three are all cross-checked against each other. As an example, one might set Bank to LSB amd Root to 0 (MSB), effectively giving one extended bank control compatible with all sequencers.

Also, different instances have their own config files so that one can have (say) the main instance with current root(9), bank(23) while instance 4 has current root(2), bank(6). One can call up instances by number and thus access saved settings for that instance. As each instance has its own MIDI and audio ports, they can behave more-or-less independently.

In doing all of this we have completely changed the way we manage the structure internally, resulting in much greater efficiency, at the cost of only a slightly slower startup. Swapping roots performs *no* file operations. Swapping banks only fetches the directory list of the newly selected bank. Changing an instrument doesn't have to search for a file, only load from its already known location.

If one changes a bank root path, either through the gui or via MIDI, it will always reset the current bank to the lowest numbered one it can find. This is because there may not be a bank in the new root with the same ID, and even if there is, there is no guarantee that it will have the same name or contents.

Also if an attempt is made to reload the same root, nothing will actually happen. The same is true of banks. Both of these are kept fully up-to-date so there would be no point.

However, reloading the same *instrument* will be performed every time, as one may have changed what is currently loaded without saving it. This provides an effective 'restore' operation.

Finally, it is generally advisable to make root and bank changes on channel 1 so that one can more easily keep track of them. However they are not channel sensitive as they don't directly affect the sound, so one can set them in any convenient channel then perform individual program changes on the desired channels.

A "CC" is a MIDI "continuous controller". A MIDI bank change is usually a CC#0 value of 0, followed by a CC#32 value of X, where X is the desired bank number from 0 to whatever. (However, in some cases it may simply be a CC#0 on its own with a value of X). Many synths also require that one send a program change after the bank change, to select the program within the bank.

4.1 Roots

In *Yoshimi*, a *root* is a location in which banks can be stored. It is basically a directory, though it ultimately is assigned a number by *Yoshimi*, presumably to be able to access it in an automated way. By choosing a root and making it the current root, one can hone in on a smaller collection of banks.

One cannot reach root paths through the **Yoshimi / Settings** menu any more; it was causing a nightmare of syncing with the other entry routes. One can reach it from the Banks window or the Instruments window, and both of the latter also have multiple entry routes. Roots can also be reached through the **Yoshimi / Paths** menu.

4.2 Banks

Another important concept in *Yoshimi* is *banks*. Instruments can be stored in banks. These are loaded and saved automatically by the program. On program start, the last used bank is loaded. A single bank can store up to 128 instruments normally, and 160 using extended programs.

Also note that, as well as bank and program changes, there is the ability to set a MIDI CC to access the voices from 129 to 160 (numbered re 1). All the Bank controls are contained in a tab in the main **Settings** window, and take immediate effect.

Bank root directories are identified with ID numbers that can be changed by the user in the user-interface. This feature is also made available for selection over MIDI. MIDI only sees banks in the *current* root directory, but all banks are accessible to the user-interface.

4.2.1 Bank Directories

Banks are arranged in directories, with each directory containing a number of instrument files.

Each instrument's file-name should begin with a 4-digit number (left-padded with 0's to make it 4 digits long). This number can serve as a MIDI patch number for automated selection of the instrument via a MIDI program-change message.

Unnumbered instruments in a bank will be given a temporary ID starting from number 160 and working down. If those numbers already exist then they will be skipped over. This can get very confusing. However, if one simply loads it and re-saves it to the same instrument slot, it will gain that ID and be properly fixed. One can then move-swap it with others.

5 Menu

The *Yoshimi* menu, as seen at the top of figure 2 "Yoshimi Main Screen, 1.3.8 and Above" on page 17, is fairly simple, but it is important to understand the structure of the menu entries.

5.1 Menu / Yoshimi

The *Yoshimi* menu entry contains the sub-items shown in figure 5 "Yoshimi Menu Items" on page 34. The next few sub-sections discuss the sub-items in the *Yoshimi* sub-menu. (Note that, in *ZynAddSubFX*, this menu is called the *File* menu.)

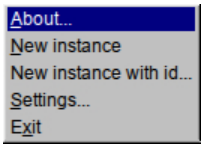

Figure 5: Yoshimi Menu Items

Bug: There seems to be a bug in that the expected menu hot-keys (Alt-Y, Alt-I, Alt-P, and Alt-S) do not work (Yoshimi 1.3.5).

5.1.1 Menu / Yoshimi / About...

There is no **Help** menu in *Yoshimi*. Therefore, the **About** dialog appears in the **Yoshimi** menu, as shown in figure 6 "Yoshimi Menu, About Dialog" on page 35. These guys need some acknowledgment for their hard work! And they acknowledge the massive groundwork laid by the *ZynAddSubFX* project.

Figure 6: Yoshimi Menu, About Dialog

5.1.2 Menu / Yoshimi / New instance

Creates a new instance of *Yoshimi*. If JACK is running, start a normal (JACK-using) instance of *Yoshimi*. Then use this menu entry. *Yoshimi* will start another instance of itself, with an ID of 1. This instance can be verified by running a JACK session manager such as QJackCtl.

It is important to note that each instance of *Yoshimi* has its own configuration file. Each also has its own MIDI and audio ports. Thus, these instances are partly independent of each other.

Opening a new instance creates a copy that has it's own dynamic memory for running storage. In the future, some data (such as recent history) will be shared between instances. This will be done only where instances actually need to be in sync.

Each instance has it's own file-store in *Yoshimi*'s configuration directory. The means that if one opens a numbered instance, one will get back all the settings that were previously used for that instance.

Instances no longer fight for access to JACK/ALSA audio; they will simply try to find another route to a soundcard. Failing to find one, they will revert to null audio, but will nonetheless start cleanly.

5.1.3 Menu / Yoshimi / New instance with id...

Creates a new instance of *Yoshimi* with an ID that is a number. See figure 7 "Yoshimi Menu, Instance Dialog" on page 36. It tries to open a *Yoshimi* instance based on the configuration found in the file `~/.config/yoshimi/yoshimi.configXX`, where *XX* is the ID one supplied.

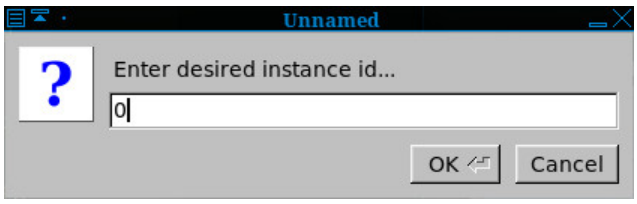

Figure 7: Yoshimi Menu, Instance Dialog

Useful when connecting devices with JACK. Start a normal (JACK-using) instance of *Yoshimi*. Then use this menu entry, supply a number as an ID. *Yoshimi* will start another instance of itself, with an ID of whatever number one specified. This instance can be verified by running a JACK session manager such as *QJackCtl*.

In a non-JACK setup it won't fail, but in the absence of any specific setting, it will have null audio, but (probably) will still connect to ALSA MIDI. Not too useful, but we should test that sometime.

5.1.4 Menu / Yoshimi / Settings...

The *Yoshimi Settings* dialog contains four tabs that control the major and overall settings of *Yoshimi*. At the bottom of this dialog are two buttons: **Save and Close** and **Close Unsaved**.

Please note that the **Save and Close** and **Close Unsaved** buttons apply to the *whole* **Settings** window. Furthermore, the "saving" does *not* refer to preserving the changes that have been made in any of the tabs for the current *Yoshimi* session. Any changes made in **Settings** always remain in place for the current *Yoshimi* session. However, the changes to the settings are saved to the state file *only* if **Save and Close** is clicked.

1. Save and Close. This selection saves the settings made in **all** of the tabs to the state file, and closes the *Yoshimi* settings dialog.

2. Close Unsaved. Close Unsaved, Main Settings.

This selection closes the *Yoshimi* settings dialog. However, note that any changes made in the tabs *are preserved*. They are preserved for the current *Yoshimi* session, but are not saved to the filesystem.

5.1.4.1 Menu / Yoshimi / Settings / Main Settings

The Main Settings tab controls the main configuration items that follow, which apply to all patches/instruments. The main settings are shown in figure 8 "Yoshimi Main Settings Tab" on page 37.

Some settings only take effect after restarting the synthesizer. In Main settings, only items marked with an asterisk ('*') need a restart. (This in now only the first two. Some of the other ones had prevously been wrongly marked.)

The settings dialogs are quite different between *ZynAddSubFX* and *Yoshimi*. There are some differences even between *Yoshimi* versions earlier than 1.3.5, and the current version (currently 1.3.9).

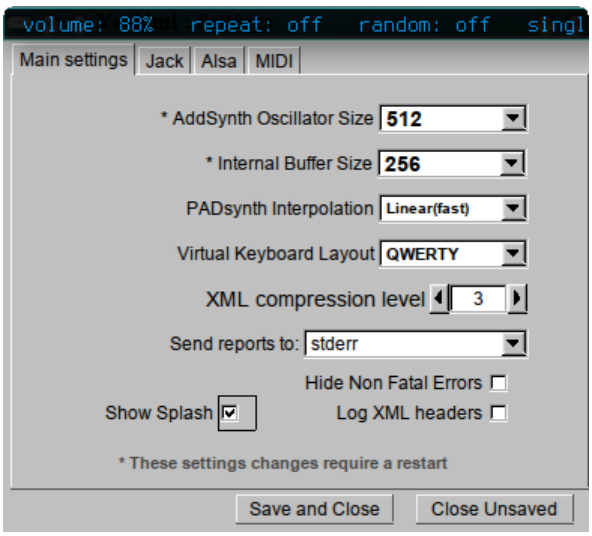

Figure 8: Yoshimi Main Settings Tab

The following settings exist in the *Main settings* tab:

1. **AddSynth Oscillator Size** (was "OscilSize")
2. **Internal Buffer Size** (new)
3. **PADsynth interpolation**
4. **Virtual Keyboard Layout**
5. **XML compression level**
6. **Send reports to**
7. **Hide Non Fatal Errors**
8. **Log XML headers**
9. **Show Splash** (new)
10. **Save and Close**
11. **Close Unsaved**

1. AddSynth Oscillator Size. ADDsynth Oscillator Size (in samples). This item used to be called "OscilSize". Sets the number of the points of the ADDsynth oscillator. Bigger is better, but it takes more CPU time on the start of any note, and it may add latency to some processes.

The default value for *Yoshimi* is shown marked with an asterisk, and the default value for *ZynAddSubFX* is 512. This asterisk/plus-sign convention is used throughout this manual. See figure 9 "OscilSize Values" on page 37, shown below for the AddSynth Oscillator Size drop-down element.

Values: `256, 512*, 1024, 2048, 4096, 8192, 16384`

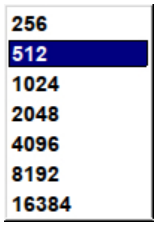

Figure 9: AddSynth Oscillator Size (samples)

2. Internal Buffer Size. This is a new item for version 1.3.6. It is actually the old **Period Size** field from the **Alsa** tab. It sets the granularity of the sound generation. To find out the internal delay in milliseconds, divide the buffer-size value by the sample-rate, then multiply the result by 1000: For example, $256/44100 * 1000 = 5.8ms$.

The default internal buffer size has been reduced from 1024 to 256. One gets better latency that way. Almost all modern computers can run *Yoshimi* with the current default (smaller) buffer-size value, and many will do so at 64 frames (and even 16 frames!) without any special precautions.

Note that, for ALSA, if the audio destination is "default", then ALSA decides on the buffer size (among other settings), and *Yoshimi* will set it's internal buffer size to match, which always seems to be 1024.

Values: `64, 128, 256*, 512, 1024`

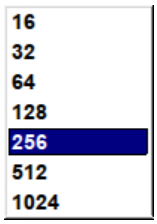

Figure 10: AddSynth Internal Buffer Size (samples)

3. PADsynth interpolation. See figure 11 "PADSynth Interpolation" on page 38, shown below, for the interpolation values. From an email conversation with Paul Nasca, Will notes that the sound improvement with cubic interpolation is quite subtle, and requires a well designed audio setup, a PADsynth instrument with a fair amount of high-frequency content... and good hearing!

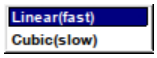

Figure 11: PADSynth Interpolation Values

Values: `Linear(fast)*, Cubic(slow)`

4. Virtual Keyboard Layout. The virtual keyboard is useful, but it is difficult to move the mouse rapidly to the next key on the virtual keyboard. Therefore, *Yoshimi* supports using the computer keyboard to produce notes.

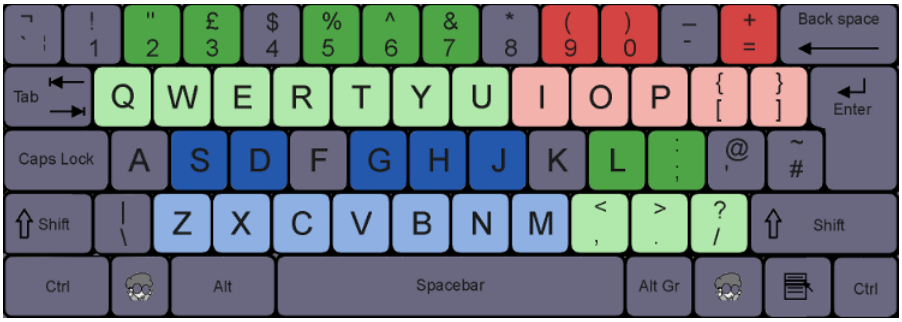

Figure 12: QWERTY Virtual Keyboard Layout

See figure 12 "QWERTY Virtual Keyboard Layout" on page 38, for the mapping of the computer keyboard to the virtual keyboard. Three octaves (blue, green, and red) are available, with the dark keys

of each color representing the "black" keys. Note that this is a QWERTY layout. *Yoshimi* also supports other keyboard layouts. See figure 13 "Virtual Keyboard Layout" on page 39, for the virtual keyboard layout settings drop-down.

Values: `QWERTY*, Dvorak, QWERTZ, AZERTY`

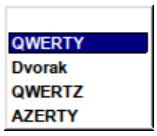

Figure 13: Virtual Keyboard Layout Values

5. XML compression level. Gzip Compression level of *Yoshimi* XML files. The settings and instruments of *Yoshimi* are preserved in XML files. The value of 0 indicates that the XML file is uncompressed.

In general, 0 is probably the best setting for debugging only. Setting this option makes the XML files a bit larger, perhaps larger by a factor of more than 10, making a 10K file into a 180K file. For a little "wasted" space and time, one can view the XML file in a text/programmer's editor. But, if one's system is tight on disk space, higher levels of compression can be specified. Using XML compression can also save file access time which may be beneficial if one's computer is borderline on latency. This setting should stay at 3 if one is going to save large patchsets that will later be loaded while running. Uncompressing is *much* faster than file loading.

Values: `0 to 9, 3*`

6. Send reports to. Notices and error messages can be sent to the standard error log of the terminal in which *Yoshimi* can be run, or, more usefully, to an output console window. Now these messages are pushed in reverse order, to avoid scrolling and to make the most recent statuses easily visible. See figure 14 "Send Reports" on page 39. It provides a depiction of the selection drop-down.

Values: `stderr*, Console Window`

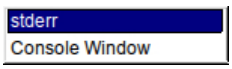

Figure 14: Send Reports To

If the **Console Window** option is chosen, then the **Reports** button in the effects panel is enabled. Pressing the **Reports** button brings up a small console dialog, as shown in figure 15 "Yoshimi Console Window" on page 39.

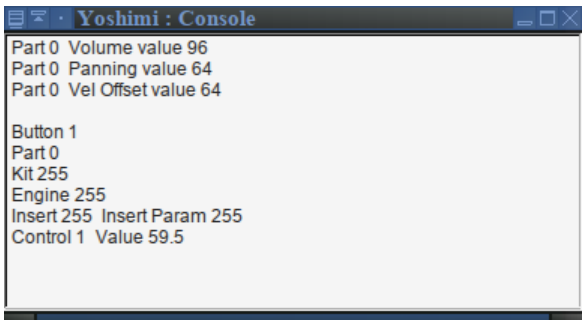

Figure 15: Console Window

An interesting feature of the console window is that one can identify user-interface elements of the *Yoshimi* configuration in this window by a middle-click on the user-interface element. The console window figure shows the results of middle-clicking on the following bottom-panel knobs in order:

1. Velocity Sense (left-click)
2. Velocity Offset
3. Pan
4. Volume

Information about each knob middle-clicked is pushed to the top of the windows in reverse order. Information about an earlier left-click is shown at the bottom of the figure. Each left-click ("Button 1") will increase the parameter represented by the knob, and each right-click ("Button3") will decrease the parameter represented by the knob, and each change is reported in the console window. And, of course, each middle-click changes nothing, but reports the current value in the console window. This powerful feature provides a way to gather the information needed to control the parameter from the command-line. The output can be selected, copied, and pasted to a script or archive text file for safe-keeping. Consider it a form of "MIDI learn" that will be developed in the future. And note that it applies to output to `stderr` as well.

7. Hide Non Fatal Errors. Especially when running from the command line (with reports going there too), under some circumstances one can get a swamp of low-level error messages (such as XRUNs) that is so large that one cannot work out what is going on. This feature disables these error messages; it is a work in progress to catch the bulk of them, while still reporting top-level messages and ones that cause a forced exit (surely not!)

8. Log XML headers. This item sends the information to the console window (or `stderr`) so that one can then see what *ynAddSubFX*/*Yoshimi* version created the file.

9. Show Splash. This item will speed up the start-up of *Yoshimi* slightly if unchecked, by not showing the splash screen while files are being loaded.

5.1.4.2 Menu / Yoshimi / Settings / Jack

JACK is the "Jack Audio Connection Kit", very useful for increasing audio performance and configurability. When using the JACK audio backend, instruments can be individually routed and sent to the main L/R outputs. This is controlled from the panel window, section 7.1 "Mixer Panel Window" on page 96, and the settings are saved with all the other parameters.

Direct part outputs carry the Part and Insertion effects, but not the System ones.

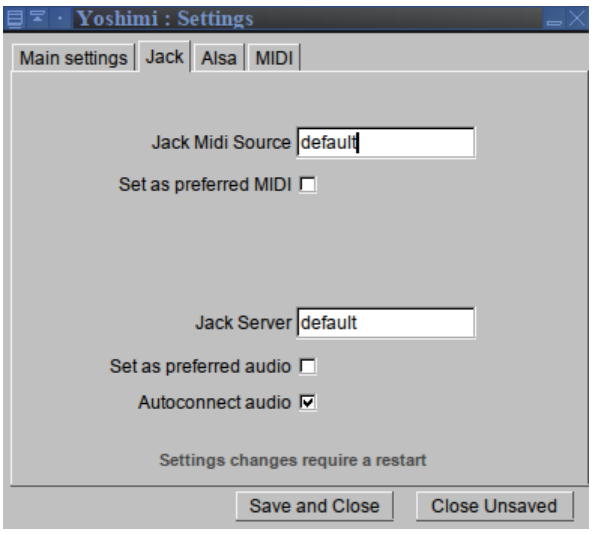

Figure 16: JACK Settings Dialog

The following items are provided by the Jack settings:

1. **Jack Midi Source**
2. **Set as preferred MIDI**
3. **Jack Server**
4. **Set as preferred audio**
5. **Autoconnect audio** (new, 1.3.9)
6. **Save and Close**
7. **Close Unsaved**

1. Jack Midi Source. Jack MIDI Source. It is possible to have more than one JACK MIDI source. This option tells this instance of *Yoshimi* which JACK client to try to auto-connect to for MIDI input. This option corresponds to the *Yoshimi* command line option `--jack-midi(=device)`.

Values: `default*, name` name, as in "jackd –name"

2. Set as preferred MIDI. Set as preferred MIDI for JACK. This setting determines which MIDI connections a particular instance will first attempt. The switches are mutually exclusive across JACK and ALSA, so if one checks ALSA for MIDI, it automatically unchecks JACK for MIDI. As well as from the GUI, this setting can be set (for instance 0) from the command line, both at start-up and once running.

3. Jack Server. Jack Server Name. It is possible to have more than one JACK server running. This option tells this instance of *Yoshimi* which JACK server to use. This option corresponds to the *Yoshimi* command line option `--jack-audio(=server)`.

Values: `default*, name` name, as in `jackd --name`.

4. Set as preferred audio. Set as preferred audio for JACK. This setting determines which audio connections a particular instance will first attempt. The switches are mutually exclusive across JACK and ALSA, so if one checks ALSA for audio, it automatically unchecks JACK for audio. As well as from the GUI, this setting can be set (for instance 0) from the command line, both at start-up and once running. Note that any of these setting changes require a restart of *Yoshimi* to take effect.

5. Autoconnect audio. Sets *Yoshimi* to connect automatically to the JACK server, just like the `-K` command-line option does. (However, note that the command-line has no way to disable this feature it the configuration has been saved.)

5.1.4.3 Menu / Yoshimi / Settings / ALSA

A significant improvement is to the handling of ALSA audio, which is still very important for some people. Until now, *Yoshimi* has insisted on a 2-channel, 16-bit format. Tests have shown that virtually all motherboard sound chipsets will handle this, but many external ones don't.

From *Yoshimi* 1.3.6 onwards, when using ALSA audio, *Yoshimi* first tries to connect 2 channels at 32 bit depth. If that connection does not succeed, then *Yoshimi* negotiates whatever the soundcard will support. For example, a card might support only 24 bits, and 6 channels. So *Yoshimi* will fall back to 24 bit, and, due to its own limits, will use only channels 1 and 2. With external sound modules in mind, endian swaps are also implemented.

To be able to reliably use ALSA audio, one needs to set a card name, not just "Default". In a terminal window enter the following command:

```
$ cat /proc/asound/card*/id
```

The result of this command should be something like:

```
PCH
K6
```

Go to the ALSA settings tab illustrated below, and in *Alsa Audio Device* enter, for example, "hw:PCH". This ensures one will always connect to this card at startup regardless of the order this and other ones. Another benefit of using this hardware name is that ALSA will now use *Yoshimi*'s internal buffer size (256), otherwise ALSA will force *Yoshimi* to accept its default size (usually 1024).

One can also set the sample rate, but bear in mind that not all cards can use all of these. The sample rates 44100 and 48000 are almost always available. If one sets a Midi Device as well (such as a keyboard) Yoshimi will try to find and connect to this device at startup.

To find the MIDI devices available, try:

```
$ grep Client /proc/asound/seq/clients
```

The result of this command should be something like:

```
Client info
Client   0 : "System" [Kernel]
Client  14 : "Midi Through" [Kernel]
Client 128 : "TiMidity" [User]
```

It is not obvious how ALSA audio is controlled and who takes command. If one sets a specific audio destination, then *Yoshimi* makes a request. It's often a negotiation on bit depth and channel count, but *Yoshimi* nearly always gets to decide the buffer size, which is the internal buffer size. However, if the

destination is 'default' then ALSA decides on the sound card, bit depth, number of channels and the buffer size, and *Yoshimi* will set it's internal buffer size to match. On most machines this seems to be 1024.

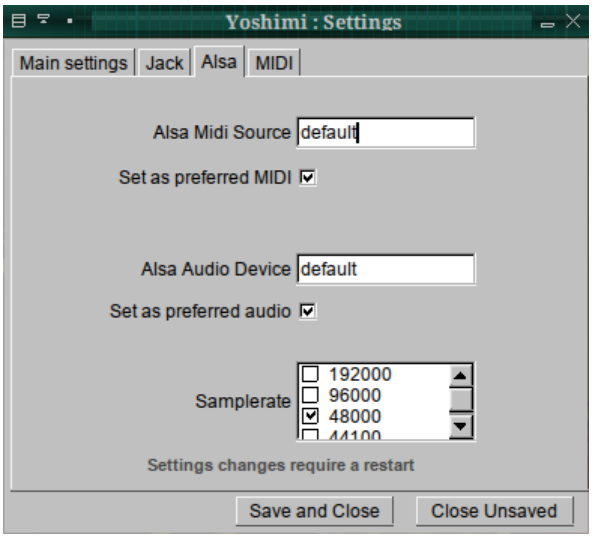

Figure 17: ALSA Settings Dialog

1. Alsa Midi Source. ALSA MIDI Source. The purpose of this setting is the same as the command line option `--alsa-midi="name"`. It is used so that *Yoshimi* can auto connect to a MIDI source such as a keyboard. For example, the one that Will has identifies itself as name = "Hua Xing". A port name, such as "128:0" (for one of the ports provided by *TiMidity*) should work as well. If the destination is "default", then ALSA decides on the sound card, bit depth, number of channels and the buffer size, and *Yoshimi* will set it's internal buffer size to match. On most machines this always seems to be 1024.

Values: `default*`

2. Set as preferred MIDI. Set as preferred MIDI for ALSA. This setting determines which MIDI connections a particular instance will first attempt. The switches are mutually exclusive across JACK and ALSA, so if one checks ALSA for MIDI, it automatically unchecks JACK for MIDI. As well as from the GUI, this setting can be set (for instance 0) from the command line, both at start-up and once running.

3. Alsa Audio Device. ALSA Audio Device. This specifies the sound card to which *Yoshimi* can connect. Normally, this will be an ALSA hardware specification such as "hw:0". ALSA audio also lets one connect to a sound card by name. For example, with a Komplete Audio KA 6 sound card, the device specification is "hw:K6". This feature is particularly useful for USB modules, as one can never be sure where they appear numerically.

Values: `default*`

4. Set as preferred audio. Set as preferred audio for ALSA. This setting determines which audio connections a particular instance will first attempt. The switches are mutually exclusive across JACK and ALSA, so if one checks ALSA for audio, it automatically unchecks JACK for audio. As well as from the GUI, this setting can be set (for instance 0) from the command line, both at start-up and once running.

5. Samplerate. Sample Rate. Sets the quality of the sound, higher is better, but it uses more CPU. One can select from a list. Note that both ALSA and JACK will support the 192000 rate, if the sound-card supports it. To find out the internal delay in milliseconds, divide the buffer-size value by the Sample Rate and multiply the result by 1000 (256 / 44100 * 1000 = 5.8 ms).

Note that, as of version 1.3.6, the **Period Size** field has been removed from the **Alsa** tab, and is replaced by the **Internal Buffer Size** field in the **Main Settings** tab. Note that any of these setting changes require a restart of *Yoshimi* to take effect.

ZynAddSubFX: if one wants a sample-rate that is not in the list, select "Custom" and change the value from the right. Default is 44100.

Values: `192000, 96000, 48000*, 44100`

5.1.4.4 Menu / Yoshimi / Settings / MIDI

The CC settings tab has been renamed the "MIDI" tab. This tab, shown in figure 18 "MIDI Preferences" on page 44, presents MIDI bank-root, bank, program change, and extended program change settings, plus some new values.

A new feature as of version 1.3.6 is that some changes to the items in this tab cause a red **Pending** button to appear. Pressing this button saves that particular change.

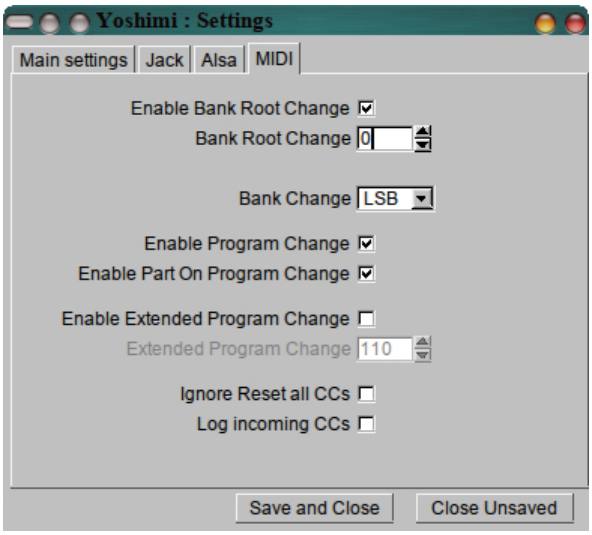

Figure 18: MIDI Preferences

The following items are provided by the MIDI settings tab:

1. **Enable Bank Root Change**
2. **Bank Root Change**
3. **Bank Change**
4. **Enable Program Change**
5. **Enable Part On Program Change**
6. **Enable Extended Program Change**
7. **Extended Program Change**
8. **Ignore Reset all CCs**
9. **Log incoming CCs**
10. **Save and Close**
11. **Close Unsaved**

The concepts of banks and roots is very useful. See section 2.3 "Concepts / Banks and Roots" on page 20. The settings in this tab affect the usage of banks and root changes controlled by MIDI messages, thereby making *Yoshimi* able to implement MIDI automation.

1. Enable Bank Root Change. Enable Bank Root Change.

Values: `Off*, On`

2. Bank Root Change.

Values: `0*, to 127`

If enabled, a new reddish button, **Pending**, appears. Once the change has been made in the scroll list, click this button to set the change. **Warning:** The **Save and Close** button will not result in the removal of the **Pending** button. This result seems counter-intuitive, but the pending button is not removed here because, at that point, it still hasn't actually been either set or abandoned. It remains available for when the user actually makes up his/her mind.

3. Bank Change. Bank Change. Defines which MIDI preferences one wants to use. Note that MIDI Controller 0 = CC0 = Bank Select MSB, and MIDI Controller 32 = CC32 = Bank Select LSB. When combined, these Bank Select messages provide

$$128 * 128 = 16384$$

banks.

But note that all a Bank Select does is selects the bank for the next Program Change event. The program doesn't change after changing a bank, until a Program Change is sent.

Bank changes can be completely disabled; some hardware synthesizers don't play nice with banks.

Values: `LSB, MSB*, Off`

4. Enable Program Change.

Values: `Off*, On`

Enables/disables MIDI program change. Program changes can be completely disabled, but some hardware synths don't play nice!

5. Enable Part On Program Change.

Values: `Off*, On`

The part is automatically enabled if the MIDI program was changed on this part.

6. Enable Extended Program Change.

Values: `Off*, On`

7. Extended Program Change. If enabled, a new reddish button, **Pending**, appears. Once the change has been made in the scroll list, click this button to set the change.

Values: `0-127, 110*`

8. Ignore Reset all CCs. Causes *Yoshimi* to ignore this message. For example, using *Yoshimi*'s fairly new CC monitor (see the next item), Will found that *Rosegarden* was sending CC 121 (reset all controllers) at the start of some song segments. Checking this option prevents unwanted resets.

Values: `Off*, On`

9. Log incoming CCs. This setting is is about the only setting that is never saved. It is there solely as an aid for when *Yoshimi* appears to ignore MIDI commands, as it tells one exactly what *Yoshimi* thinks it received.

Values: `Off*, On`

5.1.5 Menu / Yoshimi / Exit

Simply exits from *Yoshimi*. The user is prompted if unsaved changes exist, as shown in figure 19 "Yoshimi Menu, Exit" on page 46.

One can sometimes get a false parameters-changed warning if one scrolls through one of the menu type entries without actually changing it. Better safe than sorry!

Figure 19: Yoshimi Menu, Exit

5.2 Menu / Instruments

The *Yoshimi* Instruments menu lets one select instruments and work with banks of instruments. *Yoshimi* stamps instrument XML files with its own major and minor version numbers so it is possible to tell which version created the files, or whether they were created by *ZynAddSubFX*.

When opening an instrument bank one can now tell exactly which synth engines are used by each instrument. This is represented by three pale background colours:

- Red: ADDsynth
- Blue: SUBsynth
- Green: PADsynth

These new colored engine backgrounds aren't just pretty. They give real information about expected processor load, and time taken to be ready when loaded:

- *Processor Load, low to high*: PAD, SUB, then ADD.
- *Time to initialize, low to high*: SUB, ADD, PAD.

If the instruments are kits they scanned to find out if *any* member of the kit contains each engine. This scanning is duplicated in the current part, the mixer panel for the currently loaded instruments, and in the Instrument Edit window the same colors highlight the engine names when they are enabled with the check boxes.

The following sub-menus are provided, as shown in figure 20 "Yoshimi Menu, Instrument" on page 46.

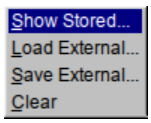

Figure 20: Yoshimi Menu, Instrument

This new version of the **Instrument** menu is somewhat different than the old version. It is actually simpler and easier to use, while still offering all of the power of the setting up of instruments in *Yoshimi*.

1. **Show Stored...**
2. **Load External...**
3. **Save External...**
4. **Clear**

5.2.1 Menu / Instrument / Show Stored...

Instruments are stored in banks. The banks (and current bank setting) are loaded/saved automatically by the program, so one doesn't have to worry about saving the banks before the program exits. On program start, the last used bank is loaded. A single bank can store up to 128 instruments. However, there is space for a number of additional instruments in the bank, the extended-program section, to allow up to 160 instruments in a bank.

When the **Show Stored...** button is selected, a dialog comes up that shows all of the instruments present in the currently-selected bank.

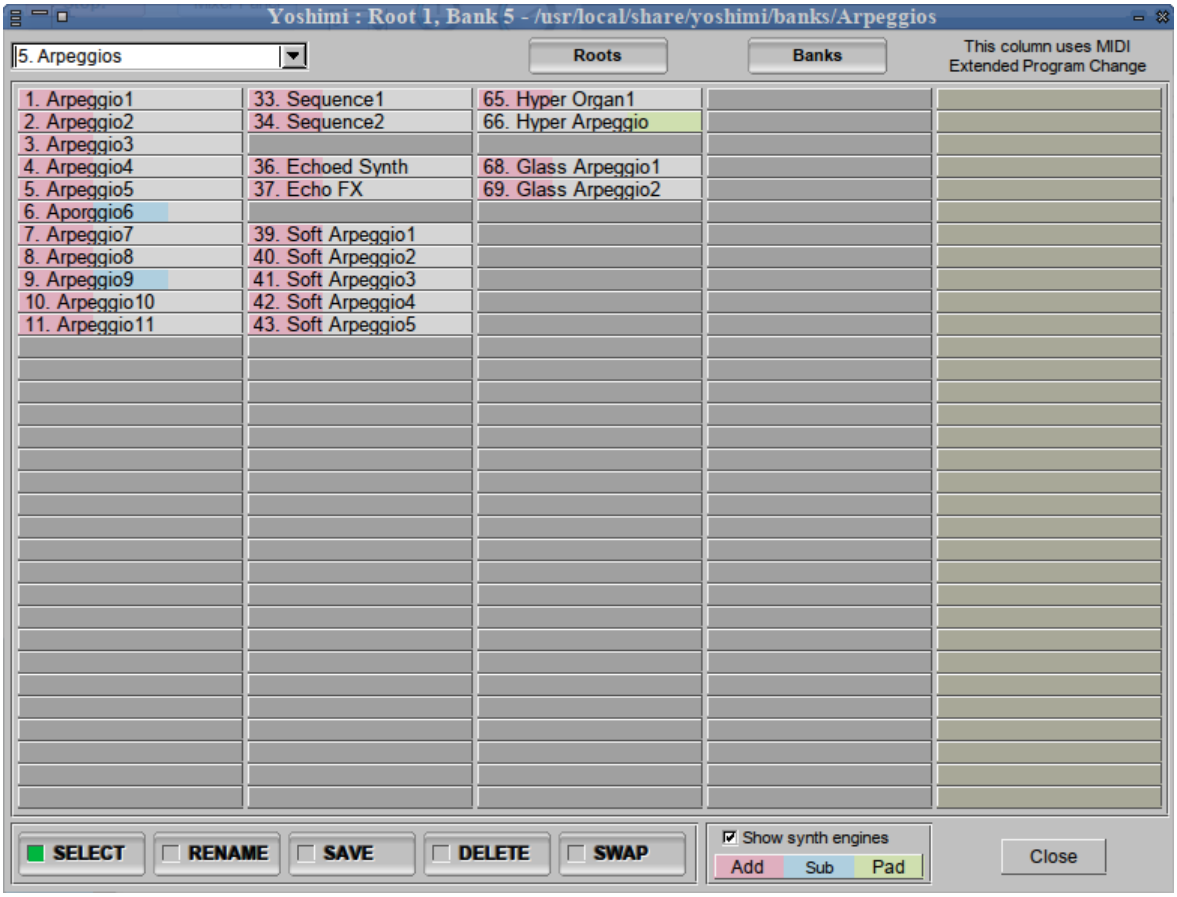

Figure 21: Instruments Stored in Current Banks

As figure 21 "Show Stored Instruments" on page 47, shows, this is a very complex dialog with a lot of options. The figure shows a default setup, with the first bank of instruments, **5. Arpeggios**, listed. If one drops this list down (shown later), one also observes that the banks are numbered in increments of 5, to make it easier for a user to insert his or her own bank(s) of instruments.

The default set of banks are spaced 5 apart only because thats the best-fit for the current number of banks. If we add more banks in future versions of *Yoshimi*, then a *clean* install is likely to have different spacing, but for an existing setup, the new entries will just be slotted in where they will fit.

Also, if one deletes banks or instruments by some external means, the next time *Yoshimi* starts, it will notice their absence and quietly remove their entries.

Note how *Yoshimi* now shows the color codings for the synth-sections used in each instrument: red for ADDsynth, blue for SUBsynth, and green for PADsynth.

Also note how the numbers at the beginning of the filenames are used as an "instrument" or "program" number. These numbers can be used in MIDI Program Change commands.

All of the instrument files (such as `0001-Arpeggio1.xiz`) with filenames starting with 4-digit numbers will be shown in the corresponding slot number. Those instrument files without numbers (or larger numbers?) will start with numbers at 129 or above ("Extended Program Change"). One could give them numbers by renaming them outside of *Yoshimi*, then reloading the bank. One can also fix unnumbered ones simply by loading them, then resaving them to the same slot. It's then probably best to swap them into the main set, if there is space.

Note that MIDI CC (see section 5.1.4.4 "Menu / Yoshimi / Settings / MIDI" on page 44) can be set to access voices from 129 to 160. All the Bank controls in the **MIDI** settings tab take immediate effect when set. Bank and program changes can be completely disabled in the settings tab; some hardware synths don't play nice with it.

Learning how to use the Instruments dialog is an important way to make instruments easier to manage, and so this will be a long discussion.

Here is a list of the user-interface items in the instruments/banks dialog:

1. **Bank Names**
2. **Roots**
3. **Banks**
4. **Instrument and Bank Matrix**
5. **SELECT**
6. **RENAME**
7. **SAVE**
8. **DELETE**
9. **SWAP**
10. **Show synth engines** (was **Show PADsynth status**)
11. **Close**

1. Bank Names. Instruments Bank Name. This item is a drop-down list of the available instrument banks in the currently-selected **root** directory. Basically, each bank is a directory name, with a number prepended. The banks are found under the current root, which is a also a directory name, and is the name of the parent directory of a set of banks. Here is the Bank Names drop-down list for the default setup, which has the default banks provided by the basic *Yoshimi* installation.

Figure 22: A Sample Bank List

And here is the directory listing associated with it, in the order produced by the UNIX/Linux `ls -1` (list single-column) command (shown in two columns to save space):

```
Arpeggios           Pads
Bass                Plucked
Brass               Reed_and_Wind
chip                Rhodes
Choir_and_Voice     Splited
Drums               Strings
Dual                Synth
Fantasy             SynthPiano
Guitar              The_Mysterious_Bank
Misc                Will_Godfrey_Collection
Noises              Will_Godfrey_Companion
Organ
```

The directories (banks) shown above come from the default **root** when *Yoshimi* and its data files are installed: `/usr/share/yoshimi/banks`. If one installed *Yoshimi* by building the source code, then this directory will be `/usr/local/share/yoshimi/banks`.

The first thing to note is that there are only 128 *Yoshimi* banks supported in a *Yoshimi* root. If the list of banks takes up about half of the available slots, it might be time to move some of those banks to a new root directory.

The numbers in the drop-down list are generated by *Yoshimi* the first time it sees a new root path or a new bank within the root path. Once set, these numbers will never change unless one actually moves them around (using the **SWAP** button).

The bank number is also the MIDI ID for the bank; one can be sure that it will always be there for bank changes, no matter how many banks are added later. *Yoshimi* always lists the banks in ID order, not alphabetical order, so one can group them sensibly and permanently. However, at first-time creation *Yoshimi* sets the IDs in alphabetical order and tries to space them evenly over the range to provide some wiggle room.

Selecting one of the items in this drop-down list selects the bank and loads it into the Banks dialog.

Right- or left-clicking on a bank in the drop-down list causes the instrument list of the previous bank to be replaced by the instrument list of the newly-selected bank.

2. Roots. Instruments Roots Button. Shows a list of directories that can serve as "root" directories. The "Bank Root Paths" dialog discussed in section 5.3.5 "Menu / Patch Sets / Patch Bank Roots" on page 58in figure 28 "Show Patch Banks" on page 56, shows the system root (e.g. `/usr/share/yoshimi/banks`) and a user's home location for his/her banks and roots.

3. Banks. Banks Button. This item brings up a Banks dialog showing all of the banks present in the current root. It is an alternative to using the **Bank Names** drop-down list to select a bank. It is also a way to reorganize and renumber the banks without using the Linux console or a file-explorer application to do so.

4. Instrument and Bank Matrix. Instruments Bank Matrix. Shows the instruments that are in the currently selected bank (directory).

The next few items are selector buttons that determine what happens when one clicks on an instrument name.

5. SELECT. Instruments SELECT. When this button is selected, then clicking on an instrument selects that instrument as the instrument for the current Part active in the main window. In the main window of *Yoshimi*, that instrument name will appear in the currently-selected **Part**. If *Yoshimi* is writing to a console window then each part, when clicked, will be shown:

```
yoshimi> Loaded 64 "Hyper Organ1" to Part 0
Loaded 65 "Hyper Arpeggio" to Part 0
Loaded 10 "Arpeggio11" to Part 0
Loaded 41 "Soft Arpeggio4" to Part 0
Loaded 67 "Glass Arpeggio1" to Part 0
```

(Remember that "Part 0" in the console is actually "Part 1" in the main window.)

6. RENAME. Instruments RENAME. When this button is selected, then clicking on a bank brings up a small dialog to rename the clicked-on bank. However, one will see the following warning message if trying to rename a file that is in a directory not modifiable by normal users:

```
! Could not rename instrument 39 to Soft Arpeggio5 [Close]
```

Note that, as soon as this operation is done, the auto-selector (green check-box) moves back to the **SELECT** button.

7. SAVE. Instruments SAVE. When this button is selected, then clicking on a bank saves the instruments as currently configured. A prompt like the following will appear:

```
? Overwrite the slot no. 43 ?  [No/Yes]
```

However, if one answers yes, and the instrument is in a non-modifiable directory, then one will see the following error message:

```
! Could not save to this location [Close]
```

8. DELETE. Instruments DELETE. Selecting this button and clicking an empty bank entry does nothing. Selecting this button and clicking an existing bank entry brings up a small dialog asking one if this bank is really to be deleted.

```
? Clear the slot no. 68?  [No/Yes]
```

However, if one answers yes, and the instrument is in a non-modifiable directory, then one will see the following error message:

```
! Could not clear this location  [Close]
```

9. SWAP. Instruments SWAP. Selecting this button, then selecting one instrument, and then another, swaps the numbering and postion of the selected instruments. However, one might also experience the following warning message:

```
! Could not swap these locations  [Close]
```

Note that all of the above error messages are also shown in the console, if it is where *Yoshimi* is running. For example:

```
40 Failed to remove /usr/local/share/yoshimi/banks/Arpeggios/0041-Soft
Arpeggio3.xiz Permission denied
```

10. Show synth engines. If enabled, then the usage of each of the *Yoshimi* synthesis engines is indicated by color coding, as shown in the figure above.

11. Close. Closes the window.

5.2.2 Menu / Instrument / Load External...

This menu entry simply brings up a file dialog, allowing the user to navigate to an arbitrary directory, and then to a solitary instrument file (`*.xiz`), and load it into the current Part.

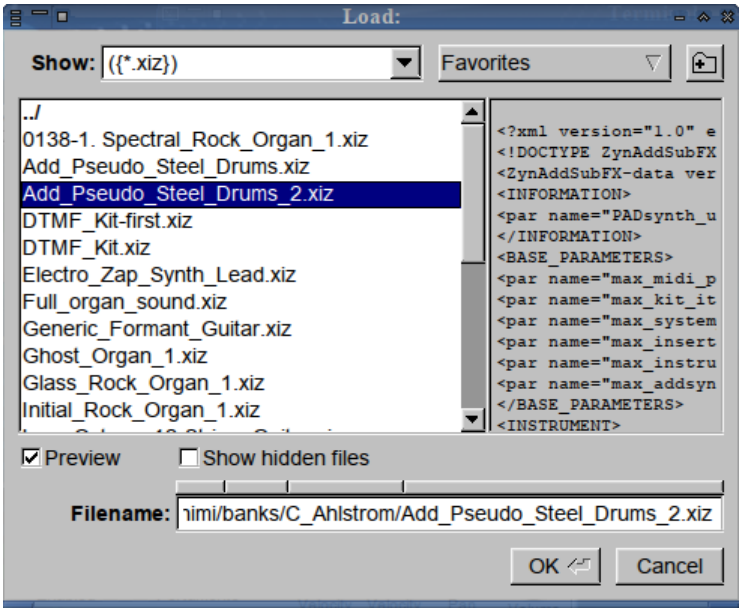

Figure 23: Instruments, Load External

These "xiz" files are normally found in a `banks` directory, but this operation allows access to instruments that are not located in a bank.

This dialog has a number of user-interface elements to discuss:

1. **Show**
2. **Favorites**
3. **Create a new diretory**
4. **Instrument List**
5. **XML Preview**
6. **Preview**
7. **Show hidden files**
8. **Directory Bar**
9. **Filename**
10. **OK**
11. **Cancel**

These elements are used in a number of different places in *Yoshimi*. Therefore, we will explain them all once, here.

1. Show. Show types of files. This item shows a file filter for selecting instrument files. The types of filters are as follows (screen shot not available):

1. **({*.xiz})** (compressed XML files)
2. **All Files (*)**
3. **Custom Filter**

2. Favorites. Favorite directories. Provides a list of options and favorite directories in which to find instrument files.

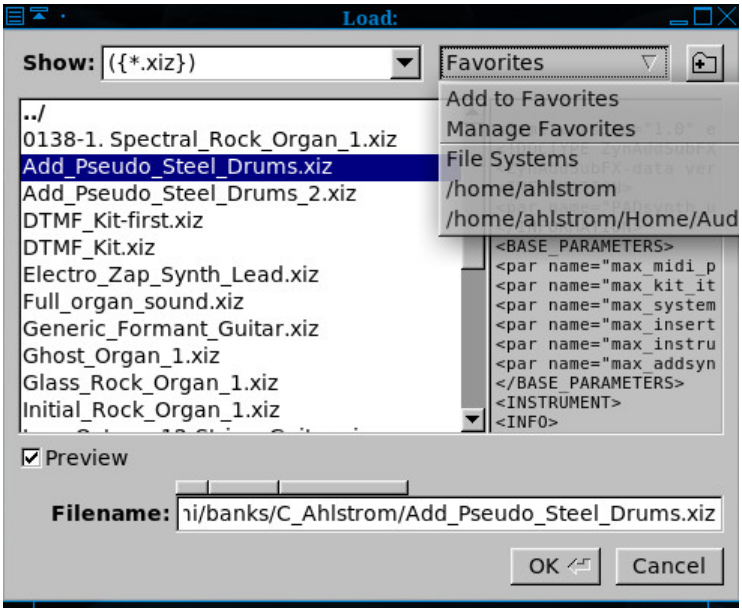

Figure 24: Manage Favorites Dialog

1. **Add to Favorites**
2. **Manage Favorites**
3. **File Systems**
4. **(Additional favorite directories)**

Add to Favorites simply adds the currently selected directory shown in the instrument list to the list of favorites.

To add favorites in the file dialog, navigate to the desired directory. Then click **Favorites**, and select **Add to Favorites**.

Once one has a number of favorites set up, there is a **Manage Favorites** that can be used. For example, if one needs to get rid of a directory, one can use the **Manage Favorites** dialog, shown in figure 25 "Favorites Drop-down" on page 53below, to do that.

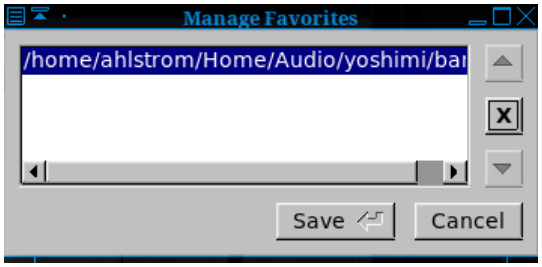

Figure 25: Favorites Drop-down

File Systems Provides a list of all file systems starting at root ("/"). This list can be pretty confusing, with a lot of entries. But note that one navigates to ("/"), and from there to `/usr/share/yoshimi/banks`

to get easy access to all the instruments that are preinstalled with *Yoshimi*. Generally, one will want to use only **Add to Favorites** and **Manage Favorites**.

3. Create Directory. Creates a New Directory. This little symbol brings up a small "New Directory?" dialog (not shown here, it is very simple and stock) into which one can type a directory name to be added to the current directory of the instrument list.

4. Instrument List. Provides a list of the instrument files available in the current directory. Also shown are sub-directories (if available) that might contain more instruments, and a ("`../`") entry to navigate to the parent directory.

5. Preview. If one thinks the preview feature is not useful, uncheck this check-box. so that one doesn't see the preview window. As a bonus, one can see more of a long instrument file-name.

6. Preview pane. XML Preview. This box can show the beginning of the XML data of an instrument file. **Bug:** The XML preview feature shows the XML only if the XML is not compressed.

7. Show hidden files. Shows file that are hidden. Not sure how useful this feature is; who would hide a *Yoshimi* instrument file?

8. Directory Bar. Provides an alternate way to move up through the directory structure. Click on each of the small bevelled rectangles to move around in the directory hierarchy.

9. Filename. File Name. Provides the full path specification for the instrument file.

10. OK/Cancel. We don't really need to discuss the **OK** and **Cancel** buttons, do we? OK, we'll cancel that discussion.

5.2.3 Menu / Instrument / Save External...

This menu entry simply brings up a file dialog, allowing the user to navigate to an arbitrary directory, and then save the current Part to a solitary instrument file (`*.xiz`).

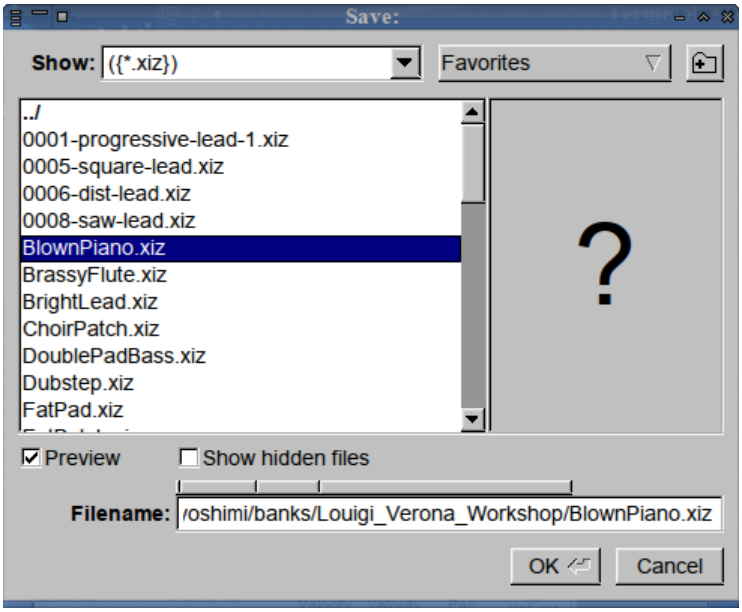

Figure 26: Instruments, Save External

This dialog is very similar to the **Load External** dialog. Note the large question mark in the XML preview, indicating a compressed XML file.

Instrument saves save only what is essential to the instrument, and not the part it may be sitting in.

5.2.4 Menu / Instrument / Clear

This menu entry simply clears the instrument that is loaded into the current Part. This converts the instrument to a *Simple Sound* patch. This menu entry brings up a prompt to clear the parameters of the instrument that is currently loaded in the current part.

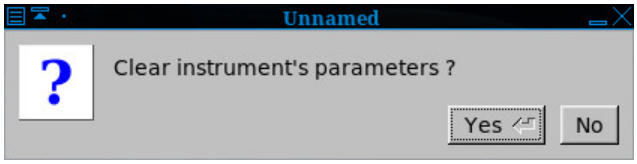

Figure 27: Clear Instrument Dialog

5.3 Menu / Patch Sets

This new menu entry is part of the very nice reorganization and simplification of the handling of roots and banks in the new *Yoshimi*. The **Patch Sets** menu replaces the old **Parameters** menu. Do you like the new name? The patch set saves all of the settings, including effects and instruments.

Yoshimi stamps its parameter XML files with its own major and minor version numbers so it is possible to tell which version created the files, or whether they were created by *ZynAddSubFX*.

The main dialog is somewhat similar in layout and function to the dialog shown in figure 21 "Show Stored Instruments" on page 47, for managing instruments in a selected bank.

5.3.1 Menu / Patch Sets / Show Patch Banks...

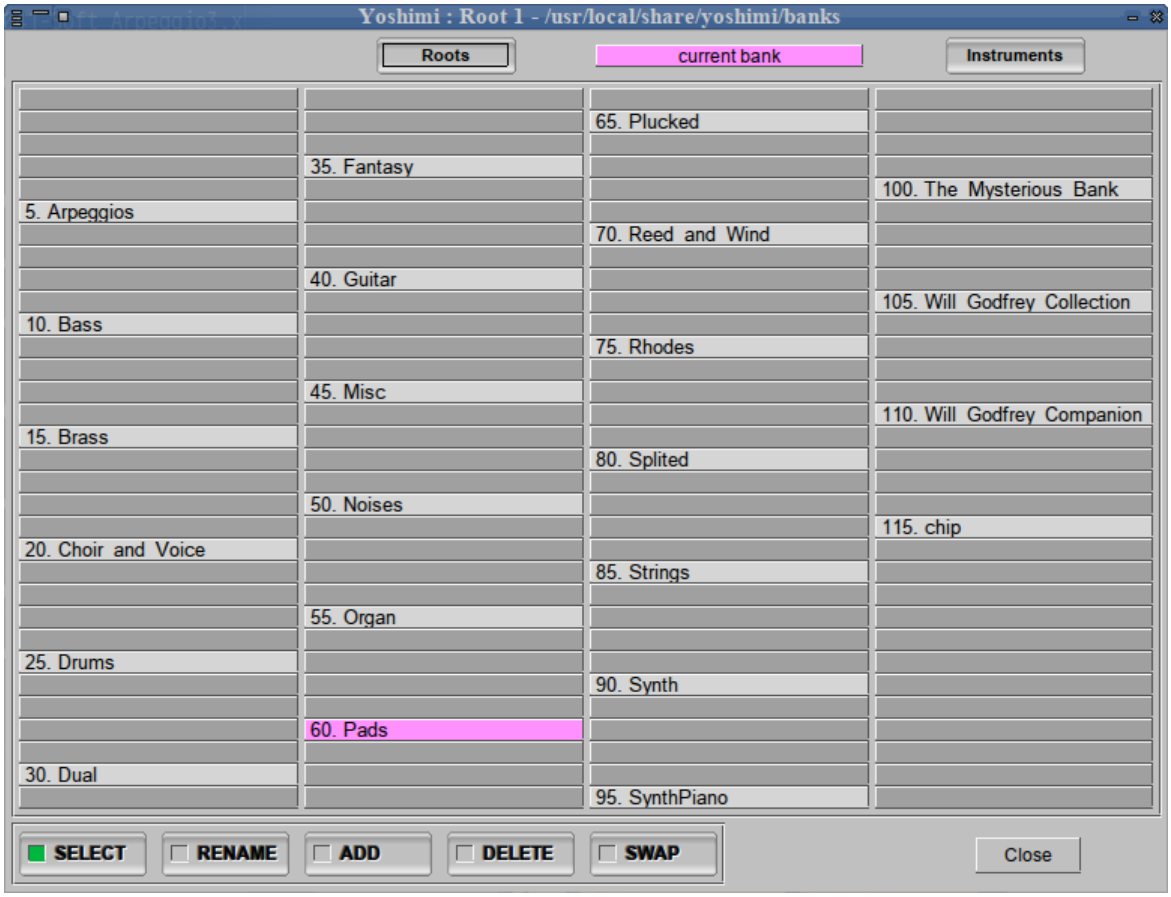

Figure 28: Show Patch Banks

Here is a list of the user-interface items in the patch-banks dialog:

1. **Roots**
2. **current bank**
3. **Instruments**
4. **Bank Matrix**
5. **SELECT**
6. **RENAME**
7. **SAVE**
8. **DELETE**
9. **SWAP**
10. **Close**

1. Roots. Show Patch Banks, Root Directories. To add a bank root path, delete a bank root path, or manage bank root path, press this button. The result is somewhat similar to a file dialog, and is described in detail in section 5.3.5 "Menu / Patch Sets / Patch Bank Roots" on page 58, later in this sub-chapter.

2. current bank. This item is highlighted in pink, and the bank that is actually the current bank is also highlighted in pink. There is no action associated with this user-interface element; it merely indicates the currently-selected bank.

3. Instrument. This button brings up an instruments window similar to the one shown in figure 21 "Show Stored Instruments" on page 47, which shows the instruments collected in the currently-selected bank. Clicking on a bank in the dialog also brings up the instruments window.

4. Bank Matrix. This view shows all of the banks available in the current root. Left-clicking on a bank in the dialog brings up the Instruments window for that bank. Right-clicking on a bank in the dialog brings up the Instruments window for that bank, but also closes the banks window, to reduce clutter.

The rest of the buttons **SELECT**, **RENAME**, **SAVE**, **DELETE**, and **SWAP** behave similarly to the same buttons in the Instruments window, as described in the section 5.2.1 "Menu / Instrument / Show Stored.." on page 47discussion.

5.3.2 Menu / Patch Sets / Load External...

This menu entry simply brings up a file dialog, allowing the user to navigate to an arbitrary directory, and then to a solitary instrument file (`*.xmz`), and load it into the current set of parts.

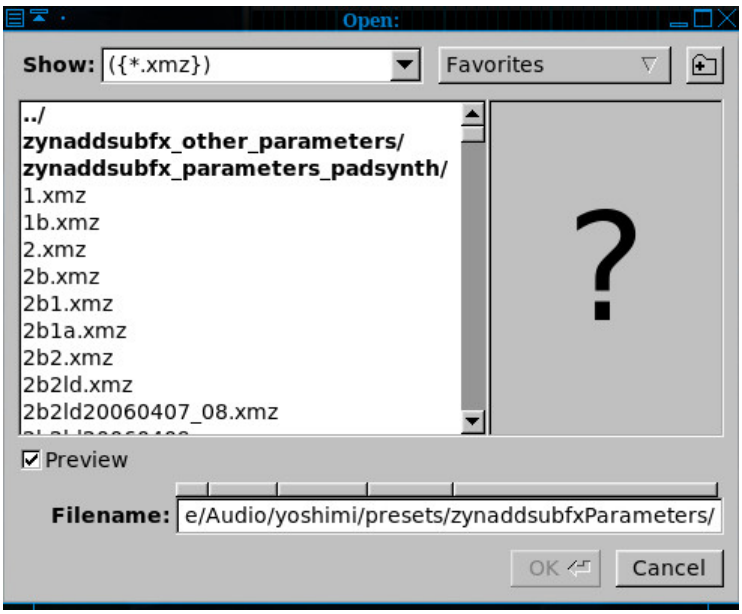

Figure 29: Load Patch Set

These "xmz" files are normally found in a `presets` directory, but this operation allows access to banks that are not located in a particular root.

When an "xmz" file is loaded, all of the instruments it contains are loaded sequentially into the Parts. Thus, a number of instruments are loaded at once. So, a patch set is a list of instruments that are related by being necessary for a given tune, rather than by being located in a particular bank.

In parameter sets, *Yoshimi* will save named-but-disabled patches. Currently, *ZynAddSubFX* does not, so be aware when transferring data between the two synthesizers.

5.3.3 Menu / Patch Sets / Save External...

This menu entry simply brings up a file dialog, allowing the user to navigate to an arbitrary directory, and then save the current Part to a solitary instrument file (`*.xiz`).

In parameter sets, *Yoshimi* will save named-but-disabled patches. Currently, *ZynAddSubFX* does not, so be aware when transferring data between the two synthesizers.

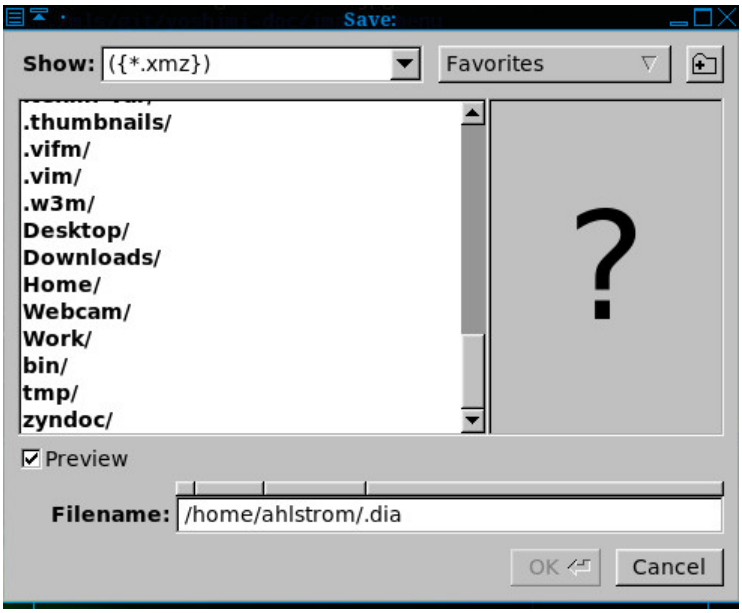

Figure 30: Save Patch Set

Patch set saves include everything that is not part of the main configuration, and so saved patch sets includes Master Volume and Detune, Part destinations, Humanise, and more.

If nothing has changed, then the following dialog is shown.

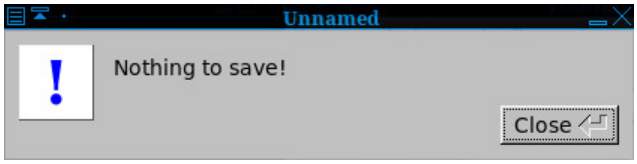

Figure 31: Patch Set, Nothing to Save

5.3.4 Menu / Patch Sets / Recent Sets

This menu entry brings up a dialog box with a list of the recent patch sets that have been loaded. This item makes it easy to move around one's frequently-used banks.

5.3.5 Menu / Patch Sets / Patch Bank Roots

Yoshimi (as installed by Debian Linux) provides a default bank at `/usr/share/yoshimi/banks`. To add one's own directory, click on the **Roots** button. It brings up the following dialog.

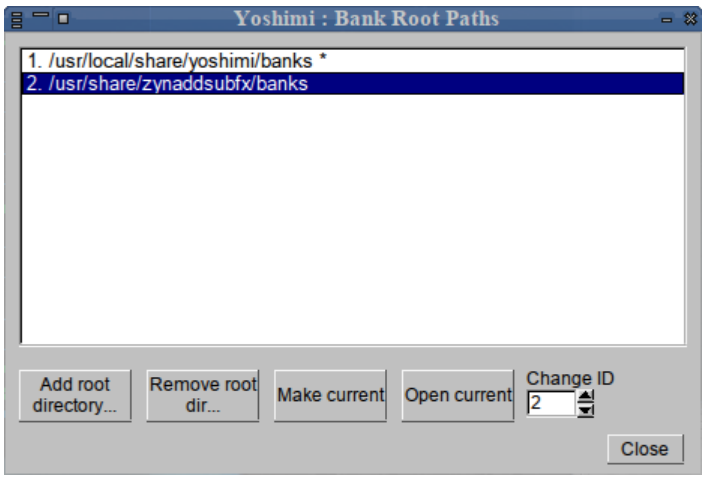

Figure 32: Bank Root Paths

This dialog has a number of buttons, some of which will be disabled if no directory in the list is selected.

Then click on the "Add root directory..." button. In the file dialog that appear, one can use the **Create Directory** button to make a new directory, if desired:

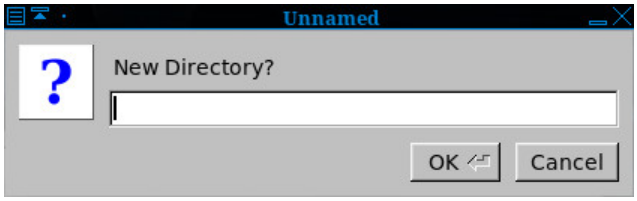

Figure 33: New Root Directory?

Otherwise, once can add an existing directory to the list.

1. Add root directory.... Bank Root Paths, Add Root Directory.

Once selected, one will see that `/usr/share/yoshimi/banks` or `/usr/local/share/yoshimi/banks` is marked with an asterisk. One can select the new root directory via the file dialog that appears, and then make it the current root by clicking the **Make current** button. Then the Banks dialog will show all the banks in that directory, one bank per subdirectory (each subdirectory "is" a bank).

2. Remove root directory.... Bank Root Paths, Remove Root Directory. If a path is selected, then this button is active, and can be used to delete the selected path from the "root paths" list.

3. Make current. Bank Root Paths, Make Current. This button marks the currently-selected path as the "current root" path.

4. Open current. Bank Root Paths, Open Current. This button opens the current root path. (Does this work?)

5. Change ID. Bank Root Paths, Change ID. We need to know more about how this ID can be used. Is it a way to make the path selectable via an extended MIDI control, or some other automation method?

Values: `0* to 127`

5.4 Menu / Paths

This menu entry provides a more direct way to set up the Bank Root and the Presets directories. It contains the following items:

1. **Bank Root Dirs...**
2. **Preset Dirs...**

5.4.0.1 Bank Root Dirs...

The Paths Bank Root Dirs dialog is described in section 5.3.5 "Menu / Patch Sets / Patch Bank Roots" on page 58, which shows figure 32 "Bank Root Paths" on page 59, and describes this dialog in full.

5.4.0.2 Preset Dirs...

The *Yoshimi* preset directories are the locations where presets can be found. When first installed, the system preset directory is one of the following, depending on whether *Yoshimi* was installed via a package manager or via source code:

```
/usr/share/yoshimi/presets
/usr/local/share/yoshimi/presets
```

The user can provide additional directories for the presets, up to a limit of 128 directories (the same limit as for roots and banks). These directories are useful for containing copies of the system presets that one can modify safely, and for providing custom presets designed by the user.

The following items are provided by the preset directory settings:

1. **Preset list**
2. **Add preset directory...**
3. **Remove preset directory...**
4. **Make default**
5. **Save and Close**
6. **Close Unsaved**

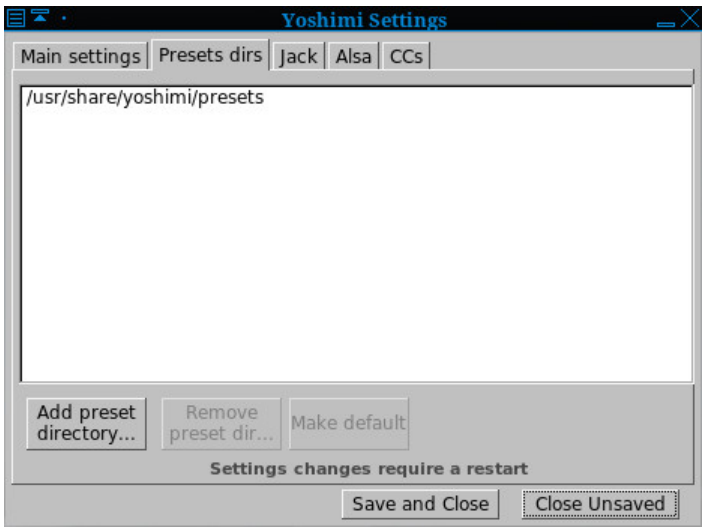

Figure 34: Yoshimi Preset Dirs Dialog

1. Preset list. This interface element contains a list of preset directories. By default, the only directory present is the installed preset directory. For example, `/usr/share/yoshimi/presets`.

Tip: If there is no directory in this dialog, then one must add one, otherwise there is no place to store the presets. So make this one of the first items specified when first running *Yoshimi*!

Another example would be this project; let YOSHIMI-DOC be the directory where this project is stored. Then one can add `YOSHIMI-DOC/config/yoshimi/presets` to this list, using the button described next.

2. Add preset directory.... Use this button and dialog to add a preset directory to the list, for easy access.

Press the **Add preset directory...** button, revealing the following dialog.

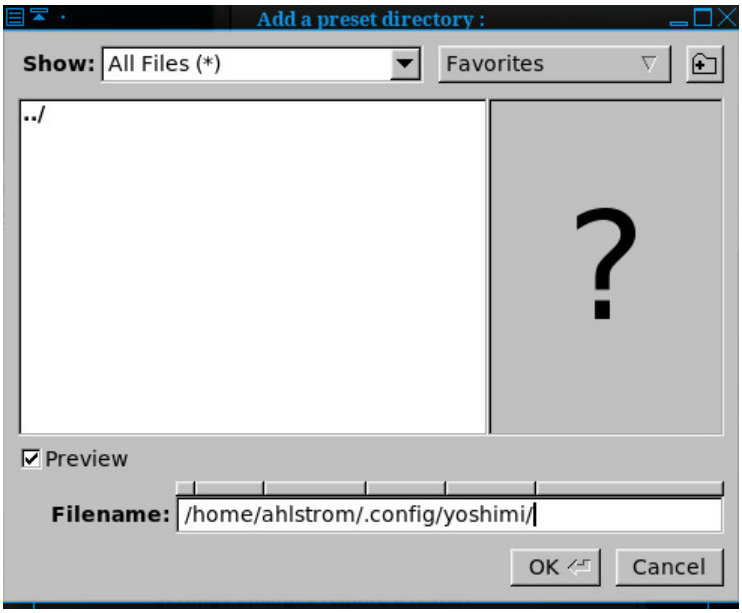

Figure 35: Add a Preset Directory

Navigate to the desired directory, select it, and press the **Ok** button. (There is no need to press the **Save and Close** button; the directory is added as soon as *OK* is clicked. However, one tends to want to click it anyway, to be sure.) *Important*: Restart *Yoshimi* to use the preset directory.

3. Remove preset directory.... Select one of the preset directories in the preset list, then press this button to remove the preset directory from the list of preset directories. It is removed immediately, with no need to confirm the deletion, click an OK button, or click a Save button.

4. Make default presets. Make Default Presets Directory. Select one of the preset directories in the preset list, then press this button to make the preset directory the default preset directory. It should be a directory for which one has write permissions. By default, it is `~/.config/yoshimi/presets`.

5.5 Menu / Scales

Yoshimi is a microtonal synthesizer, and is capable of a wide range of microtonal scales.

At present, we're not too experienced with this feature.

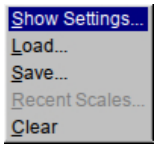

Figure 36: Yoshimi Menu, Scales

1. **Show Settings...**
2. **Load...**
3. **Save...**
4. **Recent Scales...**
5. **Clear**

5.5.1 Menu / Scales / Show Settings

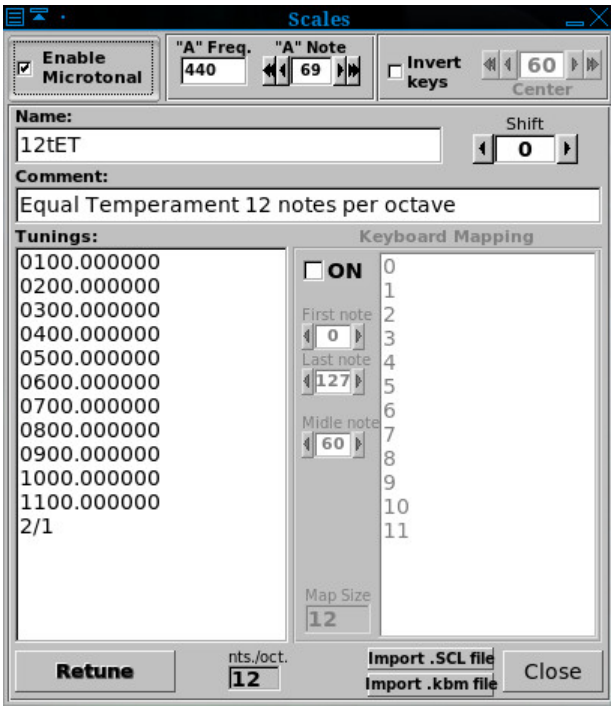

Figure 37: Yoshimi Menu, Scales Settings

5.5.1.1 Scales Basic Settings

This item controls the micro-tonal capabilities of *Yoshimi* and some other settings related to tuning. The last entry in the tunings list represents one octave. All other notes are deduced from these settings.

1. Microtonal. Enable Microtonal Scales. When disabled, the synthesizer will use equal-temperament, 12 notes per octave. Otherwise, one can input any scale one desires.

Values: `Off*, On`

2. "A" Freq. Frequency of the "A" Note. Sets the frequency of the "A" key. The standard is 440.0 Hz.

Values: `440*`

3. "A" Note. Sets the MIDI Value of the "A" Note.

Values: `0 to 127, 69*`

4. Invert Keys. Allows the keys to be inverted, so that higher-valued keys play lower notes.

Values: `Off*, On`

5. Center. Center for Inverted Keys. This is the center where the notes frequencies are turned upside-down if **Invert keys** is enabled. If the center is 60, the note 59 will become 61, 58 will become 62, 61 will become 59, and so on.

Values: `0 to 127, 60*`

6. Name. Name of the Mapping. For example, the default mapping is called "12tET".

7. Shift. Key Shift. Shift the scale. If the scale is tuned to A, one can easily tune it to another key.

Values: `-63 to 64, 0*`

8. Comment. Comment for Key Mapping. Provides a comment or a description of the scale. By default, this is "Equal Temperament 12 notes per octave".

9. Tunings. Tunings. Here one can input a scale by entering all the tunings for one octave. One can enter the tunings in two ways:

1. As the number of cents (1200 cents=1 octave) as a float number like "100.0", "123.234"
2. As a proportion like "2/1" which represents one octave, "3/2" a perfect fifth, "5734/6561". "2/1" is equal to "1200.0" cents.

The default is a series of values: `0100.0, 0200.0, ..., 1100.0, 2/1`.

10. Retune. Retune button. This button retunes the synthesizer according to the settings of the **Tunings** and **Keyboard Mapping** lists. All other changes operate immediately.

11. nts./oct. Notes Per Octave.

Values: `12*` (range not yet known)

12. Import .SCL file. Import Scala files. Scala is a powerful application for experimentation with musical tunings (intonation scales, micro-tonal,...etc.). From its home page [13], one can download more than 2800 scales which one can import directly into *Yoshimi*. Note that the zip file *must* be unzipped with the `-aa` ("autoconvert") option. However, we have converted it to a a much smaller tar file (it crams 18 Mb of files into an sub-500 Kb file), which can be untarred directly into one's configuration directory to create a `~/.config/yoshimi/scales` directory chock full of scales.

```
$ cd ~/.config/yoshimi/
$ tar xf yoshimi-scales.tar.xz
```

Note that a Scala file cannot be loaded directly. It must be imported.

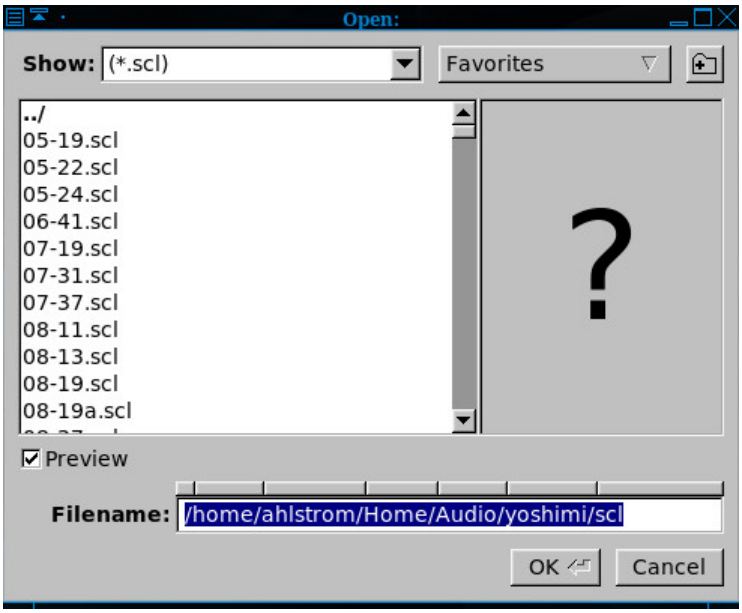

Figure 38: Yoshimi Menu, Scales, Import File

13. Import .scl file. This item is a standard file dialog for reading a `*.scl` file.

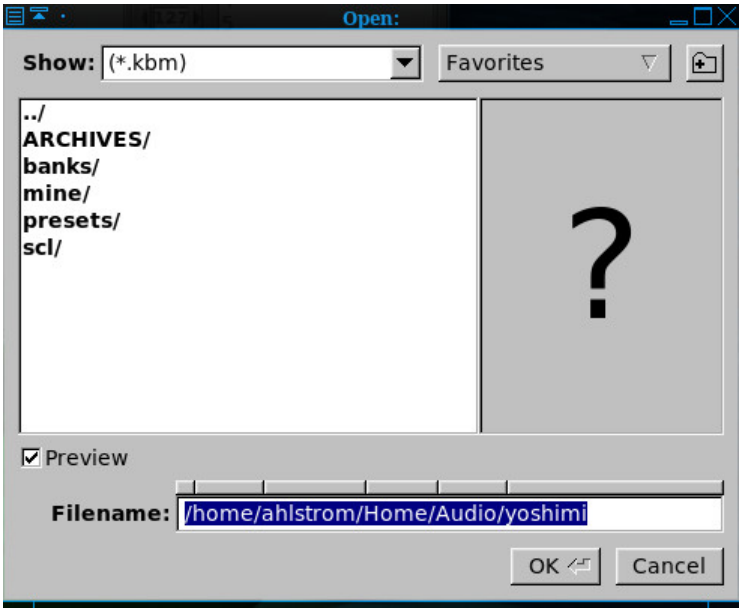

Figure 39: Yoshimi Menu, Scales, Import Keyboard Map

14. Import .kbm file. This item is a standard file dialog for reading a `*.kbm` file.

15. Close, Scales Dialog.

The items related to the **Keyboard Mapping** are discussed separately in the next section.

5.5.1.2 Keyboard Mapping

One can set the MIDI keyboard mapping to scale-degree mapping. This is used if the scale has more or less than 12 notes/octave. One can enable the mapping by pressing the **ON** check-box.

1. **ON**
2. **First Note**
3. **Last Note**
4. **Midle Note**
5. **Map**
6. **Map Size**

1. ON.

Values: `Off*, On`

2. First Note. First MIDI Note Number. Keys below this value are ignored.

Values: `0* to 127`

3. Last Note. Last MIDI Note Number. Keys above this value are ignored.

Values: `0 to 127*`

4. Middle Note. Middle note where scale-degree 0 is mapped to; the middle note represents the note where the formal octave starts. Note the misspelling of "middle".

Values: `0 to 127*`

5. Map. Scales map. This is the input field where the mappings are entered. The numbers represent the order (degree) entered on **Tunings Input** field, with the first value being 0. This number must be less than the number of notes per octave (since the values start at 0). If one doesn't want a key to be mapped, one enters an "x" instead of a number.

Values: `0 to 11`

6. Map Size. Provides the size of the scale-map.

Values: `12`

In the current version of *Yoshimi*, up to 25 recently used scales are now stored in the new history file (`yoshimi.history`), and can be quickly reinstalled with a mini-browser in exactly the same way as patch sets.

5.5.2 Menu / Scales / Load

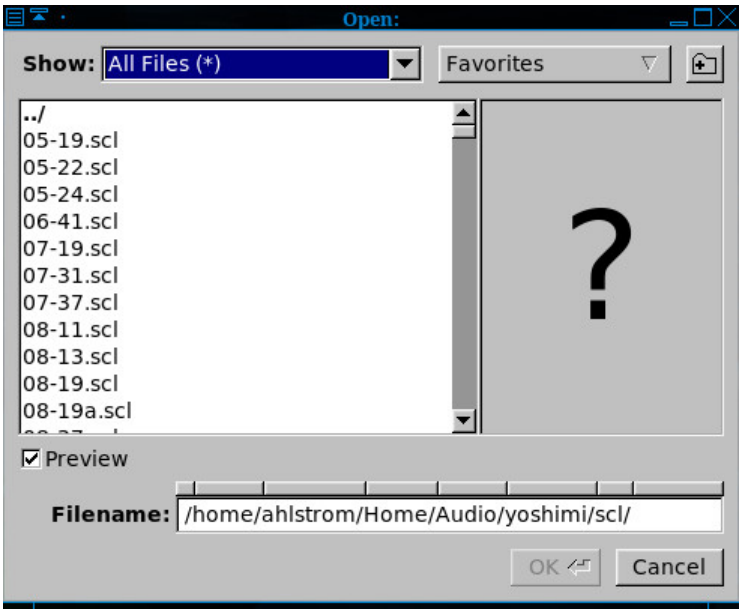

Figure 40: Yoshimi Menu, Open Scales

If the format of the scales file is not correct, then the following prompt will appear.

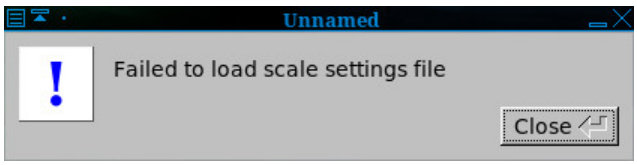

Figure 41: Yoshimi Menu, Failed to Load Scales

5.5.3 Menu / Scales / Save

This dialog opens a stock file-dialog to allow the saving of `*.xsz` files. If one has imported a scale from an `*.scl` file, and one wants direct access to it from the **Scales / Recent Scales** menu, one must first

save the imported file as an `*.xsz` files.

5.5.4 Menu / Scales / Recent Scales...

Once some scale file has been loaded (or imported and saved), then it becomes available in this list, for more convenient access to it.

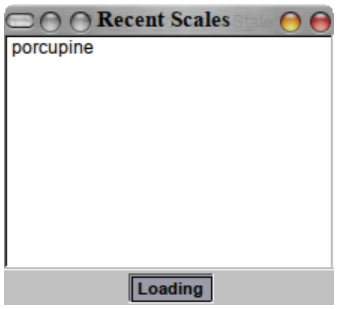

Figure 42: Yoshimi Menu, Recent Scales

5.5.5 Menu / Scales / Clear

This menu entry simply resets the *Yoshimi* scale back to it default, the twelve-tone equally-tempered scale.

5.6 Menu / State

Yoshimi state is saved in files with the extension `.state`. These files are also XML files.

Yoshimi "state" will include the system settings, as well as all patches. Some of these settings (such as Oscillator Size) can only be realised on a reload if loading via the command line at startup. However, the ones that can't be dynamically changed will be set, and if the configuration is then saved, will be set on the next load – this is not ideal. We are working on it, but don't expect improvements soon!

1. **Load**
2. **Save**
3. **Recent States...**

As the following figures show, state files are normally stored in the user's `.config/yoshimi/yoshimi.state` file.

1. State Load. Provides a way to load a previously-saved *Yoshimi* state file.

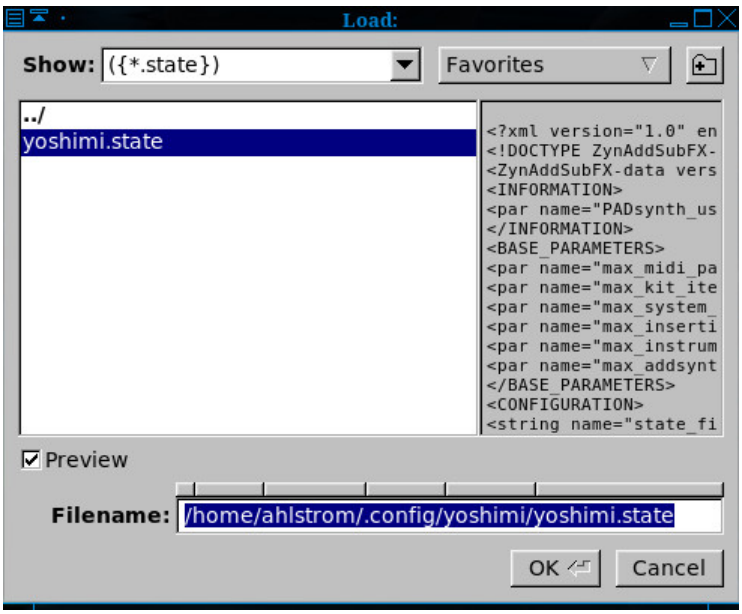

Figure 43: Yoshimi Menu, State Load

This item is a standard *Yoshimi* file dialog.

2. State Save. Provides a way to save a new or modified *Yoshimi* state file.

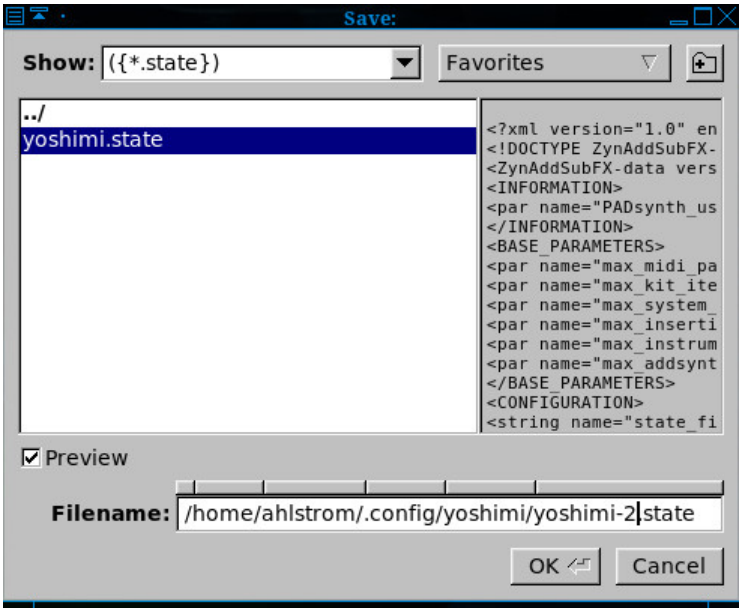

Figure 44: Yoshimi Menu, State Save

This item is a standard *Yoshimi* file dialog.

3. Recent States. This item brings up a list of states to select. The Recent States dialog will not come up if there are no states that have yet been managed.

6 Stock Settings Elements

This section collects all of the setting values and small user-interface items that one will find for audio parameters in the *Yoshimi* GUI. Sometimes the labels and tool-tips in the application are a bit too brief to understand. One will find their full meanings, their tricks, and usage notes in this section.

This section also covers the sub-panels that provide the settings. Many of these sub-panels are used in many places in *Yoshimi*, not only as user-interface elements, but as presets that can be saved and load. By describing the deep details of these sub-panels here, we can refer to them when describing how to set up specific sounds in *Yoshimi*.

Much of this material comes from http://sourceforge.net/zynaddsubfx/Doc and has been reorganized, and, it is to be hoped, expanded.

6.1 Settings Features

This section notes some minor interface and synthesizer features that may be seen thoughout *Yoshimi*.

6.1.1 Mouse Features

New: In the AddSynth dialogs and other dialogs with nested windows, if one right-clicks on a button that opens a child window, the current window is closed. Conversely, if one right-clicks on the **Close** button of a child window, it will re-open the parent. This feature is especially useful when going up and down the AddSynth stack, and avoids having a screenful of intermediate windows. Note that this feature does *not* apply to the main window.

6.1.2 Title Bars

The title bars of all editing windows display both the part number and the current name of the instrument one are working on. In the **ADDsynth Oscillator Editor**, one also sees the voice number of the oscillator one is editing. Some title bars now also include the kit entry number if that part has a kit enabled; this feature needs more work to get all of them.

6.1.3 Color Coding

A GUI enhancement for *Yoshimi 1.3.5* is color-coded identification of an instrument's use of ADD-, SUB-, and PAD-synth engines, no matter where in the instrument's kit they may be. This can be enabled/disabled in the mixer panel. It does slow down *Yoshimi*'s startup, but due to the banks reorganisation (done some time ago) it causes no delay in changing banks/instruments once *Yoshimi* is up and running. Some saved instruments seem to have had their "Info" section corrupted. *Yoshimi* can detect this issue, and step over it to find the true status. Also, if one resaves the instrument, not only will the PADsynth status be restored, but ADDsynth and SUBsynth will be included, allowing a faster scan next time.

6.1.4 Rotary Knobs

Visual rotary knobs are used for modifying numerical parameters in the user-interface. Horizontal, as well as vertical, mouse movements will adjust the knob. When rotated using the left mouse button, the rotary knobs give a coarse control of the numerical settings of the knob. When rotated using the right mouse button, the rotary knobs give a finer control of the numerical settings of the knob. One can also use the mouse scroll wheel to adjust rotary controls, which gives better control that using the mouse pointer. If the Ctrl key is held at the same time as the wheel is scrolled, the control is *extremely* fine.

6.1.5 Sliders

For vertical sliders only, if one holds down the right mouse button, then moves it slightly, the peg will go to it's default position. The same thing will happen if you click on the track with the right button.

(Doesn't seem to hold in the Parts panel, though).

6.1.6 Presets

The *ZynAddSubFX*/*Yoshimi* concept of presets is very powerful.

Absolutely every user-interface section that has blue **C** and **P** buttons can be stored in the `presets` directory. That includes entire Addsynth engines! When one looks at the copy/paste buffer, one sees only items that are relevant to the group that the C/P buttons are in.

As one wants to save, as well as load, these presets, it makes sense to copy all the default ones to a location such as `~/.config/yoshimi/presets`. That makes them fully accessible, but tucked away out of sight. *Yoshimi* creates this directory at first time start up. Preset files allow one to save the settings for any of the components which support copy/paste operations. This is done with preset files (`.xpz`), which get stored in the folders indicated by **Paths / Preset Dirs...**. Note that the number of preset directories that can be set is limited to 128 (the same as for roots and banks).

6.1.7 Automation

In *Yoshimi 1.3.5*, a number of existing, as well as new features have come together to give much greater flexibility (especially for automation) using standard MIDI messages. These are:

1. **NRPNs**
2. **ZynAddSubFX controls**
3. **Independent part control**
4. **16, 32 or 64 parts**
5. **Vector Control**
6. **Direct part stereo audio output**

1. NRPNs. NRPNs can handle individual bytes appearing in either order, and usually the same with the data bytes. Increment and decrement is also supported as graduated values for both data LSB and MSB. Additionally, the ALSA sequencer's 14-bit NRPN blocks are supported.

2. ZynAddSubFx controls. System and Insertion Effect controls are fully supported, with extensions to allow one to set the effect type and (for insertion effects) the destination part number.

3. Part control. Independent part control enables one to change instrument, volume, pan, or indeed any other available control of just that part, without affecting any others that are receiving the same MIDI

channel. This can be particularly interesting with multiply layered sounds. There are more extensions planned.

4. 16/32/64 Parts. With 32 and 64 parts, it helps to think of 2 or 4 rows of 16. When one saves a parameter block, the number of parts is also saved, and will be restored when one reloads. By default each *column* has the same MIDI channel number, but these can be independently switched around, and by setting (say) number 17 taken right out of normal access.

In tests, *compiling* for 64 parts compared with 16 parts increased processor load by a very small amount when *Yoshimi* was idling, but this becomes virtually undetectable once one has 8 or more instruments actually generating output. In normal use, selecting the different formats makes no detectable difference, but using the default 16 reduces clutter when one doesn't need the extras.

5. Vector control. Vector control is based on these parts columns, giving one either 2 (X only) or 4 (X + Y) instruments in this channel. Currently the vector CCs one set up can (as inverse pairs) vary any combination of volume, pan, and filter cut-off. More will be added. To keep the processor load reasonable it pays to use fairly simple instruments, but if one has sufficient processing power, it would be theoretically possible to set up all 16 channels with quite independent vector behavior!

6. Direct part audio. Direct part audio is JACK-specific, and allows one to apply further processing to just the defined part's audio output (which can still output to the main L+R if one wants). This setting is saved with parameter blocks. Currently it is only set in the mixer panel window, but it will also eventually come under MIDI direct part control. Again, to reduce unnecessary clutter, part ports are only registered with JACK if they are both enabled, and set for direct output. However, once set they will remain in place for the session to avoid disrupting other applications that may have seen them.

6.2 Filter Settings

This section describes filtering at a high level, in terms of frequency responses and other concepts of filtering. The end of this section covers a user interface used in filter settings. It is a stock-panel re-used in other user-interface elements. See section 6.2.5 "Filter Parameters User Interface" on page 74, if one is in a hurry.

Yoshimi offers several different types of filters, which can be used to shape the spectrum of a signal. The primary parameters that affect the characteristics of the filter are the cutoff, resonance, filter stages, and the filter type.

Filter stages are the number of times that this filter is applied in series. So, if this number is 1, one simply has this one filter. If it is two, the sound first passes the filter, and the results then pass the same filter again. In *ZynAddSubFX*, the wetness is applied after all stages were passed.

6.2.1 Filter Type

A filter removes or attenuates frequency elements or tones from a signal. Filtering changes the character of a signal.

The basic analog filters that *Yoshimi* and *ZynAddSubFX* offer are shown in figure 45 "Basic Filter Types" on page 72, with the center frequency being marked by the red line. The state variable filters should look quite similar.

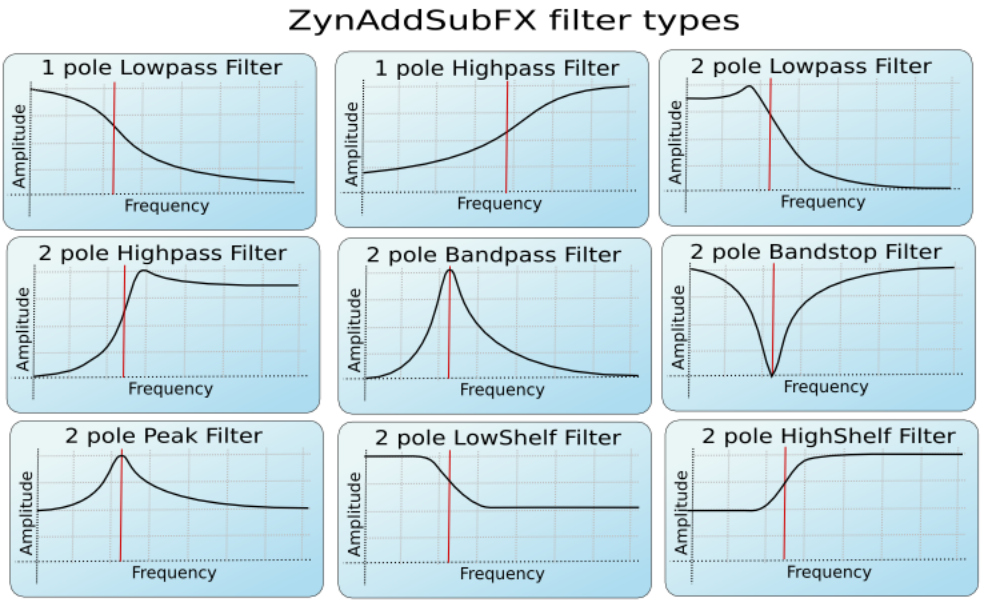

Figure 45: Filter Types, Yoshimi/ZynAddSubFX

1. A **low-pass** filter makes the sound more muffled.
2. A **band-pass** filter makes the sound more tone-like, and sometimes more penetrating, if the total energy in the passband is preserved as the bandwidth decreases.
3. A **high-pass** filter makes the sound seem sharper or more strident.

6.2.2 Filter Cutoff

The filter cutoff value determines which frequency marks the changing point for the filter. In a low pass filter, this value marks the point where higher frequencies begin to be attenuated.

6.2.3 Filter Resonance

The resonance of a filter determines how much excess energy is present at the cutoff frequency. In *Yoshimi* and *ZynAddSubFX*, this is represented by the Q-factor, which is defined to be the cutoff frequency divided by the bandwidth. In other words higher Q values result in a much more narrow resonant spike.

The Q value of a filter affects how concentrated the signals energy is at the cutoff frequency. The result of differing Q values are shown in figure 46 "Low Q vs. High Q" on page 73. For many classical analog sounds, high Q values were used on sweeping filters. A simple high Q low pass filter modulated by a strong envelope is usually sufficient to get a good sound.

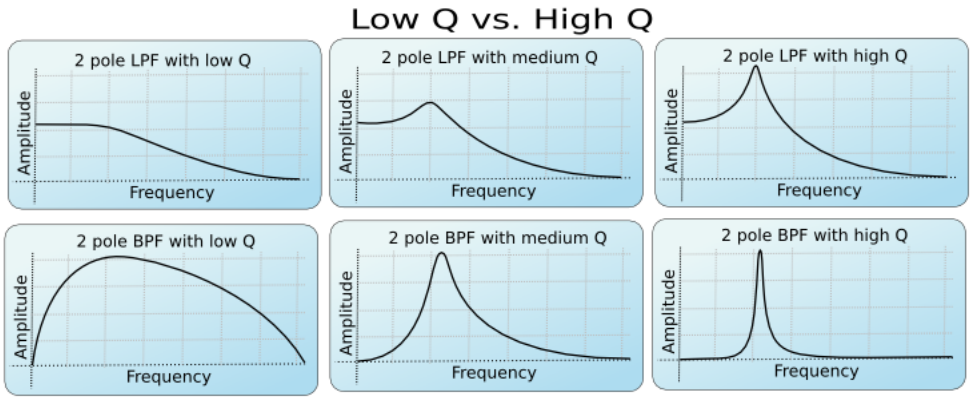

Figure 46: The Effect of the Q Value

6.2.4 Filter Stages

The number of stages in a given filter describes how sharply it is able to make changes in the frequency response. The more stages, the sharper the filter. However, each added stage increases the processor time needed to make the filter calculation.

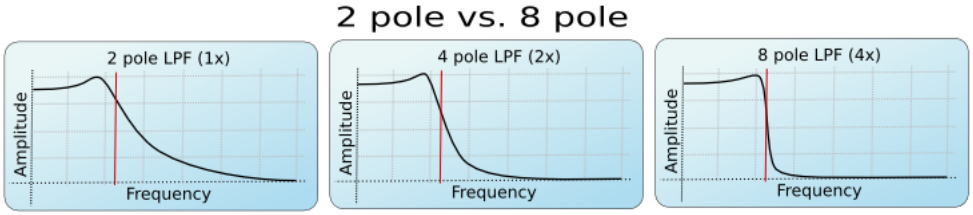

Figure 47: The Effect of the Order of a Filter

The affect of the order of the filter can be seen in the figure above. This is roughly synonymous with the number of stages of the filter. For more complex patches, it is important to realize that the extra sharpness in the filter does not come for free, as it requires many more calculations to be performed. This phenomena is the most visible in SUBsynth, where it is easy to need several *hundred* filter stages to produce a given note.

There are different types of filters. The number of poles define what will happen at a given frequency. Mathematically, the filters are functions which have poles that correspond to that frequency. Usually, two poles mean that the function has more "steepness", and that one can set the exact value of the function at the poles by defining the "resonance value". Filters with two poles are also often referred to as *Butterworth Filters*.

For the interested, functions having poles means that we are given a quotient of polynomials. The denominator has degree 1 or 2, depending on the filter having one or two poles. In the file `DSP/AnalogFilter.cpp`, `AnalogFilter :: computefiltercoefs()` sets the coefficients (depending on the filter type), and `AnalogFilter :: singlefilterout()` shows the whole polynomial (in a formula where no quotient is needed).

6.2.5 Filter Parameters User Interface

Figure 48: Stock Filter Parameters Sub-Panel

The user interface for filter parameters is a small stock sub-panel that is re-used in a number of larger dialog boxes, as shown in the figure above. Let's describe each item of this sub-panel.

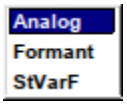

Figure 49: Filter Categories, Dropdown Box

1. **Category**
2. **Filter Type**
3. **C.freq**
4. **Q**
5. **V.SnsA**
6. **freq.tr**
7. **gain**
8. **St**
9. **C**
10. **P**

1. Category. Determines the category of filter to be used. There are three categories of filters (as shown in the dropdown element shown in figure 49 "Filter Categories Dropdown" on page 74).

1. **Analog** (the default)
2. **Formant**
3. **StVarF**

An **analog** filter is one that approximates a filter that is based on a network of resistors, capacitors, and inductors.

A **formant** filter is a more complex kind of filter that acts a lot like the human vocal tract, allowing for sounds that are a bit like human voices.

A **state variable** ("StVarF") filter is a type of active filter. The frequency of operation and the Q factor can be varied independently. This and the ability to switch between different filter responses make the state-variable filter widely used in analogue synthesizers.

Values: `Analog*, Formant, StVarF`

2. Filter Type. Selects the type of filter to be used, such as high-pass, low-pass, and band-pass. See the dropdown element in figure 50 "Filter Type Dropdown" on page 75.

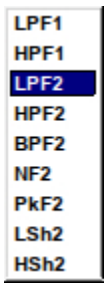

Figure 50: Type of Filter Passband, Dropdown Box

Values: `LPF1, HPF1, LPF2*, HPF2, BPF2, NF2, PkF2, LSh2, HSh2`

3. C.freq. Cutoff frequency or center frequency. This items has various definitions in the literature. Usually it refers to the frequency at which the level drops to 3 Db below the maximum level. In various dialogs, this value is the center frequency of the filter or the base position in a vowel's sequence.

Values: `0 to 127, 90*`

4. Q. The level of resonance for the filter. It indicates a measure of the sharpness of a filter. The higher the Q, the sharper the filter. Generally, a higher Q value leads to a louder, more tonal affect for the filter. Note that some filter types might ignore this parameter.

5. V.SnsA. Velocity sensing amount for filter cutoff. Velocity sensing amount of the filter.

Values: `0 to 127, 64*`

6. V.Sns. Velocity sensing function of the filter. Set the amplitude of the velocity sensing.

Values: `0 to 127, 64*`

7. freq.tr. Filter Frequency Tracking Amount. When this parameter is positive, higher note frequencies shift the filters cutoff frequency higher. For the filter frequency tracking knob, left is negative, middle is zero, and right is positive.

Values: `0 to 127, 64*`

8. gain. Filter gain. Additional gain/attenuation for a filter. Also described as the filter output gain/damping factor.

Values: `0 to 127, 64*`

9. St. Filter stages. The more filter stages applied to a signal, the stronger (in general) the filtering. It is the number of additional times the filter will be applied (in order to create a very steep roll-off, such as 48 dB/octave). This dropdown element is shown in figure 51 "Filter Stage Dropdown" on page 75. Obviously, the more stages used, the more calculation-intensive the filter will be. This should also increase the latency (lag) of the filter.

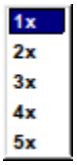

Figure 51: Filter Stage Dropdown

Also present in this sub-panel are the usual **C**opy and **P**aste buttons that call up a copy-parameters or paste-parameters dialog.

6.3 Stock Resonance Settings

Yoshimi provides for setting very arbitrary "resonance" settings for some sounds. In fact, "resonance" is too limiting a word. A lot of control over the *spectrum* is possible. The following dialog is used by the ADDsynth editor, figure 10.7 "ADDsynth / Resonance" on page 150, and the the PADsynth editor,

The resonance editor is brought on-screen via the **Resonance** button of the ADDsynth or PADsynth global part editors.

The resonance effect acts as a "resonance box" or a filter with arbitrary frequency response. This produces very realistic sounds. The cursor location is shown below the graph (the frequency, kHz, and the amplitude, dB).

Paul Nasca has a video on YouTube that includes a demonstration of how the resonance dialog works and affects the sound, if you care to look for it.

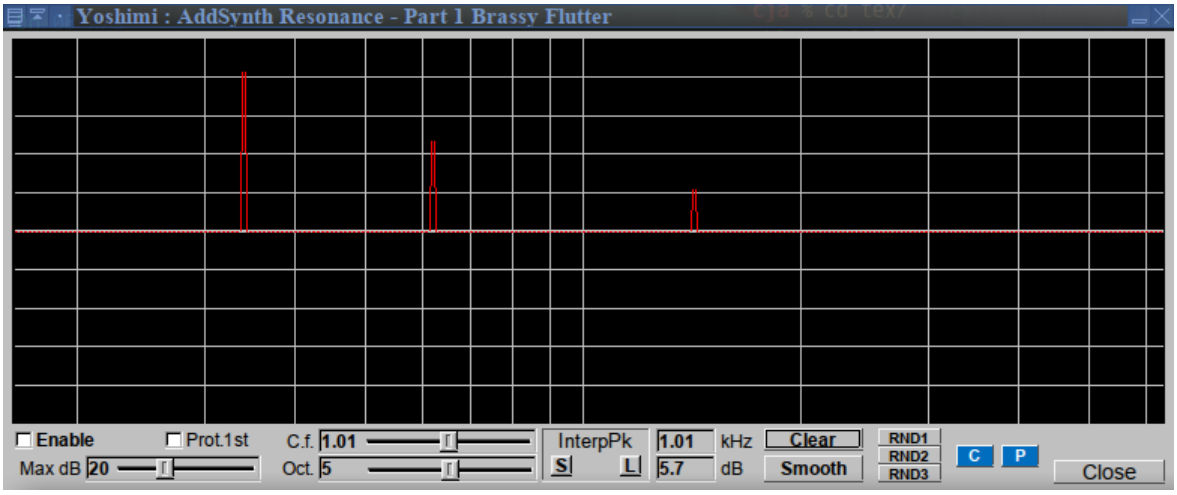

Figure 52: ADDsynth/PADsynth Resonance

1. **Graph Window**
2. **Enable**
3. **Max dB**
4. **C.f.**
5. **Oct.**
6. **P.1st** (gone)
7. **InterpPk**
8. **KHz**
9. **dB**
10. **Zero**
11. **Smooth**
12. **RND1**
13. **RND2**
14. **RND3**
15. **C**
16. **P**
17. **Close**

1. Graph Window. Resonance Graph Window. Lets one draw the resonance frequency response in "freehand" mode.

2. Enable. Resonance Enable. Turn the Resonance effect on.

Values: `Off*, On`

3. Max dB (wheel). The Maximum Amplitude (dB) wheel. Sets the amount of resonance: lower values have little effect. Use the roller below to set it.

Values: `1 to 90, 20*`

4. C.f. (knob). Center Frequency (kHz). Sets the center frequency of the graph. The value is shown in the read-only text-box to the left.

Values: `0 to 127, 64*` for `0.10 to 10.0, 1.0*`

5. Oct. Number of Octaves. Sets the number of octaves the graph represents. The value is shown in the read-only text-box to the left.

Values: `0 to 127, 64*` for `0 to 10, 5*`

6. P.1st. Protect the fundamental Frequency. Do not damp the first harmonic. What does this mean, what affect does it have?

Values: `Off, On`

7. InterpPk. Interpolate the resonance peaks. This setting used to be a weird one where the mouse button (left versus right) affected the kind of interpolation used, but also affects the next field as well, but in *Yoshimi* 1.3.9 the mechanism for interpolation has been made more clear. In addition, some of the controls have been changed to sliders for easier usage.

This setting allows one to make resonance functions very easily. To use it effectively, first, clear the graph using the **Clear** button. Click the left button on a position on the graph to create a peak (or do it more than once to create more peaks). Click either the **InterpPk S** button (smooth interpolation) or the **InterpPk L** button (linear interpolation). *Yoshimi* will interpolate automatically between the peaks drawn, as shown in figure 53 "ADDsynth/PADsynth Resonance Interpolated" on page 77, which shows *smooth interpolation*.

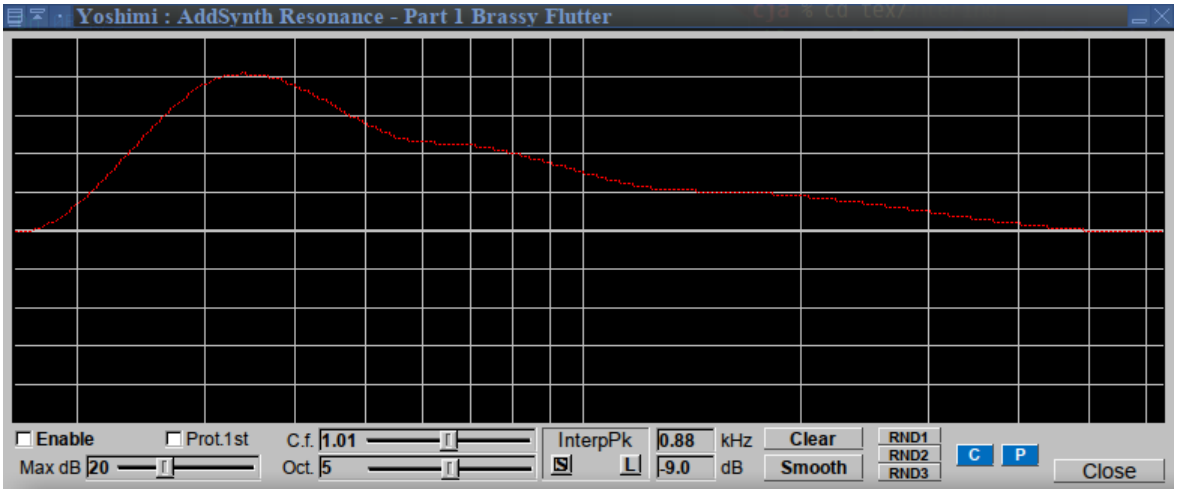

Figure 53: ADDsynth/PADsynth Resonance Interpolated

Please do not confuse this "smoothing" with the processing done using the **Smooth** button discussed below.

Also note that one can clear a part of the graph by dragging with the right mouse button. In fact, the **interpPk** functionality interpolates between non-zero values. Oh, and note that the **kHz** and **dB** fields update to match it. And don't forget to try the middle-click feature to see the corresponding command-line needed to achieve the setting.

8. KHz. The current frequency on graph.

9. dB. The current level on graph window.

Values: `-90 to +90`

10. Clear. Clear the resonance function. (Used to be called "Zero".) Clear the graph.

11. Smooth. Smooth the resonance function. Smooth the graph. This button causes each jagged portion of the graph to be smoothed. This smooth does not interpolate between the peaks, unlike the **InterpPk** functionality described earlier. Compare the interpolation shown in figure 53 "ADDsynth/PADsynth Resonance Interpolated" on page 77, and the smoothing shown in figure 54 "ADDsynth/PADsynth Resonance Smoothed" on page 78.

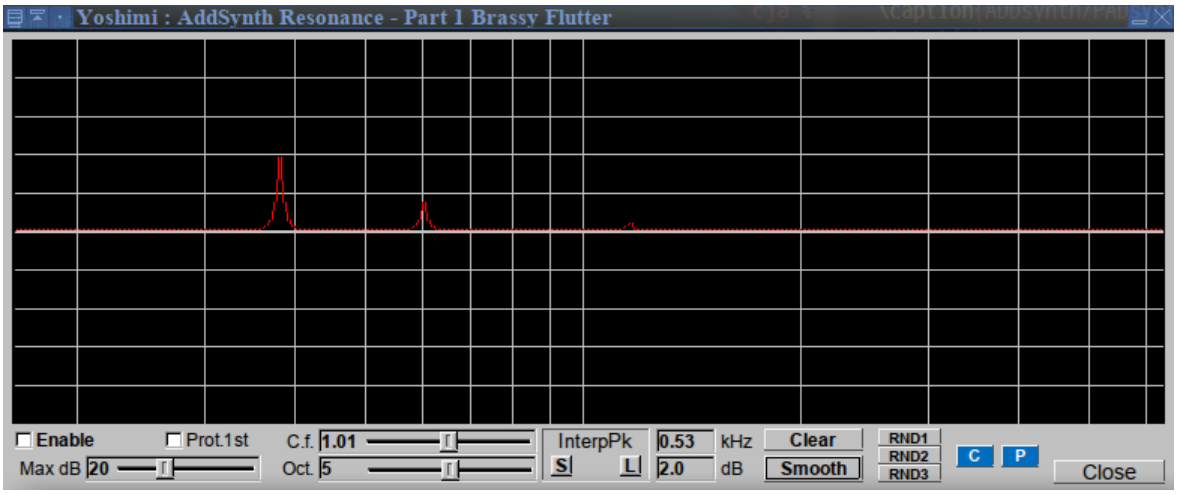

Figure 54: ADDsynth/PADsynth Resonance Smoothed

Note how the amplitude of the peaks is also reduced by the smoothing. Presumably, a frequency-smoothing window is applied to the peaks, thus making each new data-point a weighted average of the data-points around it. Don't forget to try the middle-click feature to see the corresponding command-line needed to achieve the setting.

12. RND1. Randomize the resonance function, 1. RND1, RND2, RND3 are used to create random resonance functions.

13. RND2. Randomize the resonance function, 2.

14. RND3. Randomize the resonance function, 3.

15. C. Copy Dialog.

16. P. Paste Dialog.

17. Close. Close.

6.4 LFO Settings

Yoshimi provides LFOs for it amplitude, frequency, and filtering functions. "LFO" means Low Frequency Oscillator. These oscillators are not used to make sounds by themselves, but they change parameters cyclically as a sound plays.

LFOs are, as the name says, oscillators with, compared to the frequency of the sound, low frequency. They often appear in order to control the effect.

6.4.1 LFO Basic Parameters

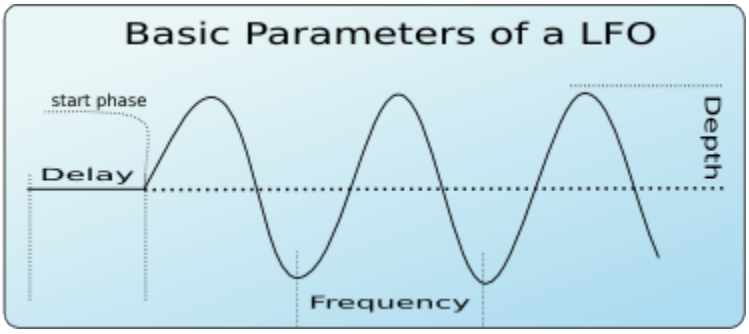

Figure 55: Basic LFO Parameters

1. **Delay**.
2. **Start Phase**.
3. **Frequency**.
4. **Depth**.

The LFOs has some basic parameters (see figure 55 "Basic LFO Parameters" on page 79.

1. Delay. LFO Delay. This parameter sets how much time takes since the start of the note to start the cycling of the LFO. When the LFO starts, it has a certain position called "start phase".

2. Start Phase. LFO Start Phase. The angular position at which a LFO waveform will start.

3. Frequency. LFO Frequency. How fast the LFO is (i.e. how fast the parameter controlled by the LFO changes.)

4. Depth. LFO Depth. The amplitude of the LFO (i.e. how much the parameter is controlled by the LFO changes.)

6.4.2 LFO Function

Another important additional LFO parameter is the shape or type of the LFO. There are many LFO Types that vary according to the function used to generate the LFO. *Yoshimi* supports the LFO shapes shown in figure 56 "LFO Functions" on page 80.

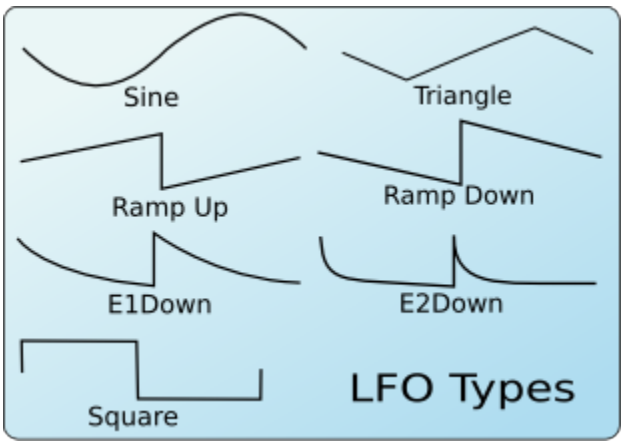

Figure 56: LFO Types, Shapes, or Functions

6.4.3 LFO Randomness

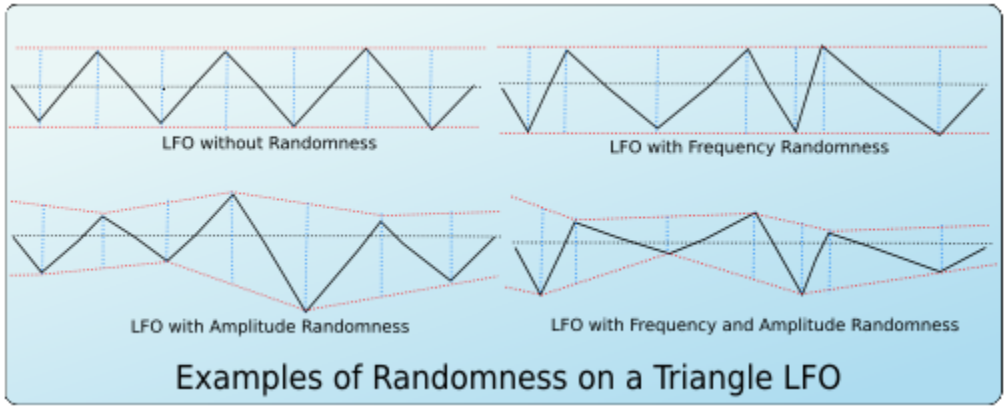

Figure 57: LFO Randomization

Another parameter is the LFO Randomness. It modifies the LFO amplitude or the LFO frequency at random. In *Yoshimi* one can choose how much the LFO frequency or LFO amplitude changes by this parameter. Observe figure 57 "LFO Randomization" on page 80. It shows some examples of randomness and how it changes the shape of a triangle LFO.

6.4.4 LFO, More Settings

Other settings are available as well.

Continous mode: If this mode is used, the LFO will not start from "zero" on each new note, but it will be continuous. This is very useful if one applies on filters to make interesting sweeps.

Stretch: It controls how much the LFO frequency changes according to the notes frequency. It can vary from negative stretch (the LFO frequency is decreased on higher notes) to zero (the LFO frequency will be the same on all notes) to positive stretch (the LFO frequency will be increased on higher notes).

6.4.5 LFO User Interface Panels

Figure 58: Amplitude LFO Sub-Panel

In *Yoshimi*, LFO parameters are available for amplitude, filters, and frequency. They all have essentially the same interface elements. Note figure 58 "Amplitude LFO Sub-Panel" on page 81, which shows an example of an LFO stock sub-panel.

These parameters are:

1. **Freq**
2. **Depth**
3. **Start**
4. **Delay**
5. **A.R**
6. **F.R**
7. **C** or **C.**
8. **Str**
9. **Type**
10. **C** (copy)
11. **P** (paste)

1. Freq. LFO Frequency. This parameter varies from 0 to 1. We still need to figure out what that scale means, however. Obviously, it is a relative scale, and is perhaps related to the overall sampling frequency.

Values: `0 to 1, 0.63*`

2. Depth. LFO Depth. Also called "LFO Amount".

Values: `0* to 127`

3. Start. LFO Start Phase. If this knob is at the lowest value, the LFO Start Phase will be random.

Values: `0 = random, to 127, 64*`

4. Delay. LFO Delay.

Values: `0* to 127`

5. A.R. LFO Amplitude Randomness.

Values: `0* to 127`

6. F.R. LFO Frequency Randomness.

Values: `0* to 127`

7. C. LFO Continous Mode.

Values: `Off*, On`

8. Str. LFO Stretch. See the image in figure 58 "Amplitude LFO Sub-Panel" on page 81. It shows that the LFO stretch is set to zero, though the tooltip would show it to be 64.

Values: `0 to 127, 64*`

9. Type. LFO Function. Also present in this sub-panel are the usual **C**opy and **P**aste buttons that call up a copy-parameters or paste-parameters dialog.

Values: `SINE*, TRI, SQR, R.up, R.dn, E1dn, E2dn`

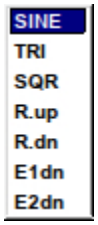

Figure 59: LFO Function Type Drop-down

10. Type. LFO Type (or Shape, or Function). The various shapes of LFO functions are shown in figure 56 "LFO Functions" on page 80. The values that can be selected are shown in figure 59 "LFO Type Drop-down" on page 82. Also present in this sub-panel are the usual **C**opy and **P**aste buttons that call up a copy-parameters or paste-parameters dialog.

Values: `SINE*, TRI, SQR, R.up, R.dn, E1dn, E2dn`

For reference, figure 60 "Filter LFO Sub-Panel" on page 82shows the LFO sub-panel for a filter, and figure 62 "Frequency LFO Sub-Panel" on page 83shows the LFO sub-panel for frequency.

6.4.6 Filter LFO Sub-panel

Figure 60: Filter LFO Sub-Panel

1. **Enable** (present on some versions of this sub-panel).
2. **Freq.**
3. **Depth**
4. **Start**
5. **Delay**
6. **Str.**
7. **C.**
8. **A.R.**
9. **F.R.**
10. **Type**
11. **C**
12. **P**

1. Enable. Enable the panel. (Present on some versions of this sub-panel).

2. Freq. LFO Frequency.

Values: `0 to 1, 0.64*`

3. Depth. LFO Amount.

Values: `0* to 127`

4. Start. LFO Startphase (leftmost is random).

Values: `0 to 127, 64*`

5. Delay. LFO Delay.

Values: `0* to 127`

6. Str. LFO Stretch.

Values: `0 to 127, 64*`

7. C. Continuous LFO.

Values: `Off*, On`

8. A.R. LFO Amplitude Randomness.

Values: `0* to 127`

9. F.R. LFO Frequency Randomness.

Values: `0* to 127`

10. Type. LFO Type.

Figure 61: LFO Function Type Dropdown

Values: `SINE*, TRI, SQR, R.up, R.dn, E1dn, E2dn`

11. C. Copy to Clipboard/Preset.

12. P. Paste from Clipboard/Preset.

6.4.7 Frequency LFO Sub-panel

Figure 62: Frequency LFO Sub-Panel

This panel is basically identical to the Filter LFO panel described in the previous section.

6.5 Envelope Settings

Envelopes control how the amplitude, the frequency, or the filter changes over time. The general envelope generator has four sections:

1. **Attack**. The attack is the initial envelope response. It begins when the key for the note is first held down (at Note On). The volume starts at 0, and rises fast or slowly until a peak value. In *Yoshimi*, the attack is always linear.
2. **Decay** When the attack is at its highest value, it immediately begins to decay to the sustain value. The decay can be fast or slow. The attack and decay together can be used to produce something like horn blips, for example.
3. **Sustain** This is the level at which the parameter stays while the key is held down, i.e. until a Note Off occurs.
4. **Release** When the key is released, the sound decays, either fast or slowly, until it is off (the volume is 0).

Together, these values are called "ADSR". The ADSR envelope generally controls the amplitude of the sound. In *Yoshimi*, amplitude envelopes can be *linear* or *logarithmic*.

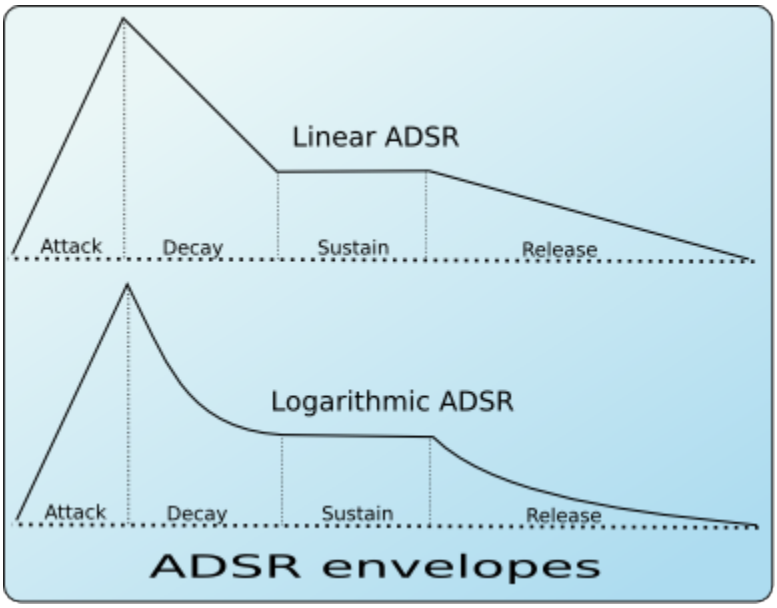

Figure 63: ADSR Envelope (Amplitude)

See figure 63 "ADSR Envelope (Amplitude)" on page 84, it shows a depiction of an ADSR envelope. The ADSR is mostly applied to amplitude envelopes.

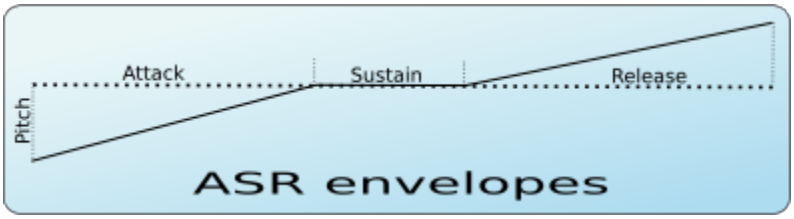

Figure 64: ASR Envelope, Frequency

Frequency envelopes control the frequency (more exactly, the pitch) of the oscillators. The following image depicts the stages of these envelopes.

For frequency envelopes, a simpler form of envelope is used. This envelope is an ASR envelope, shown in figure 64 "ASR Envelope, Frequency" on page 84. The dotted line represents the real pitch of the sound without the envelope. The frequency envelopes are divided into 3 stages:

1. **Attack**. It begins at the Note On. The frequency starts from a certain value and glides to the real frequency of the note.
2. **Sustain**. The frequency stays the same during the sustain period.
3. **Release**. This stage begins on Note Off and glides the frequency of the note to a certain value.

6.5.1 Amplitude Envelope Sub-Panel

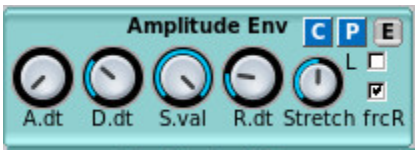

Figure 65: Amplitude Envelope Sub-Panel

1. **A.dt**
2. **D.dt**
3. **S.val**
4. **R.dt**
5. **Str**
6. **L**
7. **frcR**
8. **C**
9. **P**
10. **E**

1. A.dt. Attack duration, attack time. We need to determine the units of time at play for ADSR durations.

Values: `0* to 127`

2. D.dt. Decay duration, decay time.

Values: `0 to 127, 44*`

3. S.val. Sustain value. This is the (relative?) level at which the envelope will settle while the note is held down. The only stage that always remains defined is the Sustain, where the envelopes freezes until a Note Off event.

Values: `0 to 127*`

4. R.dt. Release time.

Values: `0 to 127, 25*`

5. Str. Stretch. How the envelope is stretched according the note. Envelope Stretch means that, on lower notes, the envelope will be longer. On the higher notes the envelopes are shorter than lower notes. In the leftmost value, the stretch is zero. The rightmost use a stretch of 200%; this means that the envelope is stretched about 4 times per octave.

Values: `0 to 127, 64*`

6. L. Linear envelope. If this option is set, the envelope is linear, otherwise, it will be logarithmic.

Values: `Off*, On`

7. frcR. Forced release. This means that if this option is turned on, the release will go to the final value, even if the sustain stage is not reached. Usually, this must be set. If this option is turned on, the release will go to the final value, even if the sustain level is not reached. Also present in this sub-panel are the usual **C**opy and **P**aste buttons that call up a copy-parameters or paste-parameters dialog.

Values: `Off, On*`

8. C. opy to Clipboard/Preset.

9. P. aste from Clipboard/Preset.

10. E. mplitude Envelope Editing Window. Described in the next section.

6.5.2 Envelope Settings

This section describes the **Amplitude Envelope Editing** window.

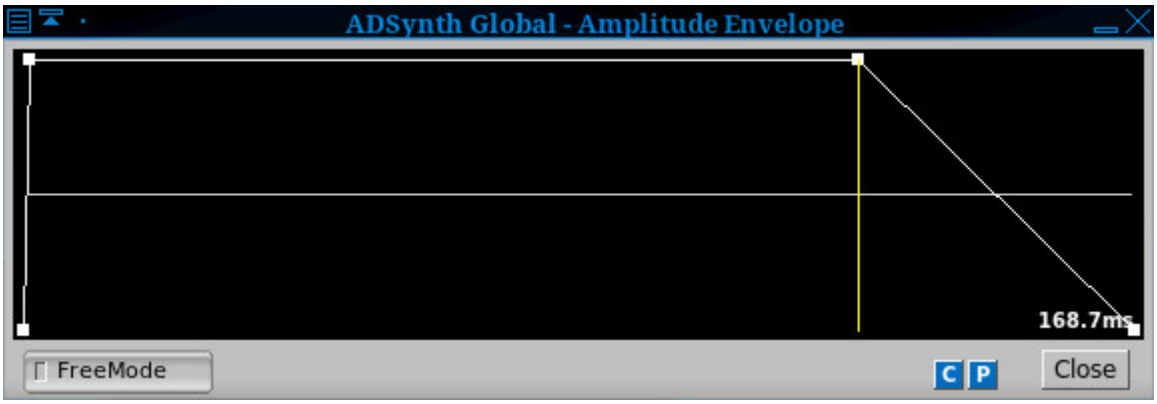

Figure 66: Amplitude/Filter/Frequency Envelope Editor

1. **Graph Window**
2. **FreeMode**
3. **C**
4. **P**
5. **Close**

1. FreeMode. Freemode Enable. Enables the envelope editor's Free Mode. See the next section for details.

Values: `Off*, On`

6.5.3 Freemode Envelope Settings

The envelope panels are parts that control a parameter (such as the frequencies) of a sound. For all envelopes, there is a mode that allows the user to set an arbitrary number of stages and control points. This mode is called *Freemode*. The only stage that always remains defined is the Sustain, where the

envelopes freezes until a Note Off event. The Freemode envelope editor has a separate window to set the parameters and controls.

The main concept of the freemode editor window is the *control point*. One can move the points using the mouse. In the right on the window, it shows the total duration of the envelope. If the mouse button is pressed on a control point, it will be shown the duration of the stage where the point is.

figure 67 "Amplitude/Filter/Frequency Envelope Freemode Editor" on page 87shows an example of the stock Freemode envelope editor, with Freemode enabled.

Figure 67: Amplitude/Filter/Frequency Envelope Freemode Editor

All of the envelope editors have some common controls.

1. **Graph Window**
2. **Add point**
3. **E**
4. **FreeMode**
5. **Add point**
6. **Delete point**
7. **Sust**
8. **Stretch**
9. **L**
10. **frcR**
11. **Close**
12. **C**
13. **P**

1. E. Editor. Graph Window. Shows a window with the real envelope shape and the option to convert to Freemode to edit it. The envelope editor shows a window in which one can view and modify the detailed envelope shape, or convert it to Freemode to edit it almost without restriction. By default, only the *Freemode* button/checkbox is visible.

If an envelope has FreeMode enabled, it allows one to edit the graph of the envelope directly. Select a point from the graph and move it. Notice that *only the line* ***before*** *the currently edited point of the envelope* changes its duration. As the point is dragged, the text on the right shows the duration of the line before it. Otherwise, the text shows the total duration of the envelope.

If the envelope doesn't have the FreeMode mode enabled, it doesn't allow one to move the points; the envelope window is then useful only to see what happens if one changes the ADSR settings.

2. FreeMode. FreeMode. Provides a mode where completely arbitrary envelopes may be drawn. Actually, the envelopes aren't completely arbitrary, as the sustain section is always flat, and its duration corresponds with the duration the note is held down. When this mode is enabled, the rest of the controls shown in figure 67 "Amplitude/Filter/Frequency Envelope Freemode Editor" on page 87appear, and are described in the following paragraphs.

Values: `Off*, On`

3. Add point. Add point. Provides a way to add a data point to the Freemode envelope. It adds the point after the currently-selected point. One can select a point by clicking on it.

4. Delete point. Delete point. Provides a way to delete the current data point from the Freemode envelope.

5. Sust. Sustain point. Sets the sustain point. The sustain point is shown using the yellow line. If the point is at 0, then sustain is disabled. It is difficult to determine the difference between 1 and 2.

1. 0 means that sustain is disabled, and the envelope immediately starts dying, even if the note is held.
2. 1 seems to mean the sustain curve follows its course while the note is held.
3. 2 seems to mean that extra sustain kicks in after the note is released.

Values: `0, 1, 2*`

6. Stretch. Envelope Stretch. How the envelope is stretched according the note. On the higher notes the envelopes are shorter than lower notes. At the leftmost value, the stretch is zero. The rightmost sets a stretch of 200%; this means that the envelope is stretched about four times/octave.

7. L. Envelope Linear. This setting is only available in the amplitude envelope. If enabled, the envelope is linear. If not enabled, the envelope is logarithmic (dB).

Values: `Off*, On`

8. frcR. Forced Release. This means that if this option is turned on, the release will go immediately to the final value, even if the sustain stage is not reached. Usually, this must be set. When the key is released, the position of the envelope jumps directly to the point after the release point. If the release is disabled, the envelope position jumps to the last point on release.

Values: `Off*, On`

9. Close. Close Dialog.

Also present in this sub-panel are the usual **C**opy and **P**aste buttons that call up a copy-parameters or paste-parameters dialog, as well as a button to bring up the editor window.

6.5.4 Envelope Settings, Frequency

These envelopes controls the frequency (more exactly, the pitch) of the oscillators. Observe figure 64 "ASR Envelope, Frequency" on page 84. It depicts the stages of these envelopes. The dotted line represents the real pitch of the sound without the envelope.

The frequency envelopes are divided into 3 stages: attack (see 1.); sustain (see 3.); and release (see 4.).

One question to answer is: can the attack and release go in the opposite directions, or do the knob ranges prohibit this?

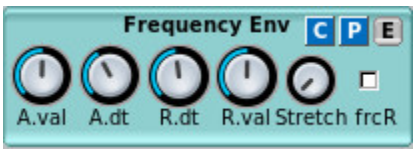

Figure 68: Frequency Envelope Sub-Panel

1. **Enable** (present on some versions of this sub-panel).
2. **A.value** or **A.val**
3. **A.dt**
4. **R.dt**
5. **R.val** (present on some versions of this sub-panel).
6. **Stretch**
7. **frcR**
8. **C**
9. **P**
10. **E**

For Frequency Envelopes the interface has the following parameters:

1. Enable. Enable the panel. (Present on some versions of this sub-panel).

2. A.val. Attack value. We need to figure out what this means.

Values: `0 to 127, 64*`

3. A.dt. Attack duration. Attack time.

Values: `0 to 127, 40*`

4. R.dt. Release time.

Values: `0 to 127, 60*`

5. R.val. Release Value. Actually present only on the Frequency Env sub-panel.

Values: `0 to 127, 64*`

6. Stretch. Envelope Stretch. Envelope Stretch (on lower notes make the envelope longer).

Values: `0 to 127, 64*`

7. frcR. Forced release. If this option is turned on, the release will go to the final value, even if the sustain level is not reached.

Values: `Off, On*`

Also present in this sub-panel are the usual **C**opy and **P**aste buttons that call up a copy-parameters or paste-parameters dialog, as well as a button to bring up the editor window.

6.5.5 Envelope Settings for Filter

This envelope controls the cutoff frequency of the filters. The filter envelopes are divided into 4 stages:

1. **Attack**. It begins at the Note On. The cutoff frequency starts from a certain value and glides to another value.
2. **Decay**. The cutoff frequency continues to glide to the real cutoff frequency value of the filter (dotted line).

3. **Sustain**. The cutoff frequency stays the same during the sustain period (dotted line).
4. **Release**. This stage begins on Note Off and glides the filter cutoff frequency of the note to a certain value.

Figure 69: Filter Envelope Sub-Panel

1. **A.value**
2. **A.dt**
3. **D.val**
4. **D.dt**
5. **R.dt**
6. **Stretch**
7. **frcR**
8. **L**

Filter Envelopes has the following parameters:

1. A.value. Attack Value. Starting Value. We need to figure out what this means.

Values: `0 to 127, 64*`

2. A.dt. Attack Duration. Attack Time.

Values: `0 to 127, 40*`

3. D.val. Decay Value.

Values: `0 to 127, 64*`

4. D.dt. Decay Duration. Decay Time.

Values: `0 to 127, 70*`

5. R.dt. Release time.

Values: `0 to 127, 60*`

6. Stretch. Stretch. Envelope Stretch (on lower notes make the envelope longer).

Values: `0 to 127, 64*`

7. frcR. Forced Release. If this option is turned on, the release will go to the final value, even if the sustain level is not reached.

Values: `Off, On*`

Also present in this sub-panel are the usual **C**opy and **P**aste buttons that call up a copy-parameters or paste-parameters dialog, as well as a button that bring up the editor window.

8. L. If this option is set, the envelope is linear, otherwise, it will be logarithmic.

Values: `Off*, On`

6.5.6 Formant Filter Settings

This window allows one to change most of the parameters of the formant filter. It is reached by enabling a **FILTER** panel, changing the **Category** value to *Formant*, and then clicking the **Edit** button that sits below the category drop-down list.

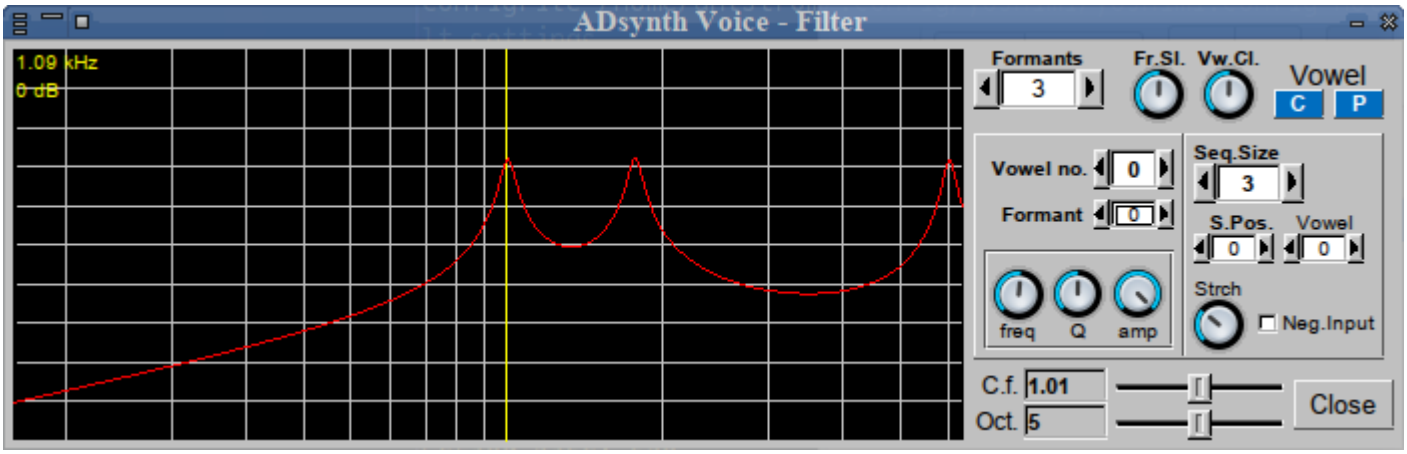

Figure 70: Formant Filter Editor Dialog

This editor dialog provides a lot of functionality:

1. **Formants**
2. **Fr.Sl.**
3. **Vw.Cl.**
4. **C.f.**
5. **Oct.**
6. **Vowel no.**
7. **Formant**
8. **freq**
9. **Q**
10. **amp**
11. **Seq.Size**
12. **S.Pos.**
13. **Vowel**
14. **Strtch**
15. **Neg Input**

6.5.6.1 Formant Parameters

9. Formants. Number of Formants Used.

Values: `1 to 12, 3*`

10. Fr.Sl. Formant Slowness. This parameters prevents too-fast morphing between vowels.

Values: `0 to 127, 64*`

11. Vw.Cl. Vowel "Clearness". Sets how much the vowels are kept "clear", that is, how much "mixed" vowels are avoided.

Values: `0 to 127, 64*`

12. C.f. Center Frequency. This slider control changes the center frequency of the graph, in relative units. Not quite sure how to describe this one, so play with the slider and see (and hear) for yourself.

Values: `0.09 to 10.00, 1.0*`

13. Oct. Number of Octaves. This slider controls the number of octaves shown in the graph.

Values: `0 to 10`

6.5.6.2 Formant Vowel Parameters

14. Vowel no. Vowel Number. The number of the current vowel. Each number represents a different vowel, and leads to a gross change in the shape of the formant spectrum. We do not yet have a mapping between the numbers and which vowel is represented.

Values: `0 to 5`

15. Formant. Formant Number. The current formant to be emphasized or modified. The vertical marker in the graph moves as this value is changed.

Values: `0 to 11`

16. freq. Formant Frequency. The frequency of the current formant. This knob changes the frequency of the formant peak selected by the Formant Number control.

Values: `0 to 127`

17. Q. Formant Resonance, Formant Q. The Q (resonance depth or bandwidth) of the current formant. Used to sharpen or make the current formant sound dull.

Values: `0 to 127`

18. amp. Formant Amplitude. Controls the amplitude of the current formant. Initially, one will want to set this to the maximum value.

Values: `0 to 127`

6.5.6.3 Formant Sequence Parameters

The sequence represents what vowel is selected to sound according to the input from the filter envelopes and LFO's. We need to learn a bit about how this setup actually works.

19. Seq Size. Sequence Size. The number of vowels in the sequence.

Values: `1 to 7`

20. S.Pos. Sequence Position. The current position of the sequence.

Values: `0 to 6`

21. Vowel. Vowel Position. The vowel from the current position.

Values: `0 to 5`

22. Strtch. How the sequence is stretched. This number probably means that the duration of the sequence decreases as the pitch of the selected notes increase.

Values: `0 to 127`

23. Neg Input. Negative Input. If enabled, the input from the envelope or LFO control is reversed.

Values: `off, on`

6.6 Clipboard Presets

In many of the settings panels, there are buttons labelled **C**, **P**, and **E**, **E** is the editor window, discussed in section 6.5.3 **C** and **P** are the clipboard/preset copy and paste dialogs, respectively. These buttons allow cut-and-paste for shorter sections of the XML configuration.

The preset dialog also provides a way to save a preset to a preset file. The naming convention for a preset file is `presetname.presettype.xpz`, where *presename* is the name one types into the **Copy to Preset** name field, *presettype* is the name that appears in the **Type** field, and *xpz* is the file-extension for compressed XML preset files.

The presets are stored in the current default preset directory, which is normally `~/.config/yoshimi/presets`. Preset directories can be added to the list, and the default preset directory can be changed. See section 5.4 "Menu / Paths" on page 60.

6.6.1 Clipboard/Preset Copy

Note that figure 71 "Copy to Clipboard" on page 93shows an example of the copying dialog for the clipboard.

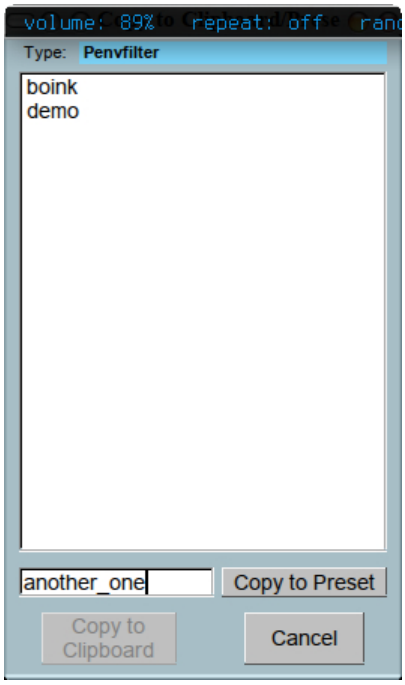

Figure 71: Copy to Clipboard/Presets

1. Type. Clipboard type for copying. This field indicates the context (e.g h. "envamplitude") or name of the clipboard to which the data will be copied. If the preset is saved/copied to a file, this field becomes the second part of the preset's file-name.

2. Clipboard list. Clipboard list. This itemm is actually a list of preset files available to be selected for this block of *Yoshimi* settings.

3. Copy to Preset. Clipboard to preset. Provides a way to specify the preset (and, indirectly, the preset file) to which this data should be copied.

To save to a preset, type the desired name of the setting. This entry will enable this button. When the button is pressed, the preset will be saved to the default preset directory. Be sure to set up a default present directory where ordinary users have write permissions! A good choice for a preset directory is `~/.config/yoshimi/presets`. The file-name of the of the preset will be a non-hidden file such as

```
my_preset.ADnoteParameters.xpz
```

The middle part of this name is shown near the top of the preset dialog, as a cue. There is no way in *Yoshimi* to change this part of the file-name. And don't do it using file system commands! Modify the first part of the file-name to distinguish it from other versions of the preset. Only the type-name will ever be visible in the *Yoshimi* presets **Type** field.

Note that *Yoshimi* ships with a number of non-hidden `.xpz` files.

4. Copy to Clipboard. Copies the preset to the clipboard.

6.6.2 Clipboard/Preset Paste

Observe figure 72 "Paste from Clipboard" on page 94. It shows an example of the pasting dialog for the clipboard.

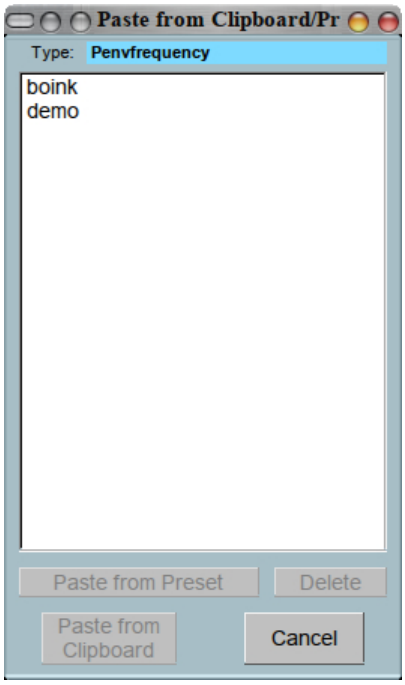

Figure 72: Paste from Clipboard/Presets

1. **Add point**
2. **Type**
3. **Clipboard list**
4. **Paste from Preset**
5. **Paste from Clipboard**

1. Type. Clipboard type for pasting. This field indicates the context (e.g h. "envamplitude") or name of the clipboard to which the data will be copied.

2. Clipboard list. Clipboard list.

3. Paste from Preset. Paste from preset. Provides a way to specify the preset to which this data should be copied.

4. Paste from Clipboard. Clipboard to preset.

7 Top Panel

The *Yoshimi* top panel provides quick access to some major features of the application. The top panel is shown in figure 2 "Yoshimi Main Screen, 1.3.8 and Above" on page 17.

Here are the major elements of the top panel.

1. **Stop!**
2. **Panel**
3. **VirKbd**
4. **Key Shift**
5. **Detune**
6. **Reset Detune**
7. **Volume**

1. Stop!. Stop! This button causes *Yoshimi* to "Cease all sound immediately!"

2. Panel. This button brings up a panel that shows a "mixer" view of all of the parts that have been created in the current state of *Yoshimi*.

For the details of this panel, see section 7.1 "Mixer Panel Window" on page 96.

3. VirKbd. This button brings up the virtual keyboard, which is a way to enter MIDI information without a real MIDI keyboard. It also provides a way to use the computer keyboard for faster playing. See section 7.2 "Virtual Keyboard" on page 98.

4. Key Shift. Master Key Shift. This is the key-shift (transpose) that applies to all parts, in units of semitones. In recent versions of *Yoshimi*, this range has been extended. Also note that the master key shift can be set via the user-interface, the command-line, or (we suspect) by MIDI NRPN commands.

Values: `-36 to 36, 0*`

5. Detune. Detune. Provides a global fine detune functionality. The fine detune mapping to the knob values shown below is -64 to 63 cents.

Values: `0 to 127, 64*` (float)

6. Reset, Detune. Reset detune. Resets the overall detuning functionality of *Yoshimi* off. Resets the global fine detune to 0.

7. Volume. Volume, Master Volume. Controls the overall volume of all sounds generated by *Yoshimi*.

Values: `0 to 127, 90*`

7.1 Mixer Panel Window

The *Panel* button opens the mixer panel window. The mixer panel window provides a global view of the most important adjustable parameters of all of the defined parts. There are two views, a 2x8 view and a 2x16 view. See figure 73 "Yoshimi Part Panel" on page 97, which shows the 2x8 view.

The Panel Window allows one to edit some important part parameters (instrument/volume/panning/etc..) and it acts like a mixer. Also, this window shows VU-meters for each part. To make a part the current part, left-click on its **Edit** button. To edit an instrument, right-click on the **Edit** button for that instrument.

When using the JACK audio backend, parts can be individually routed or sent to the main L/R outputs, either by themselves, or working with the main Left and Right outputs at the same time. This is controlled from the panel window, and the settings are saved with all the other parameters.

The individual part outputs will have the part effects, and any **Insertion** effects that are linked to them, but not the **System** effects. Direct part outputs carry the part and insertion effects, but not system ones.

Yoshimi used to register all parts with JACK by default, but that is a bit much now that 64 parts are available, so now *Yoshimi* uses an "on demand" model.

In the mixer panel window one will see a field just above the **Edit** button. This field determines the audio destination on a part-by-part basis, defaulting to just the main L+R pair. The direct part outputs are only exposed on parts that are active, and have the destination set to either **part** or **both**. Once activated, they will remain in place for the entire session, even if the part is later disabled or routed to main only. This is so that other programs won't see links suddenly disappear, although they will become silent. This setting is preserved in Yoshimi's patch sets and will be re-instated when next loaded.

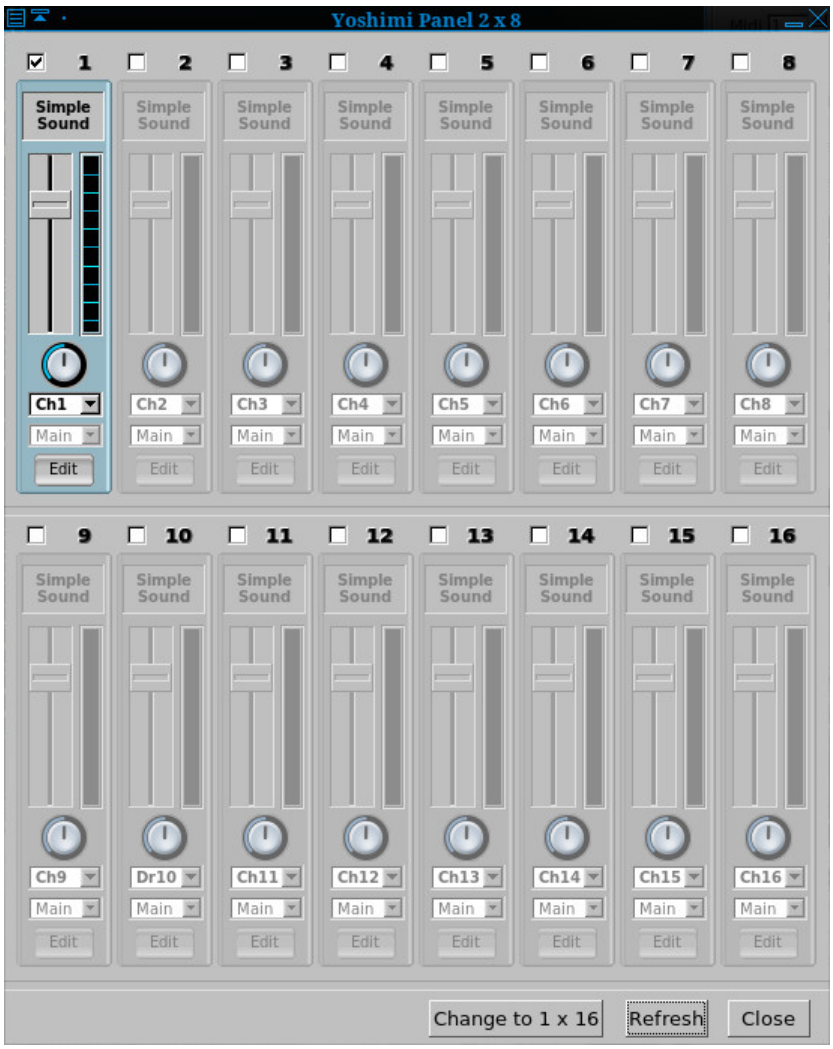

Figure 73: Yoshimi Part Panel, 2x8 View

1. Part Summary. Parts View or Summary.

2. Enable part. Enable/Disable the part. The check-box enables/disables the part. When the part is disabled, its controls are greyed out.

Values: `Off*, On`

3. Part name. Instrument name. Click on this box to change the instrument (it will open up the **Edit** window.

4. Volume Slider. Volume Bar. Changes the volume of the part.

5. VU-meter display. Shows the level of the part when playing.

6. Panning Knob. Panning Dial-Button. Changes the panning of the part.

Values: `0 (left) to 64* (center) to 127 (right)`

7. Channel. Receive from MIDI channel. Changes the MIDI channel assigned to the part.

Values: `Ch1*, Ch2, ..., Ch16`

8. Main. Set Audio Destination. Sets the audio for this part to be routed to the main audio output, to the audio specified by the part setup, or to both outputs. This option requires that *Yoshimi* use JACK audio. If running ALSA, this option is disabled (greyed out). The part's audio destination (JACK) is saved with the parameter sets, and so is the number of available parts. (*ZynAddSubFX* will still load these files, but it ignores any settings it doesn't recognise. If one re-saves in *ZynAddSubFX*, the settings will be lost.)

Values: `Main, Part, or Both`

9. Edit. The Edit button provides two function Left mouse button: Part select. Right mouse button: Instrument edit. This setup is a bit unintuitive, but the tooltips make it clear which click one might want to use.

10. Parts Layout. Changes the layout of the panel to the other layout, either **Change to 2 x 8** or **Change to 1 x 16**.

11. Close. Close the window.

7.2 Virtual Keyboard

This section describes the detailed usage of the *Yoshimi* virtual keyboard. The virtual keyboard lets one play notes using the keyboard/mouse. There is no MIDI requirement.

Using the keyboard: The keyboard is split into two "octaves"(in fact it is more than 1 octave). It may happen that the keys will not trigger any note-on This is because another widget than the keyboard itself is selected. In order to continue playing using the keyboard, click with the mouse on some keys on the virtual keyboard.

Using the mouse: One can use the mouse too, to play. If one presses the shift key while pressing the mouse button, the keys will be not released when the mouse button is released. If one presses the **Stop!** or "Panic" button from the *ZynAddSubFX*/*Yoshimi* main window, all keys will be released.

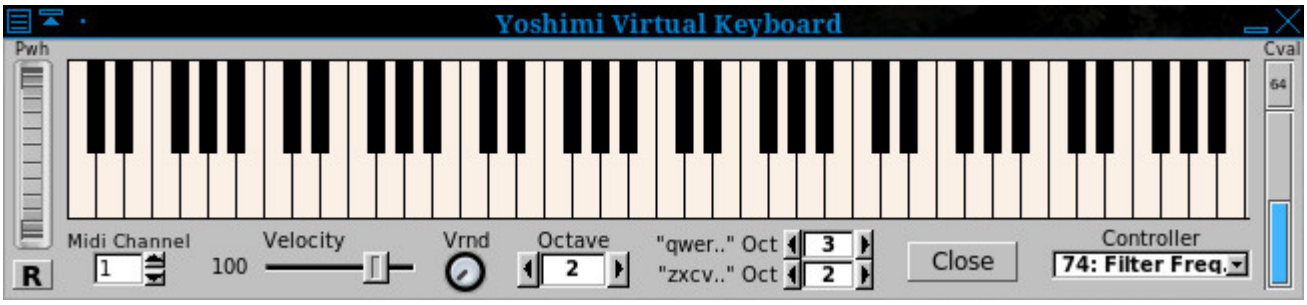

Figure 74: Yoshimi Virtual Keyboard

7.2.1 Virtual Keyboard, Basics

1. **Pwh**
2. **R**
3. **Midi Channel**
4. **Velocity**
5. **Velocity**
6. **Octave**

7. **"qwer.." Oct**
8. **"zxcv.." Oct**
9. **Controller**
10. **Cval**
11. **Close**

12. Pwh. Pitch bend knob. Pitch wheel. Press the **R** button to reset it.

13. R. Reset Pitch Bend.

14. Midi Channel. MIDI Channel. Sets the MIDI channel for the virtual keyboard.

Values: `1* to 16`

15. Velocity. Velocity of Notes. Sets the note-on velocity for the virtual keyboard.

Values: `1 to 127, 100*`

16. Velocity. Velocity Randomness.

Values: `0* to 127`

17. Octave. Transposes all of the virtual keyboard notes by the given number of octaves.

Values: `1, 2*, 3, 4, 5`

18. "qwer.." Oct. q2w3e4r5t6y Octave. Transposes the upper keys ("qwert"); the range of these keys is from C-4 to A-5 (replace the '5' with the octave).

19. "qwer.." Oct. zsxdcfvgbh Octave. Transposes the lower keys ("zxcvb"); the range of these keys is from C-3 to E-4 (replace the '4' with the octave).

Values: `1, 2*, 3, 4, 5`

20. Controller. Keyboard Controller.

Values: `01:Mod.Wheel, 07:Volume, 10:Panning, 11:Expression, 64:Sustain, 65:Portamento, 71:Filter Q, 74:Filter Freq*, 75:Bandwidth, 76:FM Gain, 77:Res.c.freq, 78:Res.bw.`

Sets the controller to be changed according to the **Cval** controller. See section 7.2.3 "Virtual Keyboard, Controllers" on page 100.

21. Cval. Controller value. Changes the controller value. This item consists of two parts. The top part shows a tiny number representing the current value of the selected controller. The bottom part is a combination value-bar and slider that one can move up and down with the mouse, to change the controller value. Note that the **Cval** value might not reflect the internal value of the controller when one changes the controller.

Values: `1 to 127, 96*`

22. Close. Close button.

7.2.2 Virtual Keyboard, ASCII Mapping

In addition to this virtual keyboard, the QWERTY (or Dvorak, or AZERTY) keyboards can be used to produce notes. The computer keyboard layout is shown in figure 12 "QWERTY Virtual Keyboard Layout" on page 38, From lowest octave to highest, the colors are blue, then green, then red. The "white" keys are the light colors, and the "black" keys are the deeper colors. The range of the keys on the "zxcvb..." row is C3 to E4. The range of the keys on the "qwert..." row is C4 to A5. These octave ranges can be adjusted.

The computer keyboard will produce notes only when the virtual keyboard is active. Also note that we replaced the monopoly symbol with the monopolist symbol. On X11 systems, this key is known as the "Super" key.

7.2.3 Virtual Keyboard, Controllers

This section (will give) a brief overview of the controller's that this window supports.

1. **01: Mod. Wheel**
2. **07: Volume**
3. **10: Panning**
4. **11: Expression**
5. **64: Sustain**
6. **65: Portamento**
7. **71: Filter Q**
8. **74: Filter Freq.**
9. **75: Bandwidth**
10. **76: FM Gain**
11. **77: Res. c. freq**
12. **78: Res. bw.**

The following figure shows the corresponding drop-down list of controller values, each preceded by its MIDI control number, re 1.

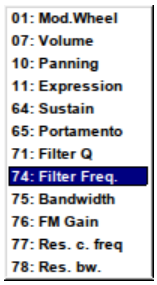

Figure 75: Virtual Keyboard Controllers

One question worth answering is if these controls, when changed, emit MIDI messages to change the control, versus them simply affecting the current playback.

1. Mod. Wheel. Sets the MIDI modulation value. This control will only have an effect on certain instruments. (It has no effect on the "Simple Sound", for example).

2. Volume. Controls the overall volume of the instrument being played by the virtual keyboard.

3. Panning. Controls the left-right location of the sounds played by the virtual keyboard.

4. Expression. Controls the expression. This probably can have different effects depending on the instrument. For example, with the "Simple Sound", this control is a lot like volume.

5. Sustain. Controls the sustain duration. This works even with the "Simple Sound". Using it makes even this virtual keyboard capable of some "virtuoso" expression.

6. Portamento. Controls the time of transition from one pitch to another. Using it makes even this virtual keyboard capable of some "virtuoso" expression.

7. Filter Q. Controls the sharpness of the filters used in an instrument. Generally requires a complex instrument to take effect. For example, try this control with the "Weird Pad" instrument in the "Fantasy" bank.

8. Filter Freq. Controls the center frequency of the filters used in an instrument. Generally requires a complex instrument to take effect. For example, try this control with the "Weird Pad" instrument in the "Fantasy" bank.

9. Bandwidth. Controls the frequency bandwidth of the filters used in an instrument.

10. FM Gain. TODO. Haven't found a sound that exercises this control. Haven't looked all that hard yet.

11. Res. c. freq. TODO. Not sure how this differs from **Filter Freq.**. Haven't found a sound that exercises this control. Haven't looked all that hard yet.

12. Res. bw. TODO. Not sure how this differs from **Bandwidth**. Haven't found a sound that exercises this control. Haven't looked all that hard yet.

8 Effects

The *Yoshimi* Effects panel provides a number of special effects that can be applied to parts. Effects are, generally, blackboxes that transform audio signals in a specified way. More exactly, the only input data for an effect in *ZynAddSubFX* is an array of samples, which is read on line. The output is the transformed array of samples.

As described, effects have no information about anything else. For example, key presses are not recognized. Therefore, pressing a key does not initiate the LFO. Phase knobs will always be relative to a global LFO, which is only dependent on the system time.

Wetness determines the mix of the results of the effect and its input. This mix is made at the effects output. If an effect is wet, it means that nothing of the input signal is bypassing the effect. If it is dry, then the effect has no effect.

The Effects panel is shown in figure 2 "Yoshimi Main Screen, 1.3.8 and Above" on page 17.

Note that these effects have been incorporated into a separate guitar-effects project called *Rakkarrak* [11].

There are two types of effects: System effects and Insertion effects. The System effects apply to all parts and allows one to set the amount of effect that applies to each part. Also, it is possible to send the output of one system effect to another system effect. In the user interface this is shown as "source -<destination". For example: The **0 -<1** knob controls how much of the system effect 0 is sent to system effect 1.

Insertion effects are described in section 8.1.2 "Effects / Panel Types / Insertion" on page 105.

8.1 Effects / Panel Types

There are three variations of Effects sub-panels:

- **System Effects**.
- **Insertion Effects**.
- **Part/Instrument Effects**.

Here are the major elements of the main effects panel, which shows the System and Insertion effects tabs.

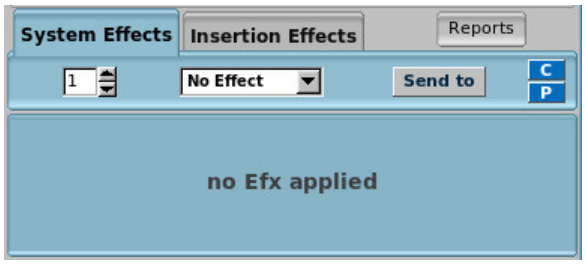

Figure 76: System Effects Dialog

1. **System Effects Tab**
2. **Effect Number**
3. **Effect Name**
4. **Send to**
5. **C**
6. **P**
7. **Effects Panel**
8. **Insertion Effects Tab**
9. **Reports**

1. System Effects Tab. System Effects Tab. The items in this tab are described in the next few paragraphs.

2. Effect Number. Effect Number. Up to 8 effects can be supported at one time by one part.

3. Effect Name. Effect Name.

Values: `No Effect*, Reverb, Echo, Chorus, Phaser, AlienWah, Distortion, EQ, DynFilter`

Figure 77: Effects Names

4. Send to. Effects Send To. Each knob controls how much of the system effect indicated by the left number is sent to the system effect indicated by the right number.

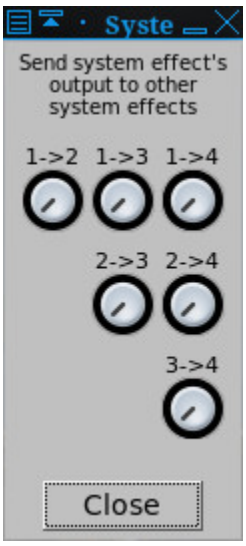

Figure 78: Effects, Send To

5. C. Copy-to-clipboard Dialog. We need to learn more about how this dialog gets populated.

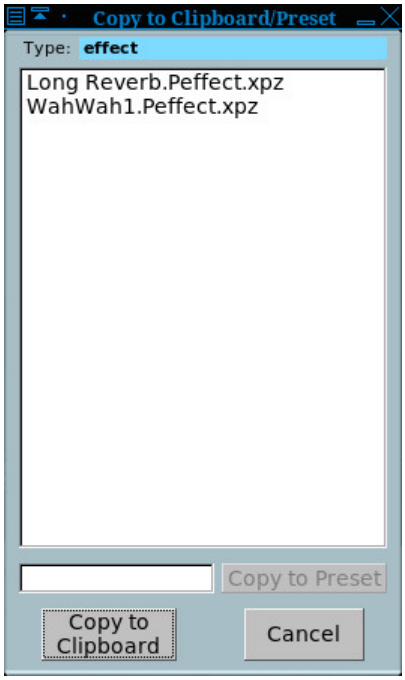

Figure 79: Effects / Copy To Clipboard

Note that, in recent versions of *Yoshimi*, the Type label at the top is not "effect", but "Peffect". What does this mean?

6. P. Paste-from-clipboard Dialog. We need to learn more about how this dialog gets populated.

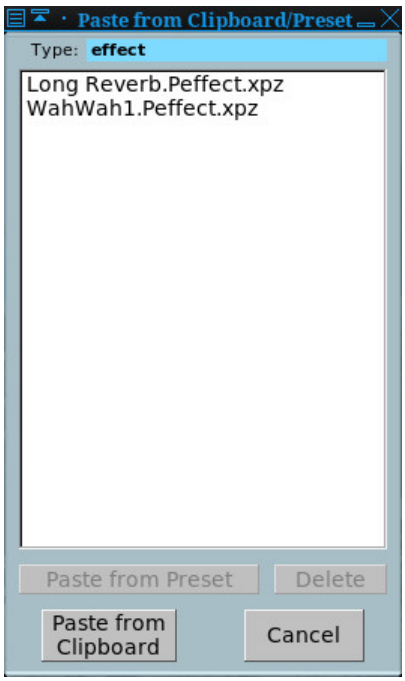

Figure 80: Effects / Paste From Clipboard

7. Effects Panel. Effects Panel. This area is filled by the controls for the selected effect.

8. Insertion Effects Tab. Insertion Effects Tab. The items in this tab are described below, in the 8.1.2 sub-section.

9. Reports. Effects Reports.

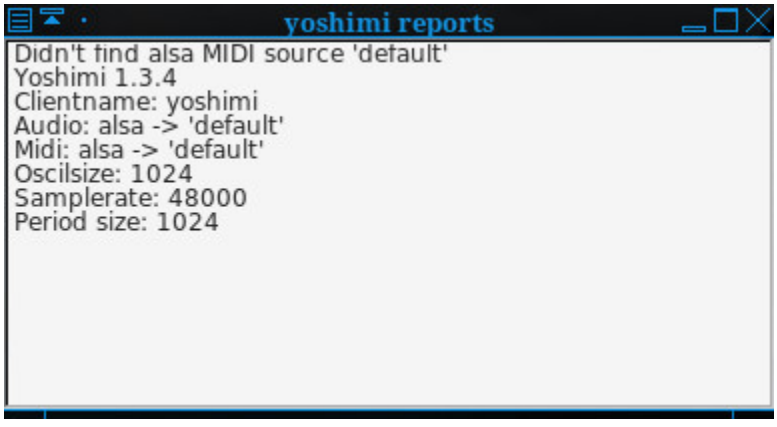

Figure 81: Effects / Reports

The next sub-sections show the variations on the effects panels, using the DynFilter effect as the subject effects panel.

8.1.1 Effects / Panel Types / System

The first variation appears when you enable an effect in the **System Effects** panel of the main *Yoshimi* dialog. It contains the standard controls for the given effect, plus the following interface items.

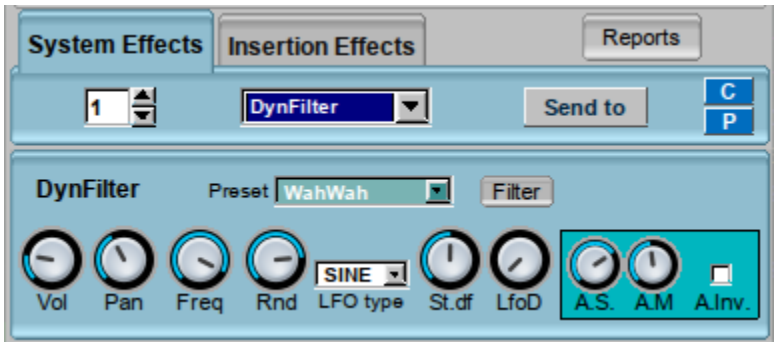

Figure 82: Sample System Effects Dialog

1. **Effect number**
2. **Effect selection**
3. **Effect Filter**
4. **C**
5. **P**

8.1.2 Effects / Panel Types / Insertion

The second effects variation appears when you enable an effect in the **Insertion Effects** panel of the main *Yoshimi* dialog. It contains the standard controls for the given effect, plus the following interface items.

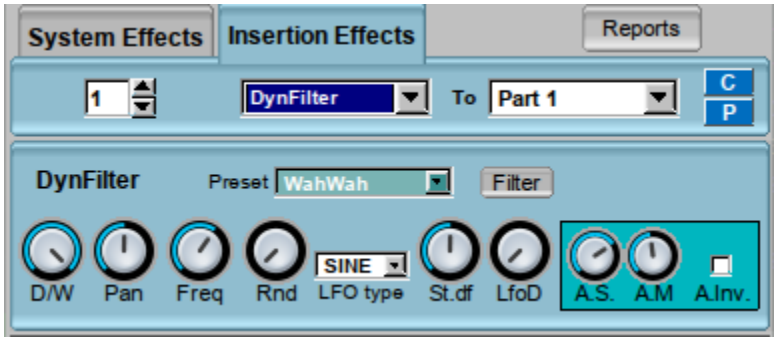

Figure 83: Sample Insertions Effects Dialog

1. **Effect number**
2. **Effect selection**
3. **To**
4. **C**
5. **P**

The insertion effects apply to one part or to the master output. One may use more than one insertion effect for one part or the master output. If one uses more than one effect, the effects with smaller indexes

will be applied first (eg. first insertion effect 0 occurs, then effect 1, and so on). If the part selected for insertion effect is "-1" then the effect will be *disabled*; if the part is "-2" the effect will be applied to Master Out.

1. To. Send the Effect To.

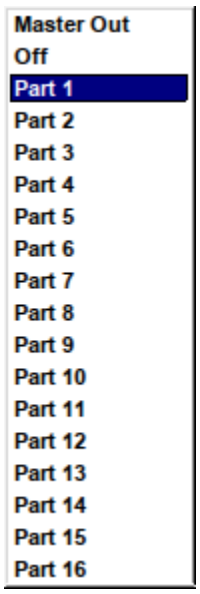

Figure 84: Part Selection Dropdown

8.1.3 Effects / Panel Types / Instrument

There is also a "part" or "instrument" effects window which is accessed by going to the main window, clicking the **Edit** button in the bottom panel to open the edit dialog, and then clicking the **Effects** button there. The part effects window has the same layout as System and Insertion effects; it is now almost identical to Insertion effects.

It contains the standard controls for the given effect, plus the following interface items.

1. **Effect number**
2. **Effect selection**
3. **To** (part-selection-dropdown.png)
4. **C**
5. **P**
6. **Bypass**
7. **Close**

"To" values:

Values: `Master Out, Off, Part 1, Part 2, ..., Part 16`

Figure 85: Sample Instrument Effects Dialog

Note the extra **Bypass** check-box, which presumably can be used to bypass this effects box in a given part.

8.2 Effects / None

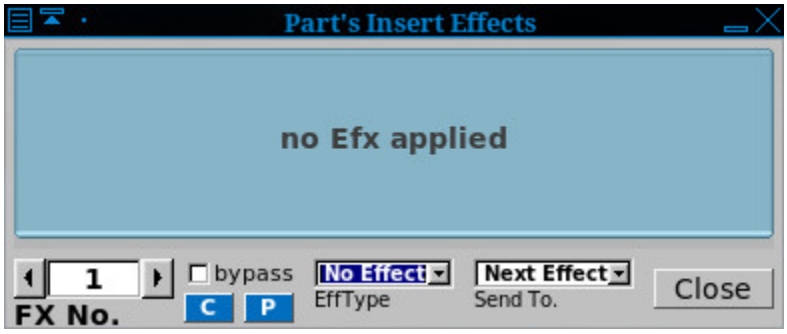

Figure 86: Effects Edit, None

This dialog form may be obsolete with the latest *Yoshimi*. It was captured around April of 2015. However, it seems that all but the **bypass** control and the **Close** button have been removed from the latest versions of *Yoshimi*. We have a lot of others in the same style that may need to be recaptured for use in new versions of the figures. For now, keep this difference in mind.

8.3 Effects / DynFilter

A dynamic filter is, as the name says, a filter which changes its parameters dynamically, dependent on the input and current time. In *ZynAddSubFX*, frequency is the only variable parameter. It can be used as an "envelope following filter" (sometimes referenced "Auto Wah" or simply "envelope filter").

8.3.1 Effects / DynFilter / Circuit

Though this filter might look a bit complicated, it is actually easy. We divide the parameters into two classes:

- Filter Parameters are the ones obtained when one clicks on Filter. They give the filter its basic settings.
- Effect Parameters are the other ones that control how the filter changes.

The filter basically works like this: The input signal is passed through a filter which dynamically changes its frequency. The frequency is an additive of:

- The filters base frequency.
- An LFO from the effect parameters.
- The "amplitude" of the input wave.

The amplitude of the input wave is not the current amplitude, but the so called "Root Mean Square (RMS)" value. This means that we build a mean on the current amplitude and the past values. How much the new amplitude takes influence is determined by the Amplitude Smoothness (see below).

RMS value plays an important role in the term *loudness*. A fully distorted signal can sound 20 db louder due to its higher RMS value. This filter takes this into account, depending on the smoothness.

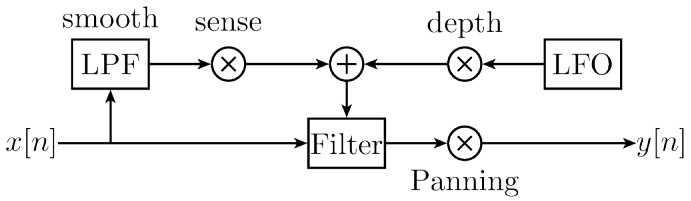

Figure 87: Dynamic Filter Circuit Diagram

8.3.2 Effects / DynFilter / User Interface

Figure 88: Effects Edit, DynFilter

This figure shows the Part/Instrument variation of the DynFilter sub-panel. The System/Insertion variation has the following elements.

1. **Preset**
2. **Filter**
3. **Vol** (system/insertion) or **D/W** (part/instrument)
4. **Pan**
5. **Freq**

6. **Rnd**
7. **LFO Type**
8. **St.df**
9. **LfoD**
10. **A.S.**
11. **A.M.**
12. **A.Inv.**

This figure shows an additional section, in the bottom panel of the effect's user-interface. See section 8.10.2 "Effects / Reverb / User Interface" on page 123, which describes these elements in more detail. However, it seems that all but the **bypass** control and the **Close** button have been removed from the latest versions of *Yoshimi*.

1. **FX No.**
2. **bypass**
3. **EffType**
4. **Send To**
5. **C**
6. **P**
7. **Close**

The four controls in the middle of the middle panel (**Freq**, **Rnd**, **LFO Type**, and **St.df**) control the LFO. Let's start with the user-interface elements present in the System/Insertion variation of this effect.

1. Preset. DynFilter Preset.

Figure 89: DynFilter Presets

Values: `WahWah, AutoWah, Sweep, VocalMorph1, VocalMorph2`

2. Filter. DynFilter Filter.

This small button brings up Filter Params stock sub-panel item. This stock user-interface item is shown and described in section 6.2.5 "Filter Parameters User Interface" on page 74.

3. Vol. DynFilter Volume.

Values: *0 to 127*

If the effect is used as a System effect, then this control appears.

4. D/W. DynFilter Dry/Wet Mix Setting.

Values: *0 to 127*

If the effect is used as an Insertion effect, then this control appears. "Dry" means the unprocessed signal and "wet" means the processed signal.

5. Pan. DynFilter Panning.

Values: *0 to 127*

After the input signal has passed through the filter, Pan can apply panning.

6. Freq. DynFilter LFO Frequency.

Values: *0 to 127*

7. Rnd. DynFilter LFO Randomness.

Values: *0 to 127*

8. LFO Type. DynFilter LFO Type.

9. St.df. DynFilter LFO. Left/right channel phase shift.

10. LfoD. DynFilter LFO Depth. This control is one that helps define the mix of the LFO and the amplitude.

11. A.S. DynFilter A.S. This control is one that helps define the mix of the LFO and the amplitude. A.S sets the Amplitude Sensing (i.e. how much influence the amplitude shall have).

12. A.M. DynFilter A.M. One of two knobs let one control the way how the RMS value of the amplitudes is measured. A.M sets the Amplitude Smoothness (this is described above). The higher one sets this value, the more slowly will the filter react.

13. A.Inv. DynFilter A.Inv. One of two knobs let one control the way how the RMS value of the amplitudes is measured. A.Inv., if set, negates the (absolute) RMS value. This will lower the filter frequency instead of increasing it. Note that this will not have much effect if the effects input is not very loud.

8.3.3 Effects / DynFilter / NRPN Values

Effects can be controlled via "non-registered parameter numbers", or NRPNs. This section will eventually (we hope) detail the NRPN values supported by the DynFilter effect.

For more information on the concept of NRPNs, see section 2.5.2 "Concepts / MIDI / NRPN" on page 27.

8.4 Effects / AlienWah

AlienWah is a nice effect done by Paul Nasca. It resembles a vocal morpher or wahwah a bit, but it is more strange. That's why he called it "AlienWah" The effect is a feedback delay with complex numbers.

The AlienWah effect is a special, dynamic formant filter (TODO: is this true?). Paul Nasca named it AlienWah because it sounded "a bit like wahwah, but more strange". The result of the filter is a sound varying between the vocals "Ahhhhh" (or "Uhhhhh") and "Eeeeee".

8.4.1 Effects / AlienWah / Circuit

No diagram, just a description of AlienWah.

Hint: Keep in mind that Effects that can be controlled by LFO can also be controlled arbitrarily: Set the LFO depth to zero and manipulate the phase knob (e.g. with NRPNs or maybe via OSC in the future).

The way that the filter moves between the two vocals is mainly described by an LFO. A bit easified, Paul Nasca has stated the formula (for i2 = -1 and R <1) as

$$fb = R * (cos() + i * sin())$$

$$yn = yn - delay * R * (cos() + i * sin()) + xn * (1 - R).$$

The input xn has the real part of the samples from the wavefile and the imaginary part is zero. The output of this effect is the real part of yn. is the phase.

8.4.2 Effects / AlienWah / User Interface

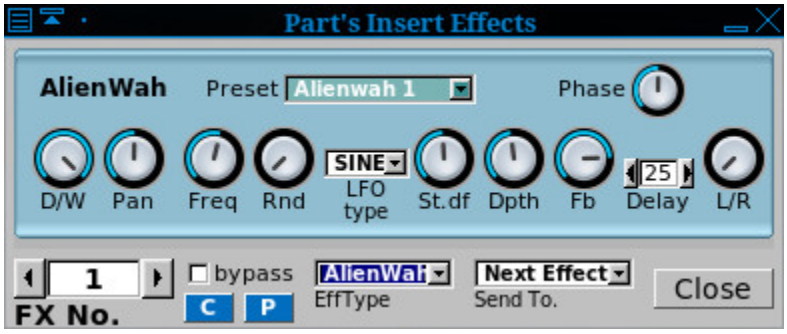

Figure 90: Effects Edit, AlienWah

1. **Preset**
2. **Phase**
3. **Vol** or **D/W**
4. **Pan**
5. **Freq**
6. **Rnd**
7. **LFO type**
8. **St.df.**
9. **Dpth**
10. **Fb.**
11. **Delay**
12. **L/R**

1. Preset. AlienWah Preset.

Values: `AlienWah 1, AlienWah 2, AlienWah 3, AlienWah 4`

2. Phase. The phase of the AlienWah. See in the above formula. This lets one set where the vocal is between "Ahhhhh" and "Eeeeee".

3. Vol. AlienWah Volume.

Values: `0 to 127`

The volume control is present is this effect is used as an insertion effect.

4. D/W. AlienWah Dry/Wet.

Values: `0 to 127`

The **Vol** control is replaced by this control if the effect is used as an Insertion effect.

5. Freq. LFO Frequency.

Values: `0 to 127`

Determines the LFOs frequency in relative units.

6. Rnd. LFO Amplitude Randomness.

Values: `0 to 127`

Part of the LFO definition.

7. LFO type. Set the LFO shape.

Values: `SINE, TRI`

Part of the LFO definition. Note that the LFO in other contexts has ramps and exponential shapes that are not present here.

8. St.df. AlienWah Left/Right Chanell Phase Difference.

Values: `0 to 127`

Part of the LFO definition. Sets the phase difference between LFO for left/right channels. **St.df** lets one determine how much left and right LFO are phase shifted. 64.0 means stereo, higher values increase the right LFO relatively to the left one.

9. Dpth. LFO depth.

Values: `0 to 127`

Dpth is a multiplier to the LFO. Thus, it determines the LFO's amplitude and its influence.

10. Delay. Amount of delay before the feedback.

Values: `1 to 100`

If this value is low, the sound is turned more into a "wah-wah"-effect.

11. Fb. AlienWah Feedback.

Values: `0 to 127`

TODO: What is the effect of the AlienWah feedback setting?

12. L/R. Determines how the left/right channels are routed to output:

- *Leftmost/0.* Left to left and right to right.
- *Middle/64.* Left+right to mono.
- *Rightmost/127.* Left to right, and right to left.

L/R applies crossover at the end of every stage. This is currently not implemented for the Analog Phaser.

13. Subtract. The output is inversed (inverted?)

8.4.3 Effects / AlienWah / NRPN Values

Effects can be controlled via "non-registered parameter numbers", or NRPNs. This section details the value supported by the AlienWah effect.

8.5 Effects / Chorus

In a chorus, many people sing together. Even if each of them sings at exactly the same frequency, all their voices usually sound different. We say they have a different timbre. Timbre is the way we perceive sound and makes us differ between different music instruments. This is, physically, achieved by varying both the amplitude envelope and the frequency spectrum. Multiple sounds with slightly different timbres make a sound more shimmering, or powerful. This is called the chorus effect.

The chorus effect can be achieved by multiple people singing together. In a concert, there are many instruments, resulting in the same effect. When making electronic music, we only have an input wave and need to generate these different timbres by ourselves. ZynAddSubFX therefore simply plays the sound, pitch modulated by an LFO, and adds this to the original sound. This explains the diagram below: The multiple pitches are generated by a delayed version of the input. This version is being pitched by an LFO. More detailed, this pitch is generated by varying the reading speed of the delayed sound; the variation amount is controlled by an LFO.

Related effects to Chorus are Flangers. Flangers can be described as Chorus with very short LFO delay and little LFO depth. One can imagine a flanger as two copies of a sound playing at almost the same time. This leads to interference, which can be clearly heared. It is popular to apply flangers to guitars, giving them more "character".

8.5.1 Effects / Chorus / Circuit

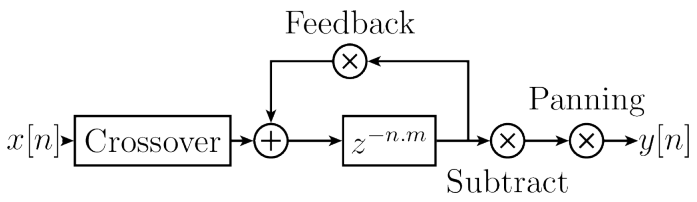

Figure 91: Chorus Circuit Diagram

First, crossover is applied.

The Freq, Rnd, LFO Type, St.df, Depth knobs control the LFO for the pitch. If the depth is set to zero, the pitch will not be changed at all.

Delay is the time that the delayed sound is delayed "on average". Note that the delay also depends on the current pitch.

After the correct element of the sound buffer is found using the LFO, the Fb knob lets one set how loud it shall be played. This is mostly redundant to the D/W knob, but we have not applied panning and substraction yet.

Next, the signal can be negated. If the Substract Checkbox is activated, the amplitude is multiplied by -1.

Finally, Pan lets one apply panning.

8.5.2 Effects / Chorus / User Interface

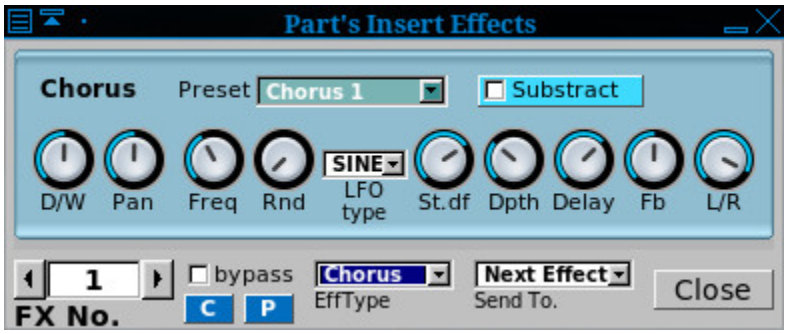

Figure 92: Effects Edit, Chorus

1. **Freq**
2. **Rnd**
3. **LFO type**
4. **St.df.**
5. **Dpth**
6. **Delay**
7. **Fb.**
8. **L/R**
9. **Subtract**

1. Freq. Chorus LFO Frequency.

2. Rnd. Chorus LFO randomness.

3. LFO type. Set the LFO shape.

4. St.df. The phase difference between LFO for left/right channels .

5. Dpth. Chorus LFO depth.

6. Delay. Delay of the chorus. If one uses low delays and LFO depths, this will result in a flanger effect.

7. Fb. Chorus Feedback.

8. L/R. How the left/right channels are routed to output:

1. leftmost. Left to left and right to right.
2. middle. Left+right to mono.
3. rightmost. Left to right, and right to left.

9. Subtract. The Chorus output is inversed (inverted?)

8.5.3 Effects / Chorus / NRPN Values

Effects can be controlled via "non-registered parameter numbers", or NRPNs. This section details the value supported by the Chorus effect.

8.6 Effects / Distortion

Distortion means, in general, altering a signal. Natural instruments usually produce sine-like waves. A wave is transformed in an unnatural way when distortion is used. The most distorted waves are usually pulse waves. It is typical for distortion to add overtones to a sound. Distortion often increases the power and the loudness of a signal, while the dB level is not increased. This is an important topic in the Loudness War.

As distortion increases loudness, distorted music can cause ear damage at lower volume levels. Thus, one might want to use it carefully. Distortion can happen in many situations when working with audio. Often, this is not wanted. In classical music, for example, distortion does not occur naturally. However, distortion can also be a wanted effect. It is typical for Rock guitars, but also present in electronic music, mostly in Dubstep and DrumNBass.

The basic components of distortion are mainly

- A preamplifier.
- The waveshaping function.
- Filters.

Preamplification changes the volume before the wave is shaped, and is indeed the amount of distortion. For example, if one clips a signal, the louder the input gets, the more distortion one will get. This can have different meanings for different types of distortions, as described below.

The filters are practical. A reason for using them afterwards is that distortion can lead to waves with undesired high frequency parts. Those can be filtered out using the LPF. A reason for using filters before applying is to achieve multiband distortion. ZynAddSubFX has no "real" multiband distortion by now, however.

The topic of types of distortion is completely discussed in the Oscillator Section.

(FIND THE REFERENCE)

Note that one can use the Oscillator editor in order to find out what the distortion effect does. Also note that while the Oscillator editors distortion is limited to some oscillators one can produce in the Oscillator editor, the distortion effect can be used on every wave that one can generate with ZynAddSubFX.

8.6.1 Effects / Distortion / Circuit

We explain the functionality in a diagram and list the components below.

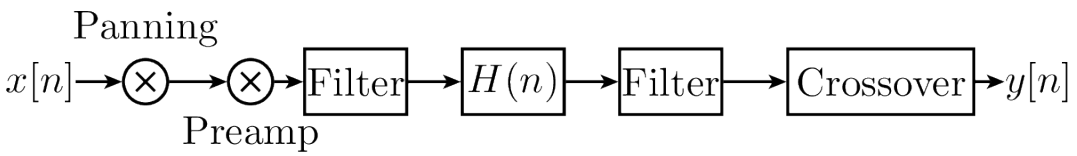

Figure 93: Distortion Circuit Diagram

Negation is the first thing to happen. If the Neg Checkbox is activated, the amplitude is multiplied by -1.

Panning is applied. Note, however, that one must activate the Stereo Checkbox, labelled St, before.

Pre-amplification is done next. The amount can be changed using the Drive nob. Indeed, this is the amount of distortion. For example, if one clips a signal, the louder the input gets, the more distortion one will get. This can have different meanings for different types of distortion, as described above.

HPF and LPF are filters with 2 poles. Whether they are used before or after the waveshape, depends on the checkbox labeled PF.

The next step is the wave shape. This defines how the wave is actually modified. The Type ComboBox lets one define how. We will discuss some types below.

After the wave shape, we scale the level again. This is called output amplification. One can change the value using the Level knob.

Crossover is the last step. This is controlled by the knob LR Mix and means that afterwards, a percentage of the left side is applied to the right side, and, synchronously, the other way round. It is a kind of interpolation between left and right. If one sets the LR Mix to 0.0, one will always have a stereo output.

8.6.2 Effects / Distortion / User Interface

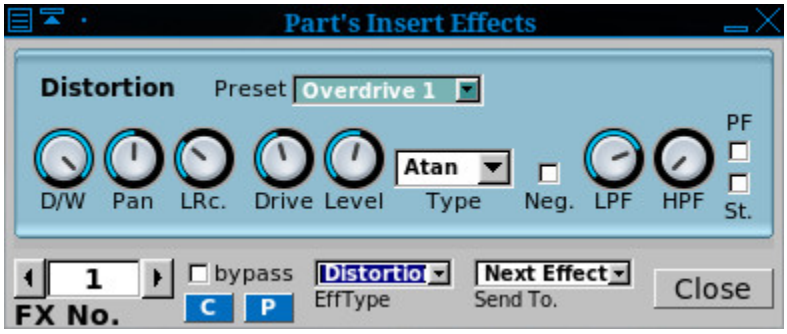

Figure 94: Effects Edit, Distortion

1. **Drive**
2. **Level**
3. **Type**
4. **Neg.**
5. **LPF**
6. **HPF**
7. **St.**

1. Drive. Set the amount of distortion.

2. Level. Amplify or reduce the signal after distortion.

3. Type. Set the function of the distortion (like arctangent, sine).

4. Neg. Negates the amplitude (invert the signal).

5. LPF. Low Pass Filter.

6. HPF. High Pass Filter.

7. St. Set the distortion mode (stereo or mono, checked is stereo).

8.6.3 Effects / Distortion / NRPN Values

Effects can be controlled via "non-registered parameter numbers", or NRPNs. This section details the value supported by the Distortion effect.

8.7 Effects / Echo

The echo effect, also known as delay effect, simulates the natural reflection of a sound. The listener can hear the sound multiple times, usually decreasing in volume. Echos can be useful to fill empty parts of songs with.

8.7.1 Effects / Echo / Circuit

The good circuit diagram is shown in an old printout we have, but the current version of the Echo description at http://zynaddsubfx.sourceforge.net/Doc/ shows a junk file. So Paul Nasca's description will have to suffice.

In ZynAddSubFX, the echo is basically implemented as the addition of the current sound and a delayed version of it. The delay is implemented as in the picture below. First, we add the delayed signal to the effect input. Then, they pass an LP1. This shall simulate the effect of dampening, which means that low and especially high frequencies get lost earlier over distance than middle frequencies do. Next, the sound is delayed, and then it will be output and added to the input.

The exact formula in the source code for the dampening effect is as follows:

$$Y(t) := (1 - d)X(t) + dY(t - 1)$$

where t be the time index for the input buffer, d be the dampening amount and X,Y be the input, respective the output of the dampening. This solves to

$$Y(z) = Z(Y(t)) = (1 - d)X(z) + dY(z)z - 1H(z) := Y(z)X(z) = 1 - d1 - dz - 1$$

which is used in $Y(z) = H(z)X(z)$. So H(z) is indeed a filter, and by looking at it, we see that it is an LP1. Note that infinite looping for d=1 is impossible.

8.7.2 Effects / Echo / User Interface

Figure 95: Effects Edit, Echo

TODO (yoshimi): Pan lets one apply panning of the input.

1. **Delay**
2. **LRdl.**
3. **LRc.**
4. **Fb.**
5. **Damp**

1. Delay. The delay time of one echo.

2. LRdl. Left-Right-Delay. The delay between left/right channels. If it is set to the middle, then both sides are delayed equally. If not, then the left echo comes earlier and the right echo comes (the same amount) later than the average echo; or the other way round. Set the knob to 0 to hear on the right first.

3. LRc. Echo Crossover. The "crossing" between left/right channels.

4. Fb. Echo feedback. Feedback describes how much of the delay is added back to the input. Set Fb. to the maximum to hear an infinite echo, or to the minimum to just hear a single repeat.

5. Damp. Echo damping. How high frequencies are damped in the Echo effect. The Damp value lets the LP1 reject higher frequencies earlier if increased.

8.7.3 Effects / Echo / NRPN Values

An equalizer is a filter effect that applies different volume to different frequencies of the input signal. This can, for example, be used to "filter out" unwanted frequencies. ZynAddSubFXs implementations follow the "Cookbook formulae for audio EQ" by Robert Bristow-Johnson. (NEED A REFERENCE)

Effects can be controlled via "non-registered parameter numbers", or NRPNs. This section details the value supported by the Echo effect.

TODO TODO TODO TODO

8.8 Effects / EQ

EQ is a parametric equalizer. On the equalizer graph there are 3 white vertical bars for 100Hz, 1kHz, 10kHz.

8.8.1 Effects / EQ / Circuit

8.8.2 Effects / EQ / User Interface

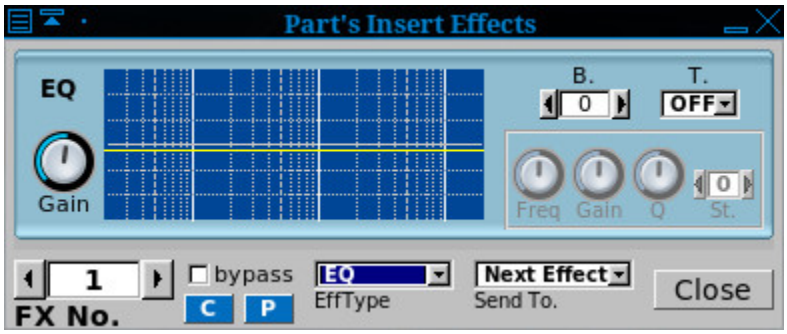

Figure 96: Effects Edit, EQ

We describe all parts of the GUI here. The term passband (or often just "band") refers to the amount of frequencies which are not significantly attenuated by the filter.

1. **Gain**
2. **B**
3. **T**
4. **Freq**
5. **Gain**
6. **Q**
7. **St**

Global:

1. Gain. Amplifies or reduce the signal that passes through EQ.

2. B. Set the current frequency band (or filter). B lets one choose the passband number. Multiple passbands define one filter. This is important if one wants multiple filters to be called after each other. Note that filters are commutative.

Bands:

3. T. Set the type of the filter.

4. Freq. The frequency of the filter. Freq describes the frequencies where the filter has its poles. For some filters, this is called the "cutoff" frequency. Note, however, that a bandpass filter has two cutoff frequencies.

5. Gain. The gain of the filter. Gain is only active for some filters and sets the amount of a special peak these filters have. Note that for those filters, using the predefined gain makes them effectless.

6. Q. The Q (resonance, or bandwidth) of the filter. Resonance lets one describe a peak at the given frequency for filters with 2 poles. This can be compared to real physical objects that have more gain at their resonance frequency.

7. St. Number of additional times the filter will be applied (in order to do very steep roll-off - eg. 48 dB/octave). St. lets one define multiple filter stages. This is equivalent to having multiple copies of the same filter in sequence.

8.8.3 Effects / EQ / NRPN Values

Effects can be controlled via "non-registered parameter numbers", or NRPNs. This section details the value supported by the EQ effect.

8.9 Effects / Phaser

The Phaser is a special dynamic filter. The result is a sweeping sound, which is often used on instruments with a large frequency band, like guitars or strings. This makes it typical for genres like rock or funk, where it is often modulated with a pedal, but also for giving strings a warm, relaxing character.

8.9.1 Effects / Phaser / Circuit

No circuit diagram, just a picture of the results of the phaser effect in the form of a spectrogram.

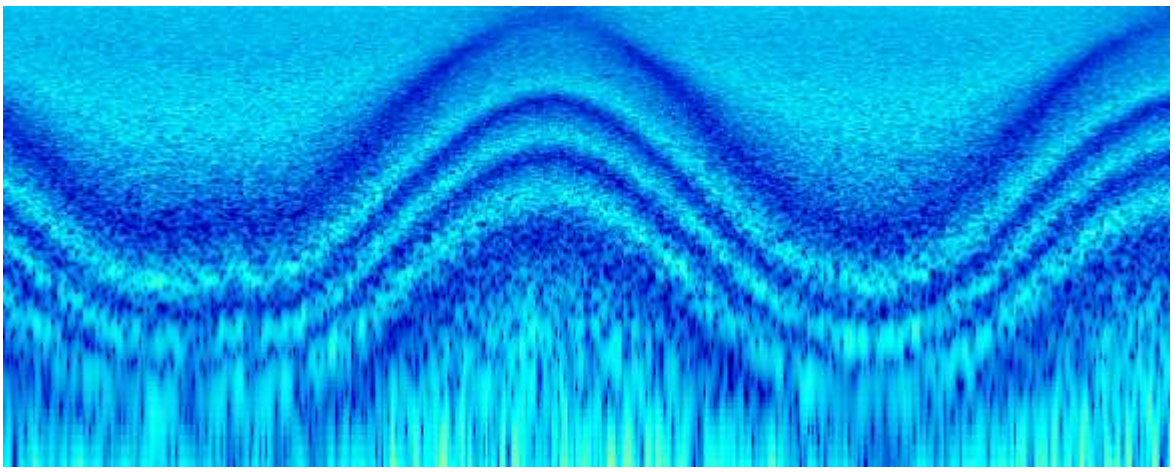

Figure 97: Phaser Circuit Diagram

The audio signal is split into two paths. One path remains unchanged. The other one is sent to a delay line. The delay time (the so called phase) is made dependent on the frequency. Therefore, an all-pass filter is applied to the signal, which preserves the amplitude, but determines the delay time. In the end, both paths are added.

The following picture describes how this works on white noise. Light blue signalises that the frequency is not present at the current time, and dark blue signalises the opposite. The dark blue peaks appear if the delay time is very short, because then, the second path almost equals the first one, which results in duplication of the signal. If the delay line is very long, then it is – in the case of white noise – totally at random whether the delayed signal currently duplicates the unchanged path, or whether it cancels it out to zero. This random effect results in white noise between the clear blue structures.

ZynAddSubFX offers different types of phasers:

- Analog and "normal" phasers. Analog phasers are more complicated. They sound punchier, while normal phasers sound more fluently. However, analog filters usually need more filter stages to reach a characteristic sound.
- Sine and triangle filters. Note that an analog triangle filter with many poles is a barber pole filter and can be used to generate Shepard Tones, i.e. tones that seem to increase or decrease with time, but do not really.
- The LFO function can be squared. This converts the triangle wave into a hyper sine wave. The sine squared is simply a faster sine wave.

TODO: Barber is deactivated, since PLFOtype is only 0 or 1?

For the normal phaser:

1. First, the LFO is generated. There are 4 controls (Freq,Rnd,LFO tpye,St.df) that define the LFO.
2. Phase and Depth are applied afterwards in the usual way (TODO: I dont understand the code here for the normal phase). For the analog phaser, Phase is not implemented, yet.
3. If hyp is being set, then the LFO function is being squared.
4. Next, the input is being used.
5. Analog decides whether the phaser is analog or "normal".
6. First, Pan applies panning to the original input in every loop.

7. Next, barber pole phasing is being applied (Analog only).
8. Fb applies feedback. The last sound buffer element is (after phasing) multiplied by this value and then added to the current one. For normal filter, the value is added before, for analog after the first phasing stage.
9. Now, Stages phasing stages are being applied. dist sets the distortion for when applying the phasing stages. This has only effect for analog phasers.
10. The feedback is performed next.
11. In the end, Substract inverts the signal, multiplying it by -1.

8.9.2 Effects / Phaser / User Interface

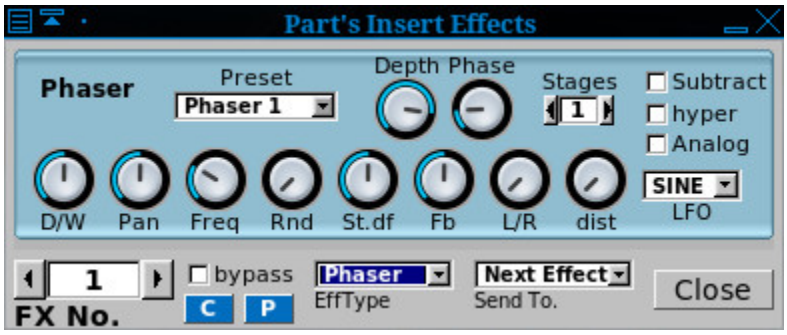

Figure 98: Effects Edit, Phaser

TODO. Include the item-paragraphs for each GUI element.

1. **Preset.**
2. **Depth**
3. **Phase**
4. **Stages**
5. **Subtract**
6. **hyper**
7. **Analog**
8. **D/W**
9. **Pan**
10. **Freq**
11. **Rnd**
12. **St.df**
13. **Fb**
14. **L/R**
15. **dist**
16. **LFO**

The extra fields that are shown if the effect is an insertion effect are not shown. They are described in section TODO TODO TODO.

8. Preset. Phaser Presets.

TODO: need a diagram of the dropdown

Values: `Phaser 1, ...` TODO.

9. Depth. Phaser Depth. Phaser LFO Depth?

Values: `TODO`

10. Phase. Phaser Phase.

Values: `TODO`

11. Stages. Phaser Stages.

Values: `TODO`

12. Subtract. Phaser Subtract.

Values: `Off*, On`

13. hyper. Phaser Hyper.

Values: `Off*, On`

14. Analog. Phaser Analog.

Values: `Off*, On`

15. D/W. Phaser Dry/Wet.

16. Pan. Phaser Panning.

17. Freq. Phaser Freq.

18. Rnd. Phaser Randomness.

19. St.df. The phase difference between LFO for left/right channels.

20. Fb. Phaser Feedback.

21. L/R. L/R. How the left/right channels are routed to output:

1. leftmost. Left to left and right to right.
2. middle. Left+right to mono.
3. rightmost. Left to right, and right to left.

22. dist. Phaser Distortion? TODO

23. LFO. Phaser LFO Type.

8.9.3 Effects / Phaser / NRPN Values

Effects can be controlled via "non-registered parameter numbers", or NRPNs. This section details the value supported by the Phaser effect.

8.10 Effects / Reverb

A Reverberation actually expresses the effect of many echoes being played at the same time. This can happen in an enclosed room, where the sound can be reflected in different angles. Also, in nature, thunders approximate reverbs, because the sound is reflected in many different ways, arriving at the listener at different times.

In music, reverbs are popular in many ways. Reverbs with large room size can be used to emulate sounds like in live concerts. This is useful for voices, pads, and hand claps. A small room size can simulate the sound board of string instruments, like guitars or pianos.

8.10.1 Effects / Reverb / Circuit

As mentioned, a reverb consists of permanent echo. The reverb in ZynAddSubFX is more complex than the echo. After the delaying, comb filters and then allpass filters are being applied. These make the resulting sound more realistic. The parameters for these filters depend on the roomsize. For details, consider the information about Freeverb.

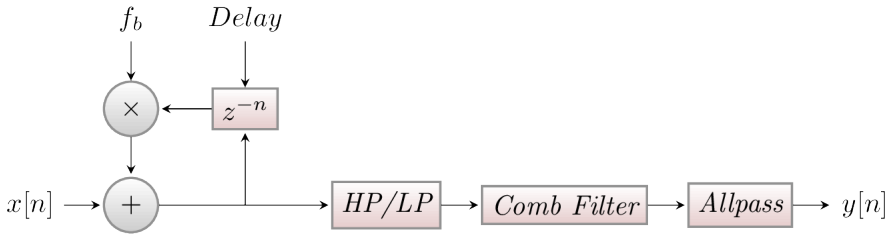

Figure 99: Reverb Circuit Diagram

8.10.2 Effects / Reverb / User Interface

The user-interface for the Reverb effect depends on whether it is used as a System effect or an Insertion effect. Observr figure 100 "Effects Edit, Reverb" on page 123, where the Insertion mode is shown. In the System mode, only the light-blue portion of the user-interface appears.

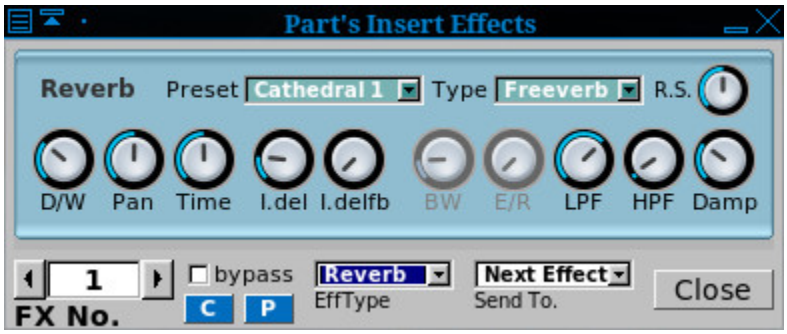

Figure 100: Effects Edit, Reverb

1. **Preset**
2. **Type**
3. **R.S.**
4. **D/W**
5. **Pan**
6. **Time**
7. **I.del**
8. **I.delfb**
9. **BW**
10. **E/R**
11. **LPF**
12. **HPF**
13. **Damp**

There is a fourth type we have screen captures for, but we can't seem to navigate to them now! Are these now out-of-date screen captures?

1. **FX No.**
2. **bypass**
3. **EffType**
4. **Send To**
5. **C**
6. **P**
7. **Close**

1. Preset. Reverb Preset.

Figure 101: Reverb Preset Dropdown

Values: `Cathedral 1, Cathedral 2, Cathedral 3, Half 1, Half 2, Room 1, Room 2, Basement, Tunnel, Echoed 1, Echoed 2, Very Long 1, Very Long 2`

2. Type. Reverb Type. The combobox lets one select a reverb type.

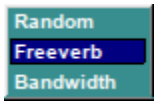

Figure 102: Reverb Type Dropdown

- Freeverb is a preset. It was proposed by Jezar at Dreampoint.
- Bandwidth has the same parameters for the comb and allpass filters, but it applies a unison before the LPF/HPF. The unisons bandwidth can be set using BW.
- Random chooses a random layout for comb and allpass each time the type or the roomsize is being changed.

Values: `Random, Freeverb, Bandwidth`

3. R.S. Reverb Room Size. The room size defines parameters only for the comb and allpass filters.

4. D/W. Reverb Dry/Wet Setting. This setting controls much of the original signal is mixed with the reverb effect.

5. Pan. Reverb Panning. Pan lets one apply panning. This is the last process to happen.

6. Time. Reverb Time. Set the duration of late reverb. Time controls how long the whole reverb shall take, including how slow the volume is decreased.

7. I.del. Reverb Initial Delay. The initial delay (I.del) is the time which the sounds need at least to return to the user.

8. I.delfb. Reverb Initial Delay Feedback. Sets the initial delay feedback. The initial delay feedback (I.delfb) says how much of the delayed sound is added to the input. It is not recommended to use this setting together with low initial delays).

9. BW. Reverb Bandwidth.

10. E/R. Reverb E/R. Echo Reflection? TODO!

11. LPF. Reverb Lowpass Filter. This filter is applied before the comb filters.

12. HPF. Reverb Highpass Filter. This filter is applied before the comb filters.

13. Damp. Reverb Damp. Damp determines how high frequencies are damped during the reverberation. The dampening control (Damp) currently only allows to damp low frequencies. Its parameters are used by the comb and allpass filters.

14. FX No. Reverb FX Number.

Values: `1 to 8?`

15. bypass. Reverb FX Bypass.

Values: `Off*, On`

16. EffType. Reverb Effect Type.

Values: `Reverb, EQ, Echo, etc.  TODO`

17. Send To. Reverb Send To.

Values: `Next Effect, ...  TODO`

18. C. Reverb Copy.

19. P. Reverb Paste.

20. Close. Close Window.

8.10.3 Effects / Reverb / NRPN Values

Effects can be controlled via "non-registered parameter numbers", or NRPNs. This section details the values supported by the Reverb effect.

TODO: detail the values supported by the Reverb effect.

9 Bottom Panel

9.1 Bottom Panel Controls

The *Yoshimi* bottom panel provides quick access to some major features of the application. The bottom panel is shown in figure 2 "Yoshimi Main Screen, 1.3.8 and Above" on page 17.

Here are the major elements of the bottom panel.

1. **Part**

2. **of**
3. **Instrument Name**
4. **Edit** (Instrument Edit Button)
5. **Midi**
6. **Mode**
7. **Enabled**
8. **Portamento**
9. **Velocity Sens**
10. **Velocity Offset**
11. **Pan**
12. **Pan Reset Button**
13. **Volume**
14. **Controllers**
15. **Minimum Note**
16. **Maximum Note**
17. **m**
18. **R**
19. **M**
20. **Key Shift**
21. **Key Limit**
22. **System Effect Sends 1**
23. **System Effect Sends 2**
24. **System Effect Sends 3**
25. **System Effect Sends 4**
26. **Sound Meter**

1. Part. Part Number.

Values: `1 to 16; 1 to 32; 1 to 64`

Show and set current part. The maximum number of values depends on the **Part of** selection.

2. of. Maximum Number of Parts.

Values: `16*, 32, 64`

Yoshimi now has up to 64 parts in blocks of 16. One can now decide how many one wants to have available using this user-interface item. By default these are wrapped around the normal MIDI channels, so that parts 1, 17, 33, and 49 all respond to channel 1 messages. This was originally implemented for Vector Control, working with up to four sounds on a channel (similar to the Yamaha SY hardware series).

However, these additional parts have other less obvious uses. One of these is getting far more than 16 completely independent tracks addressed by just the 16 channels. Most tunes run with instruments having a relatively narrow pitch range, and this is what we can make use of.

As an example, in *Yoshimi*'s main window select 64 parts, then on part 1 set (say) 'Steel Bass' and maximum note as 52 (E). Next select part 17 and enable it (easiest to use the mixer panel for this) set 'Tunnel Piano', the *minimum* note as 53 and maximum as 71 (B). Finally, enable part 33, set 'Rushes', and set its minimum note as 72, but key shift down an octave. With a 61 note keyboard that gives one quite a useful working range, on just one channel.

However, the idea really comes into its own with a sequencer like Rosegarden where one can record multiple parts over the full MIDI range and track them to the same channel. Also, in Rosegarden the

parts can be separately named, and identified as 'Bass' and 'Treble' in the notation editor. This setup makes it very convenient for those wanting a more formal musical layout.

So, with very little effort, one can now have 48 tracks playing at once! Ummm, one does need a decent processor though :) Yes, one could run more instances of Yoshimi on different MIDI ports, but where's the fun in that?

Another possibility is obtaining very smooth transitions between different sounds on the same channel. If one uses program change to do this, that part has to be muted, and there is a variable time lag (while the new part is loaded) before one can play any more notes on that channel. However, with 32 and 64 parts one can actually overlap notes with different instruments on a playing channel.

This setup is accomplished by pre-loading the wanted instruments, then switching channel numbers. If (via the direct part NRPN) one adds 16 to a sounding part's channel number, it will then only respond to Note Off events. To bring it back into operation simply restore the original channel number. An example:

1. Enable 32 parts.
2. Load 'Simple Chimes' into part 0 (part 1 in the GUI).
3. Load 'Silver Bell' into part 16 (part 17 in the GUI).
4. In your sequencer, via direct part NRPN set part 16 to channel 16. This will now be 'whited out' in the GUI.
5. Record some notes to channel 0 (1 in human-readable terms).
6. In the sequencer, just before the first note that one wants to sound as 'Silver Bell', insert two direct part NRPNs, with one to set part 0 to channel 16, and the other to set part 16 to channel 0.

Now, when played through, the last 'Simple Chimes' note will have its full release and reverb tail, blending into the first 'Silver Bell' note. To go back to using 'Simple Chimes' just reverse the NRPNs. The only time this gets complicated is if the new note is exactly the same pitch as the old one, in which case the NRPNs need to be between the old note-off and the new note-on.

By default, all the upper parts (numbers greater than 16) are mapped to the same MIDI channel numbers as the lowest ones, but have independent voice and parameter settings. They cannot normally receive independent note or control messages. However, vector control will intelligently work with however many one has set, as will all the NRPN direct part controls. See section 16.2 "Vector / Vector Control" on page 184.

This item is a fairly new feature of *Yoshimi* (as of version 1.3.5).

3. Instrument Name. Instrument Name. Left-click to open the Bank window. Right-click to change the name of the current instrument. If one changes the name of the instrument, be sure to select **Menu / Instrument / Save Instrument** to preserve that change.

The name now has color-coding to indicate the instrument's use of ADDsynth, SUBsynth, or PADsynth. One can see the "red" color for ADDsynth in the figure for the bottom panel. "Blue" would indicate SUBsynth, and "green" would indicate PADsynth.

4. Edit. Instrument Edit button. This button brings up the instrument-edit dialog shown in figure 104 "Instrument Edit Dialog" on page 132.

This dialog provides a very broad overview of the instrument, and provides access to far more detailed dialogs to edit the instrument. This dialog is explained in detail in section 9.3 "Bottom Panel Instrument Edit" on page 132.

5. Midi. MIDI Channel.

Values: `1 to 16`

6. Mode. Mode. Poly. Sets the mode (polyphonic/monophonic/legato). In polyphonic mode, multiple simultaneous notes are supported. (How many at maximum?). In monophonic mode, only one note is supported. In legato mode, the sound flows smoothly from note to note without any breaks.

Values: `Poly, Mono, Legato`

7. Enabled. Enable the part. If the Part is disabled it doesn't use CPU time.

Values: `Off*, On`

8. Portamento. Enable/disable the portamento. One can set the duration and other parameters by opening the Controllers window.

Values: `Off*, On`

9. Velocity Sens. Velocity Sensing Function.

Values: `0 to 127, 64*`

10. Velocity Offset. Velocity Offset.

Values: `0 to 127, 64*`

11. Pan. Pan.

Values: `0 to 127, 64*`

12. Pan (reset). Reset Pan to Middle (64).

13. Volume. Instrument Volume.

Values: `0 to 127, 96*`

The default volume for ADD parts (overall) and SUB parts is 96; the default volume for SUB parts is 90; the ADD voice volumen is 100; and effects volumes vary heavily with the effect.

14. Minimum Note. Minimum note the part receives.

Values: `0* to 127`

15. Maximum Note. Maximum note the part receives.

Values: `0 to 127*`

16. m. Minimum Note Capture Button.

Set minimum note to last note played.

17. R. Minimum and Maximum Note Reset Button.

Reset the minimum key to 0 and the maximum key to 127.

18. M. Maximum Note Capture Button. Sets the maximum note to the last pressed key.

19. Key Shift. Key Shift. This value is like the master **Key Shift** in the top panel, but it applies only to the current part active in the bottom panel. In recent versions of *Yoshimi*, it has been extended to a larger semitone range. Also note that the key shift can be set via the user-interface, the command-line, or by MIDI NRPN commands. With NRPN, this part shift can be set by direct part control or by channel number.

Values: `-36 to 36, 0*`

20. Key Limit. Maximum keys to be allocated for this part.

Values: `0 to 60, 20*`

21. System Effect Sends 1, 2, 3, and 4.

TODO: Describe how these sends work.

Values: `0 to 127*`

22. Sound Meter. VU Meter. Sound Meter.

This discussion of "Audio Output and Levels" comes from `Output Levels.txt`.

At the bottom of the main window there is a pair of horizontal grids representing a bargraph type display. The upper one is for the left hand channel and the lower one for the right hand one. The grid divisions each represent 1 dB, and the brighter divisions are therefore 5 dB. The thicker bright divisions therefore being 10 dB. The overall scale range is -48 dB to 0 dB.

As the output level rises pale blue strips will light up in these grids. These fast responding bars are the peak levels and should never be allowed to go above 0 dB, otherwise the output is likely to be clipped and distorted. There is also a pair of boxes on the end of these grids which will show the highest peak level seen. If clipping has happened the box background will change from black to red. To clear the clip and peak level indication, click on this area.

As well as the peak level, the display shows a much slower responding RMS level, as a yellow line on top of the blue bar. This gives and indication of the apparent accoustic power.

If one opens the panel window one will see vertical bargraphs for each individual part. On these, the faint bars are 5dB steps and the bright ones 10dB. The peak level isn't shown numerically, but if one exceeds 0dB a thick red line will appear at the top of the bargraph. This is also cleared from the box in the main window.

9.1.0.1 Tip: Using the VU Meter

The VU meter topic is very interesting, because one of the problems is a tendency to overdrive it by way of sustain pedal. At the last test, it showed up in the output before it showed up in the VU meter, so the VU meter should help a lot in analysis.

One way to avoid overdrive is to keep polyphony to 20 on each patch (two or three patches per *Yoshimi* instance, with two or three *Yoshimi* simultaneous instances depending on the patch). Another item which helps a lot is compression (for example, the Calf multiband compressor is amazingly good.

9.2 Bottom Panel / Controllers

Figure 103: Controllers Dialog

1. **Exp MWh**

2. **ModWh**
3. **Exp BW**
4. **BwDepth**
5. **PanWdth**
6. **FltQ**
7. **FitCut**
8. **Vol Rng**
9. **PWheelB.Rng**
10. **Expr**
11. **FMamp**
12. **Vol**
13. **Sustain**
14. **Resonance** (section)
15. **PortaMento** (section)
16. **Reset all controllers**
17. **Close**

1. Exp MWh. Exponential Modulation Wheel. Changes the modulation scale to exponential.

Values: `Off*, On`

2. ModWh. Modulation Wheel Depth.

Values: `0 to 127, 80*`

3. Exp BW. Exponential Bandwidth Controller. Changes the bandwidth scale to exponential.

Values: `Off*, On`

4. BwDepth. Bandwidth Depth.

Values: `0 to 127, 64*`

5. Exp BW. Exponental Bandwidth. Changes the bandwidth scale to exponential.

Values: `0 to 127, 64*`

6. PanDpth. Panning Depth.

Values: `0 to 64*`

7. FltQ. Filter Q (resonance) Depth.

Values: `0 to 127, 64*`

8. FltCut. Filter Cutoff Frequency Depth.

Values: `0 to 127, 64*`

9. Vol Rng. Volume Range.

Values: `64 to 127, 64*`

10. PWheelB.Rng. Pitch Wheel Bend Range (cents). 100 cents = 1 halftone.

Values: `-6400 to 6400, 200*`

11. Expr. Expression Enable. Enable/disable expression.

Values: `Off, On*`

12. FMamp. FM Amplitude Enable. Enable/disable receiving Modulation Amplitude controller (76).

Values: `Off, On*`

13. Vol. Volume Enable.

Values: `Off, On*`

Enable/disable receiving volume controller. Sensitivity to MIDI volume change (CC7) is now variable in 'Controllers' in the same way as pan width etc. The nnumeric range is 64 to 127; the default at 96 gives the same sensitivity as before at -12dB relative to the GUI controls. 127 gives 0dB and 64 gives -26dB

14. Sustain. Sustain Pedal Enable. Enable/disable sustain pedal.

Values: `Off, On*`

15. Reset all controllers. Reset All Controllers.

16. Close. Close Window.

9.2.1 Bottom Panel / Controllers / Resonance

1. CFdepth. Resonance Center Frequency Depth, Center Frequency Controller Depth.

Values: `0 to 127, 64*`

2. BWdepth. Resonance Bandwidth Depth, Resonance Bandwidth Controller Depth.

Values: `0 to 127, 64*`

9.2.2 Bottom Panel / Controllers / Portamento

1. Rcv. Portamento Receive, Receive Portamento Controllers. Determines if the part receives Portamento On/Off (65) controller.

Values: `Off, On*`

2. Proprt. Portamento Proportional, Enable Proportional Portamento (over fixed portamento).

Values: `Off*, On`

3. time. Portamento time. The duration of the portamento.

Values: `0 to 127, 64*`

4. t.dn/up. Portamento Time Stretch (up/down).

Values: `0 to 127, 64*`

5. threshx100 cnt. Threshold of the Portamento. The minimum or maximum difference of notes in order to do the portamento (x 100 cents). It represents the minimum or the maximum number of halftones (or hundred cents) required to start the portamento. The difference is computed between the last note and current note. The threshold refers to the frequencies and *not* to MIDI notes (one should consider this if one uses microtonal scales).

Values: `0 to 127, 3*`

6. th.type. Threshold Type (min/max). Checked means that the portamento activates when the difference of frequencies is above the threshold ("thresh"); not checked is for below the threshold.

Values: `Off, On*`

7. Propt. Proportional Portamento. If set, the portamento is proportional to ratio of frequencies.

Values: `Off, On*`

8. Prp.Rate. Distance required to double change from nonproportional portamento time. The ratio needed to double the time of portamento.

Values: `0 to 127, 80*`, requires **Proprt.** = `On`

9. Prp.Depth. The difference from nonproportional portamento.

Values: `0 to 127, 90*`, requires **Proprt.** = `On`

9.3 Bottom Panel Instrument Edit

The main instrument-editing dialog is relatively simple, and provides for editing information that identifies the instrument, and buttons to access the more complex dialogs of the ADDsynth, SUBsynth, PADsynth, Kit Edit, and Effects components.

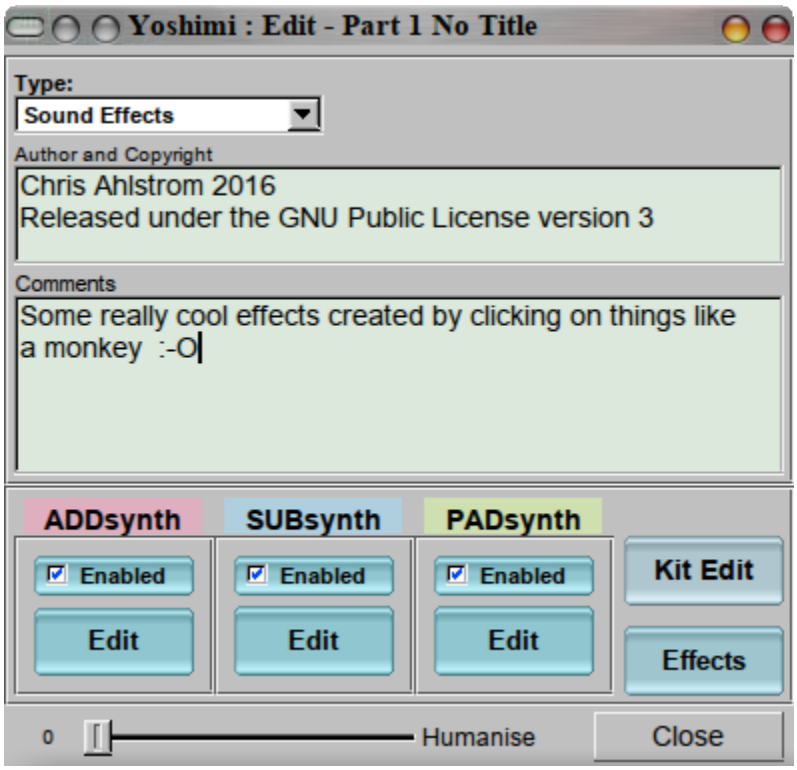

Figure 104: Instrument Edit Dialog

This dialog provides a very broad overview of the instrument, and provides access to far more detailed dialogs to edit the instrument. This dialog is called up by the **Edit** button on the bottom panel of the main *Yoshimi* main screen.

1. **Type**
2. **Author and Copyright**
3. **Comments**
4. **ADDsynth**
    1. **Enabled**
    2. **Edit**
5. **SUBsynth**

    1. **Enabled**
    2. **Edit**
6. **PADsynth**
    1. **Enabled**
    2. **Edit**
7. **Kit Edit**
8. **Effects**
9. **Rnd. Det.**, now replaced by a **Humanise** slider.
10. **Close**

The ADDsynth, SUBsynth, PADsynth, Kit Edit, and Effects dialogs are detailed in separated sections, as they are all very complex dialogs with many sub-dialogs.

1. Type. Instrument Type. Instrument Category.

This dropdown dialog allows one to tag the type of instrument, to indicate what category of instruments it fits into. The following figure shows the types.

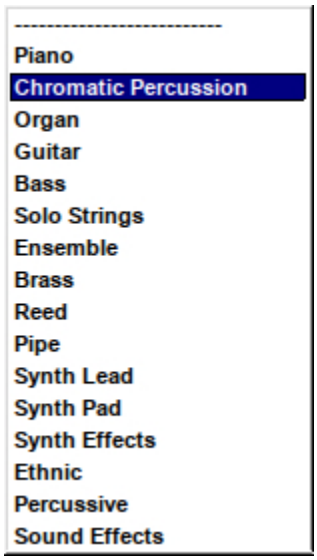

Figure 105: Instrument Type Drop-down List

Values: `Piano, Chromatic Percussion, Organ, Guitar, Bass, Solo Strings, Ensemble, Brass, Reed, Pipe, Synth Lead, Synth Pad, Synth Effects, Ethnic, Percussive, Sound Effects`

2. Author and Copyright. This field provides space for identifying the author, copyright, and license for the part.

3. Comments. Allows free-form comments and notes to be entered.

4. ADDsynth.

1. **Enabled**. Enables this synth type to be used in the part/instrument. When enabled, its marker color, red, is shown.
2. **Edit**. Brings up the editing dialog presented in figure 106 "ADDsynth Edit/Global Dialog" on page 135. There one will find a full discussion of that dialog.

5. SUBsynth.

1. **Enabled**. Enables this synth type to be used in the part/instrument. When enabled, its marker color, blue, is shown.
2. **Edit**. Brings up the editing dialog presented in figure 141 "SUBsynth Edit Dialog" on page 168. There one will find a full discussion of that dialog.

6. PADsynth.

1. **Enabled**. Enables this synth type to be used in the part/instrument. When enabled, its marker color, green, is shown.
2. **Edit**. Brings up the editing dialog presented in figure 119 "PADsynth Edit Dialog" on page 153. There one will find a full discussion of that dialog.

7. Kit Edit. Brings up the editing dialog presented in figure 145 "Kit Edit Dialog" on page 173. There one will find a full discussion of that dialog.

8. Effects. Brings up the editing dialog presented in figure 86 "Effects Edit, None" on page 107. There one will find a full discussion of that dialog.

9. Humanise (Rnd. Det.). Small Random Detune, Humanise. Sets the random detune value of the whole *part* in cents. In the most recent versions of *Yoshimi*, this item has been replaced by a **Humanise** slider. This value is an experimental feature. It lends some complexity or piquancy to the part.

Values: `0* to 50`

10. Close. Closes the Edit window.

10 ADDsynth

The *Yoshimi* ADDsynth (also spelled "ADsynth") dialog is a complex dialog for creating an instrument. This is the most complex, most advanced and most sophisticated part of the synthesizer and allows one to edit the parameters that apply to all the voices of ADDsynth.

ADDsynth, a primarily additive synthesis engine, is one of the three major synthesis engines available in *Yoshimi/ZynAddSubFX*. The basic concept of this engine is the summation of a collection of voices, each of which consists of oscillators.

"ADDsynth" (sometimes spelled "ADsynth") or "ADnote" is a complex engine which makes sounds by adding a number of voices. Each one has filters, envelopes, LFOs, morphing, modulation, resonance, etc. Each voice includes a very powerful waveform generator with up to 128 sine/non-sine harmonics. One can use Fourier synthesis, or if one doesn't like it, one can use wave-shaping/filtering of functions. This engine includes anti-aliasing. Modulation includes ring modulation, phase modulation, and more. The modulators can have any shape. [25]

The sum of the voices are passed through filters and amplification to produce the final sound. This could lead one to think that ADDsynth is just a bunch of minor post-processing, and at this level much of the sound generation is hidden.

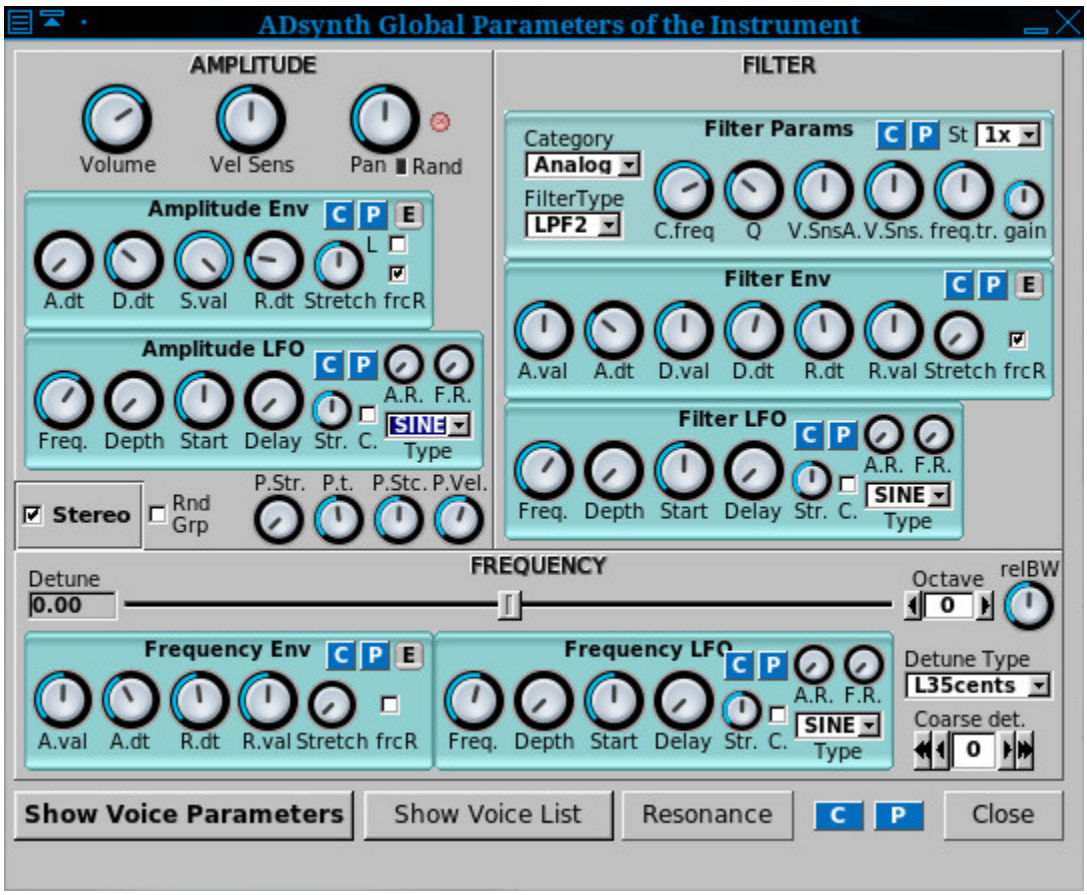

Figure 106: ADDsynth Edit/Global Dialog

The major sections of this dialog are listed:

1. **AMPLITIUDE** (stock section)
2. **FILTER** (stock section)
3. **FREQUENCY** (stock section)
4. **Show Voice Parameters** (section)
5. **Show Voice List** (section)
6. **Resonance** (stock section)
7. **C**
8. **P**
9. **Close**

This complex dialog is best described section by section. Many of the sub-sections are stock sub-panels described elsewhere in this document. References to those sections are included.

10.1 ADDsynth / AMPLITUDE

1. **Volume**
2. **Vel Sens**
3. **Pan**
4. **Rand**

5. **Reset (panning)** (red button)
6. **Amplitude Env** The Amplitude Env panel is described in detail in section 6.5.1 "Amplitude Envelope Sub-Panel" on page 85.
7. **Amplitude LFO** The Amplitude LFO panel is described in detail in section 6.4.5 "LFO User Interface Panels" on page 81.
8. **Stereo**
9. **Rnd Grp**
10. **P.Str.**
11. **P.t**
12. **P.Stc.**
13. **P.Vel.**

Note the two sub-panels, mentioned above, that are described elsewhere. They will not be discussed in detail below.

1. Volume. ADDsynth Volume. Sets the overall/relative volume of the instrument.

Values: `1 to 127, 64*`

2. Vel Sens. ADDsynth Velocity Sensing function. Velocity sensing is simply an exponental transformation from the notes velocity to some parameter change. Observe figure 107 "Velocity Sensing Function" on page 136. It shows how the velocity sensing controls affects the translation of a parameter over the whole range of possible note velocities. Turn the knob rightmost/maximum to disable this function.

Values: `1 to 127, 64*`

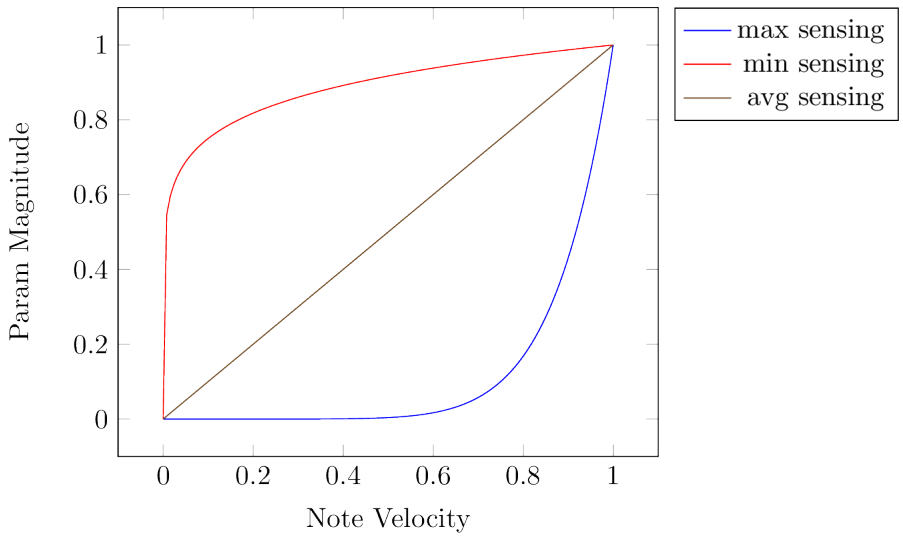

Figure 107: Velocity Sensing Function

3. Pan. ADDsynth Global Panning. Dialing the knob to leftmost or zero gives random panning.

Values: `0, 1 to 127, 64*`

4. Rand. ADDsynth Random Panning Indicator. A red fill-in color provides an indicator for the activation of random panning in this control.

5. Reset (panning). ADDsynth Reset Panning (red button). Clicking this red button changes the panning value to 64 (centered).

Next, we skip the **Amplitude Env** and **Amplitude LFO** panels, which are described elsewhere, as noted above.

6. Stereo. ADDsynth Stereo. Stereo can be enabled. When disabled, all the voices will also have panning disabled.

Values: `Off, On*`

7. Rnd Grp. ADDsynth Random Group. How the harmonic amplitude is applied to voices that use the same oscillator. We need to get a more detailed explanation of what this setting means.

Values: `Off*, On`

8. P.Str. ADDsynth Punch Strength. The punch strength of a note in ADDsynth is a constant amplification to the output at the start of the note, with its length determined by the punch time and stretch and the amplitude being determined by the punch strength and velocity sensing. The **relBW** control in the frequency panel is effectively a multiplier for detuning all voices within an ADnote.

Values: `0* to 127`

9. P.t. ADDsynth Punch Time (duration). Sets the punch effect duration (from 0.1 ms to 100 ms on an A note, 440Hz).

Values: `0 to 127, 64*`

10. P.Stc. ADDsynth Punch Stretch. Sets the punch effect stretch according to frequency. On lower-frequency notes, punch stretch makes the punch effect last longer.

Values: `0 to 127, 64*`

11. P.Vel. ADDsynth Punch Velocity Sensing. The higher this value, the higher the effect of velocity on the punch of the note.

Values: `0 to 127, 72*`

10.2 ADDsynth / FILTER

The ADDsynth FILTER block consists solely of sub-panels described in detail in the sections noted below. The sub-panels of the FILTER section are:

1. **Filter Params**
2. **Filter Env**
3. **Filter LFO**

1. Filter Params. ADDsynth Filter Parameters. The Filter Params panel is described in detail in section 6.2.5 "Filter Parameters User Interface" on page 74.

2. Filter Env. ADDsynth Filter Envelope. The Filter Env panel is described in detail in section 6.5.5 "Envelope Settings for Filter" on page 89.

3. Filter LFO. The Filter LFO panel is described in detail in section 6.4.5 "LFO User Interface Panels" on page 81.

10.3 ADDsynth / FREQUENCY

1. **Detune**
2. **FREQUENCY slider**
3. **Octave**

4. **RelBW**
5. **Frequency Env**. A stock sub-panel described in section 6.5.4 "Envelope Settings, Frequency" on page 88.
6. **Frequency LFO** A stock sub-panel described in section 6.4.7 "Frequency LFO Sub-panel" on page 83.
7. **Detune Type**
8. **Coarse det.**

1. Detune. ADDsynth Detune Value. This display box shows the value of the detune as selected by the frequency slider described below.

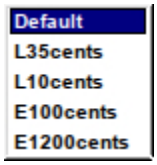

Figure 108: ADDsynth Frequency Detune Type

This value defines the number of cents that define the range of the **FREQUENCY** slider, that is 35 cents, 10 cents, 100 cents (one semitone), or 1200 cents (1 octave), below and above the main frequency. The default is 35 cents. The 1200-cents setting provides a whole octave of detuning in either direction.

The "L" probably stands for "linear", and the "E" for "exponential", to describe how the detune slider acts.

Values: `Default*, L35cents, L10cents, E100cents, E1200cents`

2. FREQUENCY slider. ADDsynth Fine Detune (cents), a slider control. While the detune type dropdown and the octave selection provide a coarse selection of detune, the slider allows for a finer selection of detune, up to roughly one-third of a semitone.

Values: `-35.00 to 35.00`, `-10.00 to 10.00`, `-100.00 to 100.00`, `-1200.00 to 1200.00`

3. Octave. ADDSynth Octave. The octave setting changes the frequency by octaves.

Values: `-8 to 7, 0*`

4. RelBW. ADDSynth Relative Bandwidth. Bandwidth: how the relative fine detune of the voice is changed.

Values: `0 to 127, 64*`

5. Frequency Env. ADDsynth Frequency Envelope. The Frequency Env panel is described in detail in section 6.5.4 "Envelope Settings, Frequency" on page 88.

6. Frequency LFO. The Frequency LFO panel is described in detail in section 6.4.5 "LFO User Interface Panels" on page 81

7. Detune Type. Frequency Detune Type. This setting provides a coarse detuning. We would welcome an explanation of exactly is meant by the numbers and the "E" versus "L" designation.

Values: `L35cents, L10cents, E100cents, E1200cents`

8. Coarse det. Coarse Detune, "C.detune". The one-arrow buttons change the value by one. The two-arrow buttons change the value by ten. Again, we need a way to explain the interactions of the slider, the octave setting, the detune type, and the coarse detune settings.

Values: `-64 to 63, 0*`

9. Show Voice Parameters. ADDsynth Show Voice Parameters. This button brings up the "voice parameters" dialog discussed in the next section. Again, this dialog is built from some stock sections and stock sub-panels, plus additional elements.

10.4 ADDsynth / Voice Parameters

Each *Yoshimi* ADDsynth instrument consists of up to 8 voices. This dialog provides a way to define each of the 8 voices in great detail. By default, an ADDsynth instrument consists of one voice, voice 1.

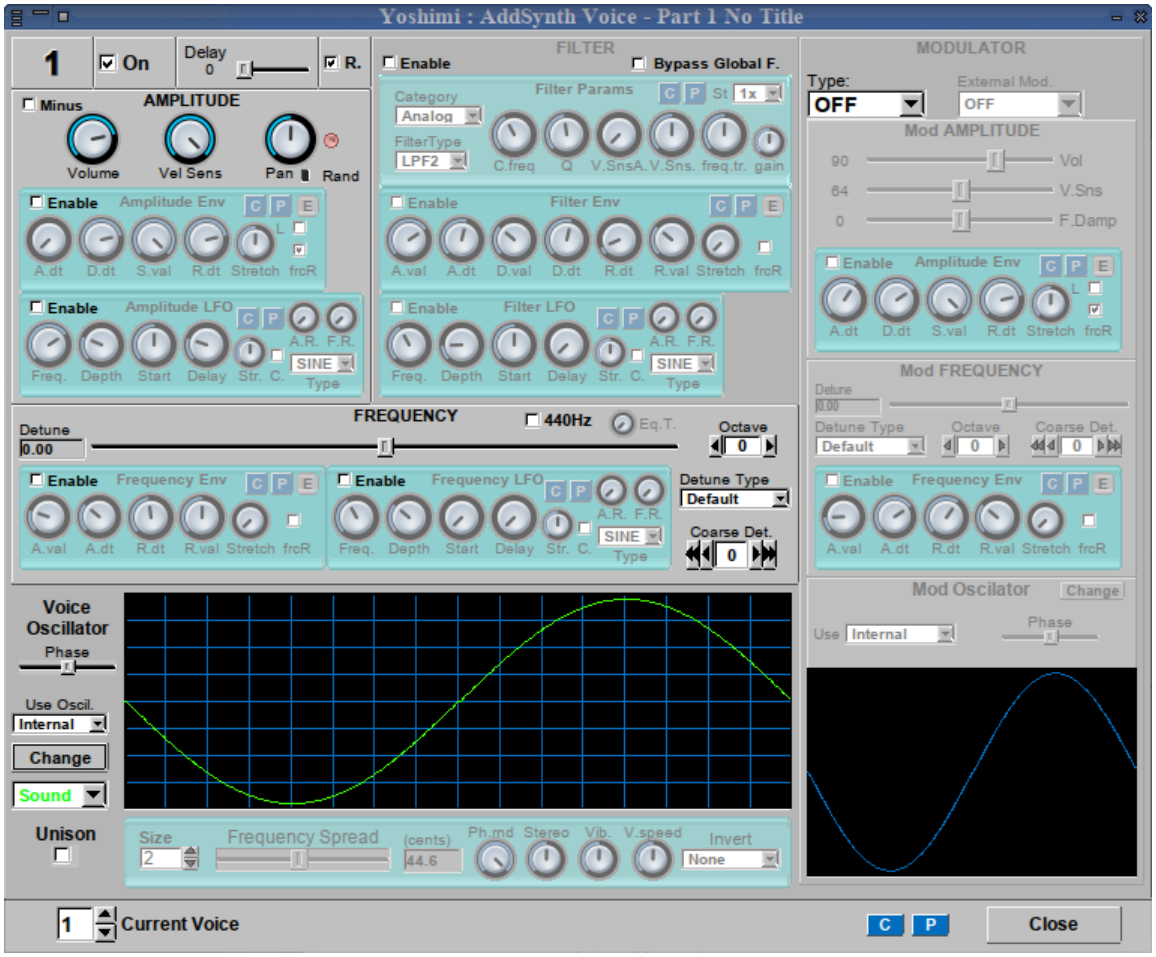

Figure 109: ADDsynth Voice Parameters Dialog

This dialog consists of a few extra settings, plus a number of stock dialog sections. Take some time to compare figure 106 "ADDsynth Edit/Global Dialog" on page 135, which covers the overall instrument, with figure 109 "ADDsynth Voice Parameters Dialog" on page 139, which covers each of the voices. The stock sections in the former cover the whole instrument as one, while the very similar stock sections in the latter cover only the voice they configure. Obviously, the combinations of settings are essentially endless.

Each voice can be amplitude-controlled, filter-controlled, and frequency-controlled. Each voice can also be modulated by an internal modulator. Another property of the voice is that one can tell *Yoshimi* to modulate a given voice with any of the voices that are numbered less than that voice.

1. **Voice Number**

2. **On**
3. **Delay**
4. **R.**
5. **AMPLITUDE** (see the stock-panel section below)
6. **FILTER** (see the stock-panel section below)
7. **MODULATOR** (see the stock-panel section below)
8. **FREQUENCY** (see the stock-panel section below)

1. Voice Number. ADDsynth Voice Number. This display element shows the voice number represented by the settings in this dialog. Each *Yoshimi* part/instrument can consist of up to eight voices. The voice being worked on can be selected using the **Current Voice** selector.

Values: `1* to 8`

2. On. ADDsynth Voice On/Off. Enables this voice in the part/instrument.

Values: `Off, On`

3. Delay. ADDsynth Voice Delay. We still need to determine what the units of the delay are.

Bug: The tooltip for this setting says "Volume".

Values: `0* to 127`

4. R. ADDsynth Voice Resonance On/Off. The rest of the GUI elements (AMPLITUDE, FILTER, MODULATOR, FREQUENCY, and Voice Oscillator) are more detailed, and discussed in the sections that follow.

Values: `Off, On*`

10.4.1 ADDsynth / Voice Parameters / AMPLITUDE

This section of the voice parameters dialog also includes a couple of stock sub-panels that have an additional "Enable" control.

1. **Minus**
2. **Volume**
3. **Vel Sens**
4. **Pan**
5. **Pan randomness indicator**
6. **Pan reset** (red button)
7. **Amplitude Env, Stock + Enable**
8. **Amplitude LFO, Stock + Enable**

1. Minus. ADDsynth Amplitude Minus. This setting enables the inversion of the volume control action. It enables negative values for the volume control of the voice.

Values: `Off*, On`

2. Volume. ADDsynth Voice Volume. Sets the (relative) volume of this voice in the part/instrument.

Values: `0 to 127, 90?`

3. Vel Sens. ADDsynth Voice velocity-sensing function; setting to rightmost/max disables this function.

Values: `0 to 127*`

4. Pan. ADDsynth Voice panning; setting to leftmost/0 gives random panning.

Values: `0 to 127, 64*`

5. Pan randomness indicator. ADDsynth Voice random panning On/Off. Fills in red to indicate that random panning is in force.

6. Pan reset (red button). ADDsynth Center Panning. Clicking this small red button resets the panning to center.

7. Amplitude Env, Stock + Enable. ADDsynth Amplitude Envelope Sub-panel. See section 6.5.1 "Amplitude Envelope Sub-Panel" on page 85. Additionally, the **Enable** checkbox allows the enabling of this component.

8. Amplitude LFO, Stock + Enable. ADDsynth Amplitude LFO Sub-panel. See section 6.4.5 "LFO User Interface Panels" on page 81. Additionally, the **Enable** checkbox allows the enabling of this component.

10.4.2 ADDsynth / Voice Parameters / FILTER

This section of the voice parameters dialog also includes a couple of stock sub-panels that have an additional "Enable" control.

1. **Enable**
2. **Bypass Global F.**
3. **Filter Params, Stock**
4. **Filter Env, Stock + Enable**
5. **Filter LFO, Stock + Enable**

1. Enable. ADDsynth Voice Enable Filter. This value enables the whole FILTER dialog section.

Values: `Off*, On`

2. Bypass Global F. ADDsynth Voice Bypass Global Filter. The voice signal bypasses the global filter.

Values: `Off*, On`

We need to make sure there is a discussion of the global filter.

3. Filter Params, Stock. See section 6.2.5 "Filter Parameters User Interface" on page 74.

4. Filter Env, Stock + Enable. See section 6.5.5 "Envelope Settings for Filter" on page 89.

5. Filter LFO, Stock + Enable. See section 6.4.6 "Filter LFO Sub-panel" on page 82.

10.4.3 ADDsynth / Voice Parameters / MODULATOR

1. **Type:**
2. **External Mod.**
3. **Mod AMPLITUDE**
4. **Mod FREQUENCY**
5. **Mod Oscillator**

1. Type:. ADDsynth Modulator Type.

Figure 110: Voice Modulator Type

1. **OFF**. This setting turns off the modulator.
2. **MORPH** The morph modulator works by combining the output of two oscillators into one, with the amplitude envelope translating between one waveform and the other.
3. **RING** The ring modulator is useful for making bell-like sounds and some weird effects. The ring modulator works by multiplying two waveforms together, producing a signal that possesses the sum and difference of the frequencies present in the waveforms. The ins-and-outs of the ring modulator are explained in detail in paragraph 10.4.3.1 "Tip: Using the Ring Modulator" on page 143.
4. **PM** The PM (pulse modulation) modulator works by using a modulator envelope to change the pulse width of a pulse waveform. Generally, set **F.Damp** to zero, so that the modulation amount doesn't depend on the note number.
5. **FM** The (frequency modulation) morph modulator works by modulator the frequency. Examples can be heard in the "Ethereal" and "Steel Wire" instruments.
6. **PITCH** The pitch modulator works by... we're not sure... is this pitch shifting?

Values: `OFF, MORPH, RING, PM, FM, PITCH`

2. External Mod.

External Oscillator. This feature allows one of the voices (of the up to 8 allowed in a single ADDsynth instrument) to be used as a modulator or external oscillator for another voice in the instrument. This option specifies to use the oscillator of another voice, or -1 for the *internal* oscillator.

The parameters must be lower than the voice index; one cannot use the oscillator from a voice with a bigger index (e.g. one can't use the oscillator of voice 8 for voice 4). This is very useful because, if one uses many voices with the same oscillator settings, one can use only one oscillator and select other voices to use this, and if one changes a parameter of this oscillator, all voices using this oscillator will be affected.

External Modulator. Use another voice as a modulator instead of the modulator of the internal voice. One can make a modulation "stack". The modulator of the voice is disabled.

External. Uses the oscillator as the modulator of another voice. It behaves like **Ext. Oscil**, except that it works on the *modulator*. Please notice the difference between this parameter and **Ext. Mod**. See below.

Values: `OFF*, Other voice numbers?`

3. Mod AMPLITUDE. Modulator Amplitude.

1. **Vol** Volume. Values: `0 to 127, 90*`
2. **V.Sns** Velocity Sensing Function; set to rightmost/max to disable. Values: `0 to 127, 64*`
3. **F.Damp** Modulator Damp at higher frequency. How the modulator intensity is lowered according to lower/higher note frequencies. Values: `0 to 127, 90*`
4. **Amplitude Env, Stock + Enable** See section 6.5.1 "Amplitude Envelope Sub-Panel" on page 85.

4. Mod FREQUENCY. Modulator Frequency.

1. **Detune slider** Fine Detune (cents). Values: `-35.00 to 35.00, 0*`

2. **Detune Type** Fine Detune (cents). Values: `L35cents, L10cents, E100cents, E1200cents` See figure 108 "ADDsynth Frequency Detune Type" on page 138.
3. **Octave** Octave. Values: `-8 to 7, 0*`
4. **Coarse Det.** Coarse Detune. Values: `-64 to 63, 0*`
5. **Filter Env, Stock + Enable** See section 6.5.5 "Envelope Settings for Filter" on page 89.

5. Mod Oscillator. Modulator Oscillator.

Bug: The word "oscillator" is mispelled in the application.

1. **Change** ADDsynth Oscillator Editor.
2. **Use** Oscillator to Use. See the paragraph below. Values: `Internal*, Other oscillators?`
3. **Phase** Oscillator Phase. Values: `0 to 360 (0 to 2PI)`
4. **Waveform graph** Waveform graph.

As far as we can tell, one has the choice between **Internal**, which in this case means a completely independent modulator oscillator per voice (extra change button), or **External**, which refers to the modulation oscillators one has already defined for the voices with a lower index. This means one can make one modulation oscillator for voice 1, and reuse it in voices 2 and 3. This is the same system used for the normal oscillators.

10.4.3.1 Tip: Using the Ring Modulator

This section is derived from one of the short text files in the *Yoshimi* source-code bundle ([17] or [18]). It notes that "Some people have been confused about how to use an 'external' Mod Oscillator", and provides usage notes that we will elaborate on here. Here is the way to use the ring modulator:

1. Open the ADDsynth editing window. Then open **Show Voice Parameters**.
2. For **Type**, select the **RING** value. This selection will activate the **Mod Oscillator**.
3. In the **Mod Oscillator**, click on **Change** to open the **ADDsynth Oscillator Editor**.
4. Set the wave-shape to **Triangle**.
5. Switch to voice number 2 and enable it.
6. Again, for **Type**, select the **RING** value. However, feel free to select one of the other modulators, if one wishes.
7. One can now use **Internal** for voice 2, or select **ext.m** 1, to use the first voice as in internal modulator.
8. Change the internal voice to, for example, **Square**.
9. Do the same setup for voice 3. One will find that one can use its **Internal** or either of the two previous ones.

Now the joker in the pack is that one can disable both the previous voices but *still* use their Mod Oscillators.

In a newsgroup ([12], the following note is found.

> Say I want the A tone ring-modulated by 880Hz. A is 440 Hz, the ring modulation setting lets me choose the modulation frequency relative to the frequency of the tone. So I choose octave 1 and let the detune at zero. If I move the detune, it'll shift the modulation frequency a bit, which will make a disharmonic effect.
>
> Wet/dry setting is controlled by volume in "modulation amplitude". The modulation frequency can further be multiplied or several modulations can be simulated by changing the oscillator waveform.
>
> One huge letdown is that it is only available for Adsynth. PadSynth does not seem to have ring modulation option, so the coolest sounds stay out of question for massive lead tones. :-(

We have provided a more useful "tutorial" on using the ring modulator in the [3] document.

10.4.4 ADDsynth / Voice Parameters / FREQUENCY

This frequency section is almost a stock part. It is similar to the ADDsynth Edit's **FREQUENCY** section.

1. **Detune**
2. **FREQUENCY slider**
3. **440Hz**
4. **Eq.T**
5. **Octave**
6. **Detune Type**
7. **Coarse det**
8. **Frequency Env, Stock + Enable**
9. **Frequency LFO, Stock + Enable**
10. **Voice Oscillator**

1. Detune. Voice Parameters Detune. Shows the value selected by the frequency slider.

2. FREQUENCY slider. Frequency Slider. Provides fine detune, in cents. Note that 35 cents is roughly one-third of a semitone.

Values: `-35.00 to 35.00, 0*`

3. 440Hz. 440 Hz Selection. Fixes the voice base frequency to 440 Hz. One can adjust this with the detune settings. No matter what key is played on the keyboard, this voice will emit only 440 Hz. This is useful for defining a constant frequency to use as a modulator for the other voices in the part. For example, one can define voice 1 to be a tone, then define voice 2 to be 440 Hz. The two voices will mix, but only voice 1 will change frequencies as different keys are played.

Values: `Off*, On`

4. Eq.T. Equal Temperament? This item is enabled only if the **440Hz** check-box is enabled. It determines how the frequency varies according to the keyboard. If set to its leftmost (0) value, then the frequency is fixed.

Values: `0 to 127?`

5. Octave. Voice Parameters Octave.

Values: `-8 to 7, 0*`

6. Detune Type. Detune Type.

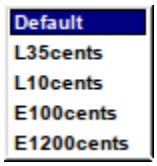

Figure 111: Frequency Detune Type

Values: `Default*, L35cents, L10cents, E100cents, E1200cents`

7. Coarse det. Coarse Detune. Is this setting in units of semitones?

Values: `-64 to 63, 0*`

8. Frequency Env, Stock + Enable. Frequency Envelope. See section 6.5.4 "Envelope Settings, Frequency" on page 88.

9. Frequency LFO, Stock + Enable. Frequency LFO. See section 6.4.7 "Frequency LFO Sub-panel" on page 83.

10. Voice Oscillator. Voice Parameters Oscillator. See the next section.

10.4.5 ADDsynth / Voice Parameters / Voice Oscillator

The ADDsynth Voice Oscillator panel is tucked in the lower left side of the ADDSynth Voice Parameters editor.

1. **Phase**
2. **Use**
3. **Waveform graph**
4. **Change**
5. **Sound**
6. **Unison**
7. **Current Voice**
8. **C**
9. **P**
10. **Close Window**

1. Phase. Voice Oscillator Phase.

Values: `0 to 360 (0 to 2*pi)`

2. Use Oscil. Use Oscillator. If the **Current Voice** is set to a value greater than 1, meaning that one is editing additional voices, then this dropdown item also includes the values of all oscillators less then this one, marked as "External". For example, if one is currently editing current voice 3, then the dropdown list includes **Internal**, **Ext. 1**, and **Ext. 2**.

Values: `Internal*, Other oscillators`

3. Waveform graph. Waveform Graph. Shows a period of the currently configured oscillator.

4. Change. ADDSynth Voice Oscillator Change. This button brings up the ADDsynth Oscillator Editor dialog. This dialog is described elsewhere.

5. Sound. ADDSynth Oscillator Type (sound/noise). Sound/Noise choice. Select the mode of the oscillator (sound versus white noise).

Figure 112: Voice Oscillator Choices

Values: `Sound*, Noise (white), Noise (pink)`

If white **Noise** is selected, then the waveform graph simply announces "White Noise". Also, the **Unison** control is disabled.

Figure 113: White Noise in ADDSynth Voice

If white **Noise** is selected, then the waveform graph simply announces "Pink Noise". Also, the **Unison** control is disabled.

Figure 114: Pink Noise in ADDSynth Voice

6. Unison. Unison is useful in creating the chorus like sound of many simultaneous oscillators.

Values: `Off*, On`

Enabling this item causes the Unison-related items to become enabled (they are discussed next, starting with **Size** and **Frequency Spread**.

1. **Size**
2. **Frequency Spread**
3. **Ph.rnd**
4. **Stereo**
5. **Vibrato**
6. **V.speed**
7. **Invert**

1. Unison Size. Sets the number of unison sub-voices.

Values: `2* to 50`

2. Unison Frequency Spread. Frequency spread of the unison (cents).

Values: `0 to 200, 44.6*`

3. Phase Randomness. Unison Phase Randomness.

Values: `0 to 127*`

4. Stereo Spread. Unison Stereo Spread.

Values: `0 to 127, 64*`

5. Unison Vibrato. Unison Vibrato.

Values: `0 to 127, 64*`

6. Vibrato Speed. Unison Vibrato Average Speed.

Values: `0 to 127, 64*`

7. Phase Invert. Unison Phase Invert. Values: `None*, Random, 50%, 33%, 25%, 20%`

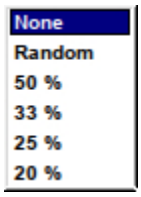

Figure 115: Unison Phase Invert Dropdown

Finally, at the bottom of the ADDsynth Voice Part dialog, we find the last few controls.

1. Current Voice. Current Voice.

Values: `1* to 8`

2. C. Copy D note parameters ("DnoteParameters").

3. P. Paste D note parameters ("DnoteParameters").

4. Close Window. Close.

10.5 ADDsynth / Voice List

The ADDsynth Voices List shows a summary of voices 1 to 8, and allows some overall control of them, almost like a simple mixer. It is brought on-screen via the **Show Voice List** button of the ADDsynth global part editor.

Figure 116: ADDsynth Voices List

1. **No. (1 to 8)**
2. **Waveform**
3. **Vol**
4. **Pan**
5. **R.**
6. **Detune**
7. **Vib. Depth**

8. **Hide Voice List**

1. No. (1 to 8). Voice List Number. This check-box enables or disables a given voice in the current part.

Values: `Off, On`

2. Waveform Icon. Waveform Icon. The waveform icon shows a rough rendering of the actual shape of the voice waveform, or the letter **N** is the voice is constructed from white noise. Note that this picture isn't updated, if the voice is edited, until the voice list is closed and reopened.

3. Vol. Voice Volume. This slider controls the relative volume of a given voice in the current part.

Values: `0 to 127, 100*`

4. Pan. Voice Panning (0/leftmost is Random). This slider controls the panning of a given voice in the current part.

Values: `0 to 127, 64*`

5. R. Resonance On/Off. Enable/disable the resonance effect of a voice. Note that the resonance is configured in by the Resonance dialog brought up by the **Resonance** button at the bottom of the ADDsynth main dialog. The resonance dialog has its own Enable button, as well. These seem to be independent settings.

Values: `Off, On*`

6. Detune Value. This read-only text-box shows the current value of detune as selected by its slider.

7. Detune. Fine Detune (cents).

Values: `-35 to 35, 0*`

8. Vib. Depth. Frequency LFO Amount/Depth. This setting can be very useful because, with the detune settings, one can create very good sounding instruments.

Values: `0 to 127, 40*`

9. Hide Voice List. Hide Voice List. A Close button, really, and that is what it is in the latest version of *Yoshimi*.

10.6 ADDsynth / Oscillator

Pressing the ambiguously-named **Change** button in the ADDsynth voice-editor dialog brings up a very complex dialog for modifying the harmonics of the voice.

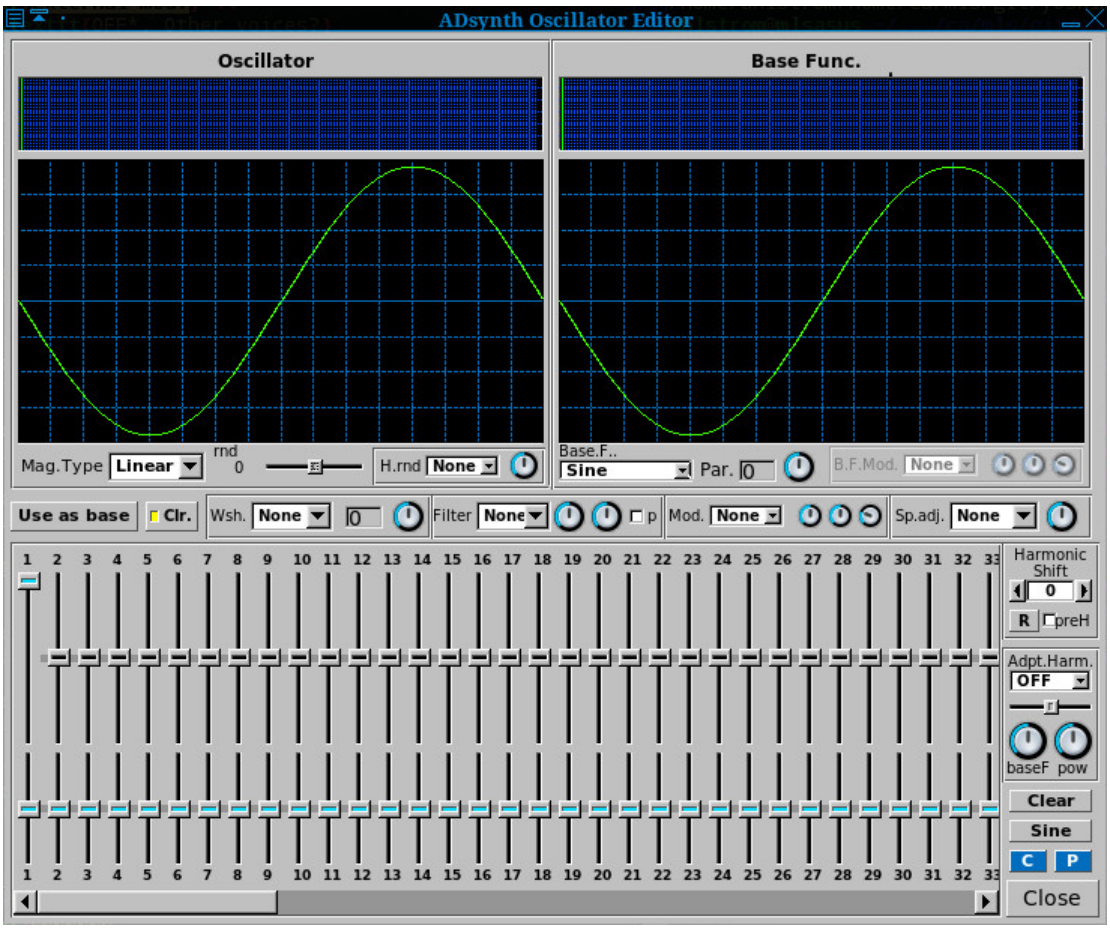

Figure 117: ADDsynth Oscillator Editor

This item is nearly identical to the PADsynth harmonic editor depicted in figure 132 "Harmonic Content Editor" on page 160, except for the items noted below. Obviously, it is a topic unto itself!

1. **Oscillator Spectrum Graph**
2. **Oscillator Waveform Graph**
3. **Mag.Type**
4. **rnd** (ADDsynth Oscillator Editor only)
5. **H.rnd** (ADDsynth Oscillator Editor only)
6. **H.rnd knob** (ADDsynth Oscillator Editor only)

1. Oscillator Spectrum Graph. Oscillator Spectrum Graph. This graph shows the spectrum of the oscillator as a series of vertical lines, a kind of frequency histogram.

2. Oscillator Waveform Graph. Oscillator Waveform Graph. This graph shows the temporal waveform of the oscillator.

3. Mag.Type. Oscillator Magnitude Type. Sets how the magnitudes from the user interface behave. See the values below.

4. rnd. rnd. Sets the randomness of the oscillator output. There are 2 types of randomness provided. The first type of randomness is *group randomness*, where the oscillator starts at random positions within the waveform, and the second type of randomness is *phase randomness*, which is settable from -64 (the maximum) to -1 (the minimum), the oscillator is phase distorted, and each harmonic is from 1 (the

maximum) to 63 (the minimum). A setting of zero (0) is no randomness. One can use this parameter to make warm sounds like analogue synthesizers.

5. H.rnd. Harmonic Amplitude Randomness. Adjusts how the randomness is employed. Not sure how this works, so, for now, experiment!

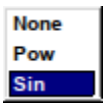

Figure 118: ADDsynth Oscillator Harmonic Randomness Selections

Values: `None, Pow, and Sin`

6. H.rnd knob. Harmonic Amplitude Randomness Knob. This knob provides the oscillator's spectrum-adjust parameter.

Values: `0 to 127, 64*`

10.7 ADDsynth / Resonance

This dialog, shared in common with the PADsynth editor, is a stock user-interface element described in section 6.3 "Stock Resonance Settings" on page 76.

11 PADsynth

The *Yoshimi* PADsynth dialog is a complex dialog for creating a pad instrument, "PADsynth" or "PADnote" is engine that makes very beautiful pads and other instruments. (These instruments can be exported for use with other programs too).

The PADsynth dialog consists of two major tabs, **Harmonic Structure** and **EnvelopesLFOs**. Each of these tabs is complex, so the discussion will break the tabs down by sub-sections.

11.1 PADsynth / Algorithm

The complexity of the PADsynth dailog merely reflects the complexity of the PADsynth algorithms.

11.1.1 PADsynth / Algorithm / General

The PADsynth algorithm generates very beautiful sounds, even if its idea is much simpler than other algorithms. It generates a perfectly looped wave-table sample which can be used in instruments. It easily generates sounds of ensembles, choirs, metallic sounds (bells) and many other types of sound. Paul Nasca wanted to make this algorithm known, and everyone is welcome to learn and use this algorithm into one's projects or products (non-commercial or commercial).

Quote [25]:

> You will not be disappointed by this algorithm.
> I hope that this algorithm will be implemented in many software/hardware synthesizers. Use it, spread it, write about it, create beautiful instruments with it. If your synthesizer uses plenty of samples, you can use this algorithm to generate many ready-to-use samples.

In order to understand how this algorithm works, one needs to be familiar with howto think about musical instruments. Please read an introduction for the description of the meaning and the importance of bandwidth of each harmonic and randomness.

This algorithm generates some large wave-tables that can be played at different speeds to get the desired sound. This algorithm describes only how these wave-tables are generated. The result is a perfectly looped wave-table. Unlike other synthesis methods, which use the Inverse Fast Fourier Transform, this one does not use overlap/add methods and there is only one IFFT for the whole sample.

The basic steps are:

1. Make a very large array that represents the amplitude spectrum of the sound (all default values are zero).
2. Generate the distribution of each harmonic in the frequency spectrum and add it to the array.
3. Put random phases to each frequency of the spectrum.
4. Do a single Inverse Fourier Transform of the whole spectrum. There is no need of any overlapping windows, because there is only one single IFFT for the whole sample.

The output is a sample which can be used as a wave-table.

11.1.2 PADsynth / Algorithm / Harmonic Bandwidth

We consider one harmonic (overtone) as being composed of many frequencies. These sine components of one harmonic are spread over a certain band of frequencies. Higher harmonics have a wider bandwidth. In natural choirs/ensembles the bandwidth is proportional to the frequency of the harmonic.

The harmonics become wider and wider, until a certain frequency, where they may merge into a noise band (as in the full spectrum image from above shows). This is a normal thing and we recommend to not avoid this by limiting the bandwidth of the harmonics.

This describes the function of the spread of the harmonic. Here are some examples of how they can be spread:

1. A special case is where there is only a single sine component inside the harmonic In this case, the harmonic and the "sine component" are the same thing.
2. Detuned. In this case there are two sine components which are detuned.
3. Evenly spread inside the harmonic (all components have the same amplitude)
4. Normal (Gaussian) distribution. The sine components amplitude are bell-shaped. The largest amplitude is in the center of the band. This distribution gives the most natural sounds (it simulates a very, very large ensemble).

Of course, one can use many other profiles of the harmonic. *ZynAddSubFX*'s PADsynth module offers many ways to generate the harmonic profile. Also, it's very important that the harmonic must have the same amplitude, regardless of the profile functions/parameters and the bandwidth. For many more details of this algorithm, see Paul Nasca's document [25].

11.1.2.1 Tip: Using the PADsynth

Keep in mind that the resulting wave-tables are perfectly looped. There are some sound-generation ideas to keep in mind:

1. When using the wave-tables for instruments, on each Note On, start from a random position and not from the start. This avoids hearing the same sound on each keystroke.
2. One can use the same wave-table for generating stereo sounds, by playing the same wave-table at different positions for left and right. The best method is to create a difference between left and right of N/2.
3. Generate different wave-tables for different pitches and use the one that is closest to the desired pitch.
4. Upsample or downsample the amplitude array of the harmonic before running the algorithm, according to the fundamental frequency. In this case we need to set a parameter "base_frequency" which represents the frequency where the array is left unchanged.

Example: We have A_orig[]=1,2,1,3,0,0,1,0 and base_frequency is equal to 440 Hz Here are some cases:

A[] for 440 Hz: is the same as A_orig[]

A[] for 220 Hz: is the A_orig[] upsampled by factor of 2

so: A[]=1, 1, 1.5, 2, 1.5, 1, 2, 3, 1.5, 0, 0, 0, 0.5, 1, 0.5, 0

(the original A_orig amplitudes are shown as bold)

A[] for 880 Hz: the A_orig[] is downsampled by a factor of 2

so: A[]=1.5, 2, 0, 0.5

A[] for F Hz: the A_orig[] is scaled by a factor of 440/F.

Even if this idea is very simple, the resulting sounds are very natural, because it keeps the spectrum constant according to the frequency of the harmonic and not to the number of the harmonics. This follows from the document where Paul Nasca describes some principles regarding synthesis.

11.2 PADsynth / Harmonic Structure

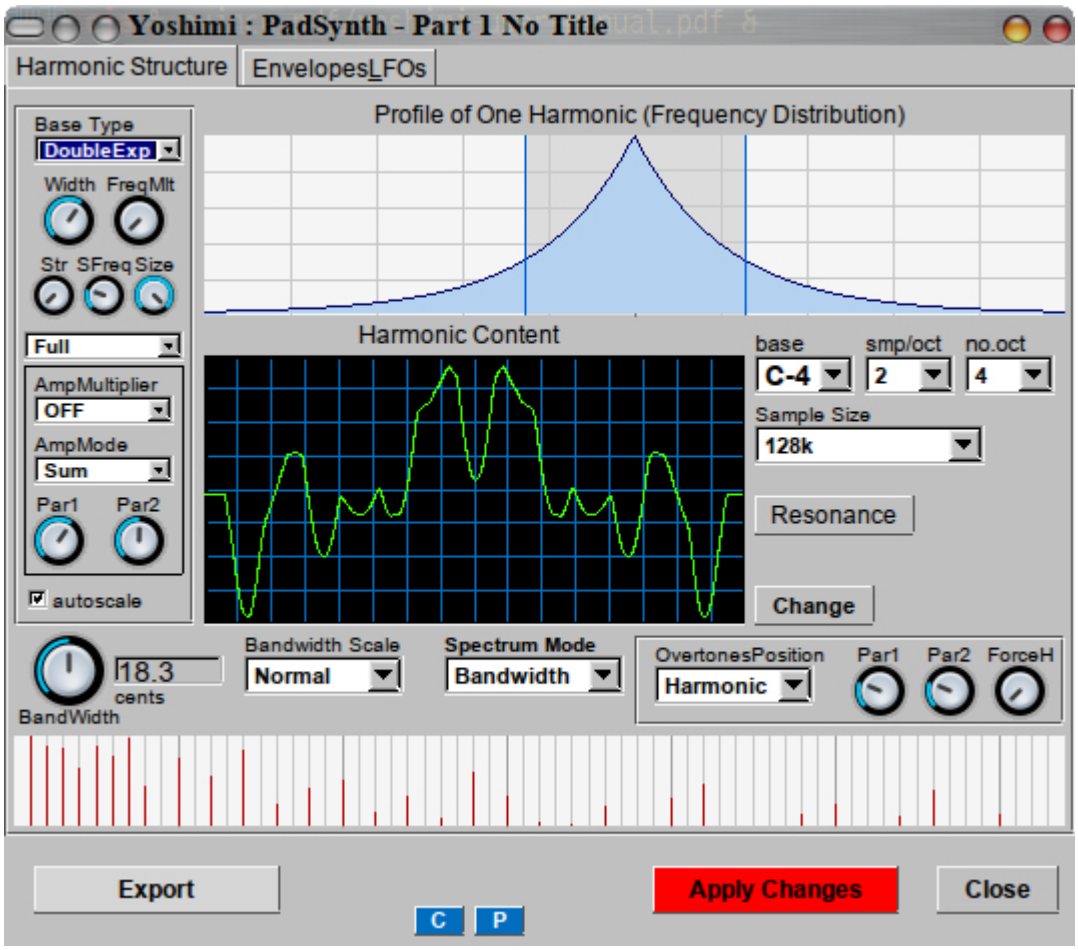

Figure 119: PADsynth Edit Dialog

Note how, in the newer versions of *Yoshimi*, the part number and part name are part of the window caption. There are a lot of parameter sections to keep track of, and to describe.

1. **Basics** (section)
2. **Harmonic** (section)
3. **Resonance** (section)
4. **Change** (section)
5. **Bandwidth and Position** (section)
6. **Export** (section)
7. **C**
8. **P**
9. **Apply Changes**
10. **Close**

Some of these elements have their own sections devoted to them, below.

11.2.1 PADsynth / Harmonic Structure / Basics

1. **BaseType**
2. **Width**
3. **FreqMlt**
4. **Str**
5. **SFreq**
6. **Size**
7. **Full/Upper/Lower**
8. **AmpMultiplier**
9. **AmpMode**
10. **Par1**
11. **Par2**

1. BaseType. Base Type of Harmonic.

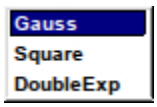

Figure 120: Base Type of Harmonic

Values: `Gauss*, Square, DoubleExp`

2. Width. Width of Harmonic. The lowest value yields a very thin **Profile of One Harmonic (Frequency Distribution)** waveform, pretty close to a Dirac delta function (in this case, pretty close to a sine wave). The highest value yields a broadband spectrum, almost a flat spectrum, but it has a hump around the center frequency.

Values: `0 to 127`

3. FreqMlt. Frequency Multiplier. Increasing this value causes more and more repetitions of the harmonic spectrum frequency distribution to appear. A value of 127 yields 32 repetitions of the spectrum.

Values: `0 to 127`

4. Str. Stretch. Not quite sure what this one does. Increasing it adds harmonics to the spectrum and alters the distribution of their levels.

Values: `0 to 127`

5. SFreq. Harmonic Sfreq. Not quite sure what this one does. Increasing it tightens up the spread of harmonics.

Values: `0 to 127`

6. Size. Harmonic Size. Increasing this value preserves the shape of the spectrum, but widens it.

Values: `0 to 127`

7. Full/Upper/Lower. Harmonic Sidebands. These menu entries select the full spectrum, or filter in only the upper sidebands of the spectrum, or the lower sidebands.

Values: `0 to 127`

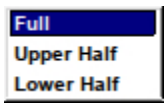

Figure 121: PADsynth Full/Upper/Lower Harmonics

Values: `Full*, Upper Half, Lower Half`

8. AmpMultiplier. Amplitude Multiplier. These values spread out the frequency components in various ways.

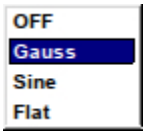

Figure 122: PADsynth Amplitude Multiplier

Values: `OFF*, Gauss, Sine, Flat`

9. AmpMode. Amplitude Mode.

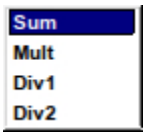

Figure 123: PADsynth Amplitude Mode

Values: `Sum*, Mult, Div1, Div2`

10. Par1. Harmonic Parameter 1. Increasing this parameter narrows the width of the central spectral component.

Values: `0 to 127`

11. Par2. Harmonic Parameter 2. Varying this parameter changes the relative amplitude of the central spectral component and the sidebands.

Values: `0 to 127`

11.2.2 PADsynth / Harmonic Structure / Harmonic

1. **Profile of One Harmonic**
2. **Harmonic Content Window**
3. **base**
4. **smp/oct**
5. **no.oct**
6. **Sample Size**
7. **Resonance** (section)
8. **Change** (section)

1. **Profile of One Harmonic.** Profile of One Harmonic (Frequency Distribution).
2. **Harmonic Content Window.** Harmonic Content Window.
3. **base.**

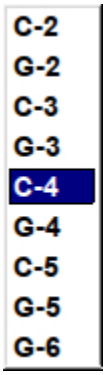

Figure 124: Harmonic Base Dropdown

Values: `C-2, G-2, C-3, G-3, C-4*, G-4, C-5, G-5, G-6`

4. **smp/oct.** Harmonic Samples Per Octave.

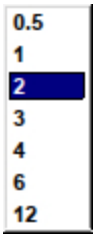

Figure 125: Harmonic Samples Per Octave

5. **no.oct.** Number of Octaves of Harmonic.

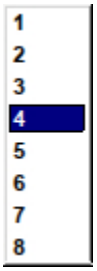

Figure 126: Harmonic Number of Octaves

Values: `1, 2, 3, 4*, 5, 6, 7, 8`

6. **Sample Size.** Harmonic Sample Size.

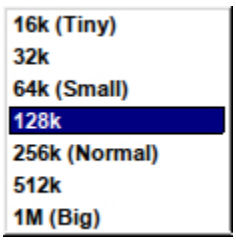

Figure 127: Harmonic Sample Size Dropdown

Values: `16k (Tiny)`, `32k`, `64k (Small)`, `128k*`, `256k (Normal)`, `512k`, `1M (Big)`

11.2.3 PADsynth / Harmonic Structure / Bandwidth and Position

1. **BandWidth**
2. **cents**
3. **Bandwidth Scale**
4. **Spectrum Mode**
5. **OvertonesPosition**
6. **Par1**
7. **Par2**
8. **ForceH**
9. **Harmonics Plot**

1. BandWidth. Harmonics Bandwidth.

Values: `0 to 127`

2. cents. Bandwidth Reading (cents).

3. Bandwidth Scale. Bandwidth Scale.

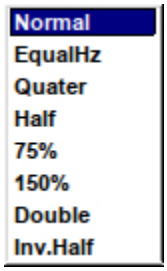

Figure 128: Harmonics Bandwidth Scale.

Values: `Normal`, `EqualHz`, `Quater`, `Half`, `75%`, `150%`, `Double`, `Inv. Half`

4. Spectrum Mode. Harmonics Spectrum Mode.

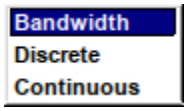

Figure 129: PADsynth Harmonics Spectrum Mode

Values: `Bandwidth*, Discrete, Continuous`

5. OvertonesPosition. Overtones Position.

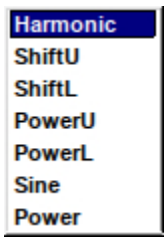

Figure 130: PADsynth Overtones Position

Values: `Harmonic*, ShiftU, ShiftL, PowerU, PowerL, Sine, Power`

6. Par1. PADSynth Bandwidth Parameters 1. If the **Overtones Position** drop-down is set to something other than `Harmonic`, then this knob changes the harmonic lines shown in the spectrum view just below this item. It is best to play with this setting and observe and hear the changes it makes.

7. Par2. PADSynth Bandwidth Parameters 2. If the **Overtones Position** drop-down is set to something other than `Harmonic`, then this knob changes the harmonic lines shown in the spectrum view (**Harmonics Plot**) just below this item. It is best to play with this setting and observe and hear the changes it makes.

8. ForceH. PADSynth Bandwidth ForceH. If the **Overtones Position** drop-down is set to something other than `Harmonic`, then this knob changes the harmonic lines shown in the spectrum view just below this item. It is best to play with this setting and observe and hear the changes it makes.

9. Harmonics Plot. PADSynth Harmonics Plot. Shows the position and amplitude of each of the harmonic lines that the settings will generate.

11.2.4 PADsynth / Harmonic Structure / Export

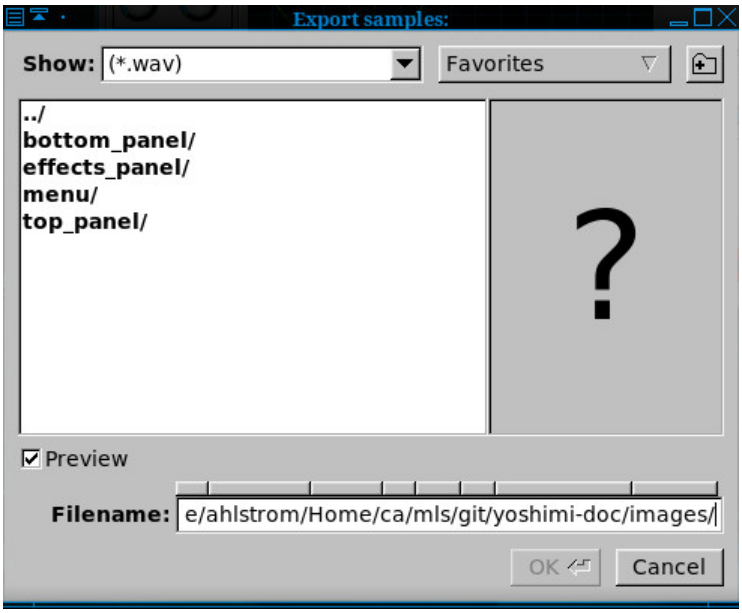

Figure 131: Harmonics Structure Export Dialog

This export dialog is a file dialog similar to other file dialogs in *Yoshimi*.

11.2.5 PADsynth / Harmonic Structure / Resonance

This dialog, shared in common with the ADDsynth editor, is a stock user-interface element described in section 6.3 "Stock Resonance Settings" on page 76.

11.2.6 PADsynth / Harmonic Structure / Change

The ambiguously-named **Change** button brings up the Harmonic Content editor, which is another complex dialog. Like figure 117 "ADDsynth Oscillator Editor" on page 149, it allows one to create an essentially unlimited number of oscillators.

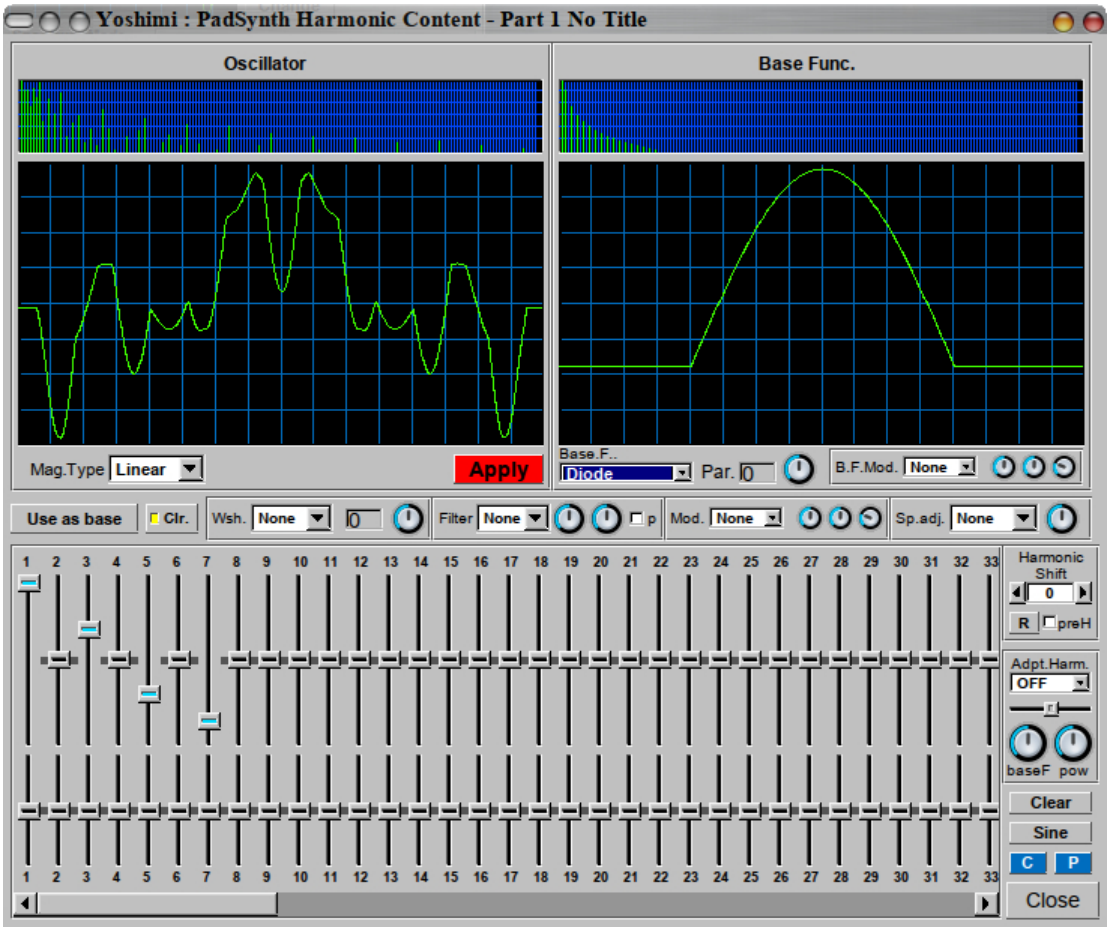

Figure 132: Harmonic Content Editor

This dialog is complex enough that it makes sense to break it down into sub-sections.

1. **Oscillator** (section)
2. **Base Function** (section)
3. **Middle** (section)
4. **Harmonic** (section)

11.2.6.1 PADsynth / Harmonic Structure / Change / Oscillator

1. **Oscillator Spectrum Graph**
2. **Oscillator Waveform Graph**
3. **Mag.Type**
4. **Apply**

1. Oscillator Spectrum Graph. Oscillator Spectrum Graph. This graph shows the spectrum of the oscillator as a series of vertical lines, a kind of frequency histogram.

2. Oscillator Waveform Graph. Oscillator Waveform Graph. This graph shows the temporal waveform of the oscillator.

3. Mag.Type. Oscillator Magnitude Type. Sets how the magnitudes from the user interface behave. See the values below.

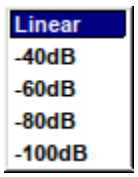

Figure 133: PADsynth Harmonic Content Mag Type

Values: `Linear*, -40dB, -60db, -80dB, -100dB`

4. Apply. PADsynth Harmonic Content Editor Apply Button.

11.2.6.2 PADsynth / Harmonic Structure / Change / Base Function

1. **Base Func. Spectrum Graph**
2. **Base Func. Waveform Graph**
3. **Base F..**
4. **Par. Value**
5. **Par. Wheel**
6. **B.F.Mod.**
7. **Wheel 1**
8. **Wheel 2**
9. **Wheel 3**

1. Base Func. Spectrum Graph. Harmonic Base Function Spectrum Graph.

2. Base Func. Waveform Graph. Harmonic Base Function Waveform Graph.

3. Base F. Harmonic Base Function. Sets what function to use as the harmonics base function. One can use any base function as harmonics.

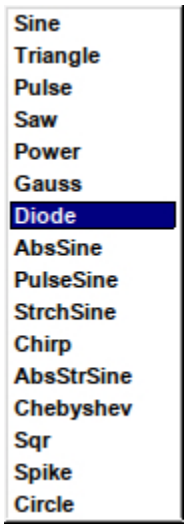

Figure 134: PADsynth Harmonic Content Base Function

Values: `Sine*, Triangle, Pulse, Saw, Power, Gauss, Diode, AbsSine, PulseSine, StrchSine, Chirp, AbsStrSine, Chebyshev, Sqr, Spike, Circle`

4. Par. Value. PADsynth Parameter Value.

5. Par. Wheel. PADsynth Parameter Wheel. Change the parameter of the base function.

6. B.F.Mod. PADSynth Base Frequency Mod. This item is very similar to the Harmonic Editor Modulation (**Mod.**) setting.

Values: `None*, Rev, Sine, Pow`

7. Wheel 1. PADsynth Wheel 1. With the **B.F.Mod.** selection set to something other than `None`, this modifies one (unknown at this time) parameter of the modulation selection.

8. Wheel 2. PADsynth Wheel 2. With the **B.F.Mod.** selection set to something other than `None`, this modifies one (unknown at this time) parameter of the modulation selection.

9. Wheel 3. PADsynth Wheel 3. With the **B.F.Mod.** selection set to something other than `None`, this modifies one (unknown at this time) parameter of the modulation selection.

11.2.6.3 PADsynth / Harmonic Structure / Change / Middle

1. **Use as base**
2. **Clr.**
3. **Wsh.**
4. **Wsh Value**
5. **Wsh Wheel**
6. **Filter**
7. **Filter Wheel 1**
8. **Filter Wheel 2**
9. **Filter p**
10. **Mod.**
11. **Mod. Wheel 1**
12. **Mod. Wheel 2**
13. **Mod. Wheel 3**
14. **Sp.adj.**
15. **Sp.adj. Wheel**

1. Use as base. Use as Base. Convert the oscillator output to a base function. Changing the Base function or its parameter will erase the converted base function.

2. Clr. Clear. Clear the settings and make the oscillator equal to a base function. If this is cleared, one can click the **Use as base** button to make multiple conversions to base functions.

3. Wsh. Harmonic Editor Wave-shaping, "W.sh".

Wave shaping function that applies to the oscillator. It has one parameter that fine-tunes the wave-shaping function.

4. Wsh Value. Harmonic Editor Wave-shaping Value.

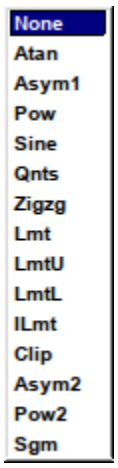

Figure 135: PADsynth Harmonic Content Editor Wave-Shaping Function

Values: `None*, Atan, Asym1, Pow, Sine, Qnts, Zigzg, Lmt, LmtU, LmtL, ILmt, Clip, Asym2, Pow2, Sgm`

The type of wave-shaping distortion has much influence on how the overtones are being placed. Sometimes, one gets a "fat" bass, and sometimes, high frequencies are added, making the sound "crystal clear".

Atan & Sigmoid. This is the default setting. It is an easy way to apply loudness to a wave without getting undesired high overtones. Thus, it can be used both for making instruments that sound like "real" ones, but also for electronic music. The transformation turns, roughly said, every amplitude into a square amplitude. Thus, sine, power, pulse and triangle turn into a usual square wave, while a saw turns into a phased square wave. A chirp wave turns into a kind of phase modulated square wave.

Quants ("Qnts") Quantization adds high overtones early. It can be seen as an unnatural effect, which is often used for electronic music. The transformation is a bit similar to building the lower sum of a wave, mathematically said. This means that the transformation effect turns an "endless high" sampled wave into only a few samples. The more distortion one applies, the fewer samples will be used. Indeed, this is equivalent to say that more input amplification is used. To see this, here is a small sample of code, where "ws" is the (correctly scaled amount of input amplification, and "n" the number of original samples.

If one turns on quantisation very high, one might be confused that, especially high notes, make no sound. The reason: High frequencies are "forgotten" if one samples with only few samples. Also, the sign of an amplitude can be forgotten. This behaviour might make some quantisations a bit unexpected.

Limiting ("Lmt*" and "Clip") Limiting usually means that for a signal, the amplitude is modified because it exceeds its maximum value. Overdrive, as often used for guitars, is often achieved by limiting: It happens because an amplifier "overdrives" the maximum amplitude it can deliver.

ZynAddSubFX has two types of limiting. Soft limiting, here as Lmt, means that the sound may not exceed a certain value. If the amplitude does so, it will simply be reduced to the limiting value. The overtones are generated in the lower frequencies first.

Hard limiting, is also called clipping and abbreviated Clip. This means that if the maximum is exceeded, instead of being constant at the limiting value, the original signal still has some influence on the output signal. Still, it does not exceed the limiting value. For ZynAddSubFX, a signal exceeding the limiting value will continue to grow "in the negative". This leads to overtones being generated on the full frequency band.

5. Wsh Wheel. Harmonic Editor Wave-shaping Wheel.

6. Filter. Harmonic Editor Filter. Sets the type of the harmonic filter.

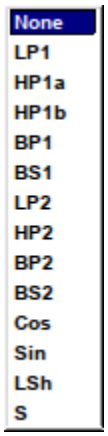

Figure 136: PADsynth Harmonic Content Filter

Values: `None*, LP1, HP1a, HP1b, BP1, BS1, LP2, HP2, BP2, BS2, Cos, Sin, LSh, S`

7. Filter Wheel 1. Harmonic Editor Filter, Wheel 1. The knob on the left sets one filter parameter, which is either the cutoff frequency, or, if the filter is a bandpass filter, the lower corner frequency. It is best to play with this knob with various kinds of filters selected from the filter drop-down list.

8. Filter Wheel 2. Harmonic Editor Filter, Wheel 2. The knob on the right sets, if the filter is a bandpass filter, the upper corner frequency. It is best to play with this knob with various kinds of filters selected from the filter drop-down list.

9. Filter p. Harmonic Editor Filter, p. If set, the filter is applied before waveshaping.

10. Mod. Harmonic Editor Modulation.

Figure 137: PADsynth Harmonic Content Editor Modulation

Values: `None*, Rev, Sine, Pow`

11. Mod. Wheel 1. Harmonic Editor Modulation Wheel 1. With the **Mod.** selection set to something other than `None`, this modifies one (unknown at this time) parameter of the modulation selection.

12. Mod. Wheel 2. Harmonic Editor Modulation Wheel 2. With the **Mod.** selection set to something other than `None`, this modifies one (unknown at this time) parameter of the modulation selection.

13. Mod. Wheel 3. Harmonic Editor Modulation Wheel 3. With the **Mod.** selection set to something other than `None`, this modifies one (unknown at this time) parameter of the modulation selection.

14. Sp.adj. Harmonic Editor Spectrum Adjust. Adjust the spectrum of the waveform.

RMS normalize. Enables the RMS normalization method (recommended); this keeps the same loudness regardless the harmonic content.

Below are the harmonics and their phases. One can use them to add to oscillator harmonics that has the waveform of the base function. Increasing the number of harmonics has virtually no effect on CPU usage. Right click to set a harmonic/phase to the default value.

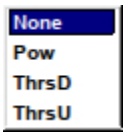

Figure 138: PADsynth Harmonic Content Editor Spectrum Adjust

Values: `None*, Pow, ThrsD, ThrsU`

15. Sp.adj. Wheel. Harmonic Editor Spectrum Adjust Wheel.

11.2.6.4 PADsynth / Harmonic Structure / Change / Harmonic

1. **Harmonics Amplitude**
2. **Harmonics Bandwidth**
3. **Harmonics Scrollbar**
4. **Harmonic Shift**
5. **Harmonic Shift R**
6. **Harmonic Shift preH**
7. **Adpt.Harm.**
8. **Adpt.Harm. Slider**
9. **Adpt.Harm. baseF**
10. **Adpt.Harm. pow**
11. **Clear**
12. **Sine**
13. **C**
14. **P**
15. **Close**

16. Harmonics Amplitude. Harmonics Amplitude. Provides 128 sliders for the amplitude of harmonics.

17. Harmonics Bandwidth. Harmonics Bandwidth. Provides 128 sliders for the bandwidth of harmonics.

18. Harmonics Scrollbar. Harmonics Scrollbar.

19. Harmonic Shift. Harmonics Shift.

20. Harmonic Shift R. Harmonics Shift Reset. Pressing this button resets the **Harmonic Shift** value to zero (0).

21. Harmonic Shift preH. Harmonics Shift preH. If set, applies the harmonic shift before the filtering and waveshaping.

22. Adpt.Harm. Adaptive Harmonics. Changes the type of the adaptive harmonics. (The tooltip spells "adaptive" wrong.)

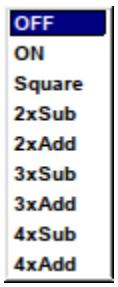

Figure 139: PADsynth Adaptive Harmonic Type

Values: `OFF*, ON, Square, 2xSub, 2xAdd, 3xSub, 3xAdd, 4xSub, 4xAdd`

23. Adpt.Harm. Slider. Adaptive Harmonics Slider. If something other that `OFF` or `ON` is selected, then this slider changes the waveform appearance. Even more informative is the change in the spectrum that is shown. Obviously, this setting is something to play with while listening to the waveform.

24. Adpt.Harm. baseF. Adaptive Harmonics Base Frequency. If something other that `OFF` is selected, then this knob changes the waveform appearance. Even more informative is the change in the spectrum that is shown. Obviously, this setting is something to play with while listening to the waveform.

25. Adpt.Harm. pow. Adaptive Harmonics Power. If something other that `OFF` is selected, then this knob changes the waveform appearance. Even more informative is the change in the spectrum that is shown. Obviously, this setting is something to play with while listening to the waveform.

26. Clear. Harmonics Clear. Clears the harmonics settings.

27. Sine. Harmonics Sine. The user is prompted to "Convert to sine?" This seems simply to reset the base function to a sine wave.

28. C. Harmonics Copy.

29. P. Harmonics Paste.

30. Close. Harmonics Close.

11.3 PADsynth / Envelopes and LFOs

This dialog is reached by click on the **Envelopes LFOs** tab of the PADsynth parameters dialog. This tab is next to the **Harmonic Structure** tab.

It consists of nothing but stock user-interface elements that are described elsewhere in this manual.

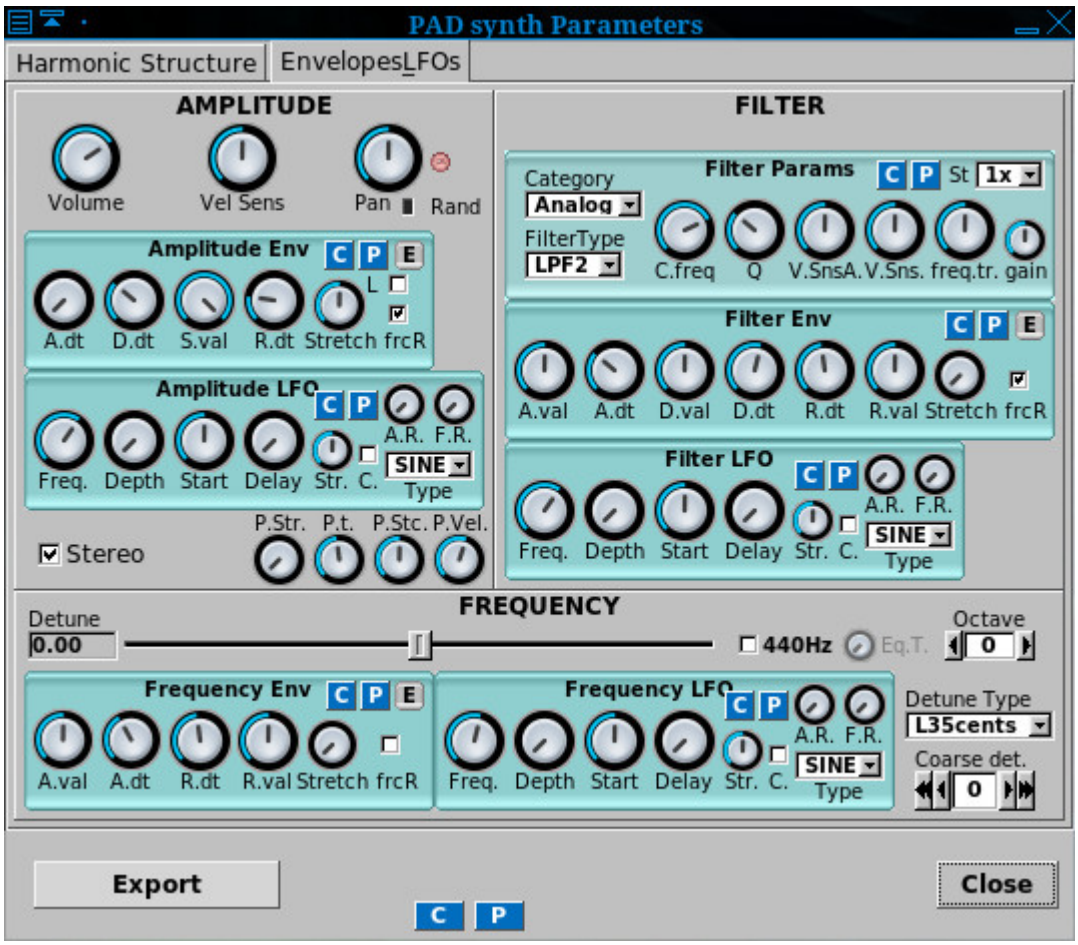

Figure 140: PADSynth Parameters, Envelopes and LFOs

1. **AMPLITUDE**
2. **FILTER** (section)
3. **FREQUENCY** (section)
4. **Export**
5. **C**
6. **P**
7. **Close**

31. AMPLITUDE. See section 10.1 "ADDsynth / AMPLITUDE" on page 135. This stock dialog section provide volume, velocity sensing, panning, an amplitude envelope sub-panel, and an amplitude LFO sub-panel.

32. FILTER. See section 10.2 "ADDsynth / FILTER" on page 137.

33. FREQUENCY. See section 10.3 "ADDsynth / FREQUENCY" on page 137.

34. Export. Very similar to figure 131 "Harmonics Structure Export Dialog" on page 159.

35. C. The stock copy dialog.

36. P. The stock paste dialog.

37. Close. Close.

12 SUBsynth

The *Yoshimi* SUBsynth dialog is yet another complex dialog, this time for creating a subtractive-synthesis instrument, "SUBsynth" or "SUBnote" is a simple engine which makes sounds through subtraction of harmonics from white noise. [25]

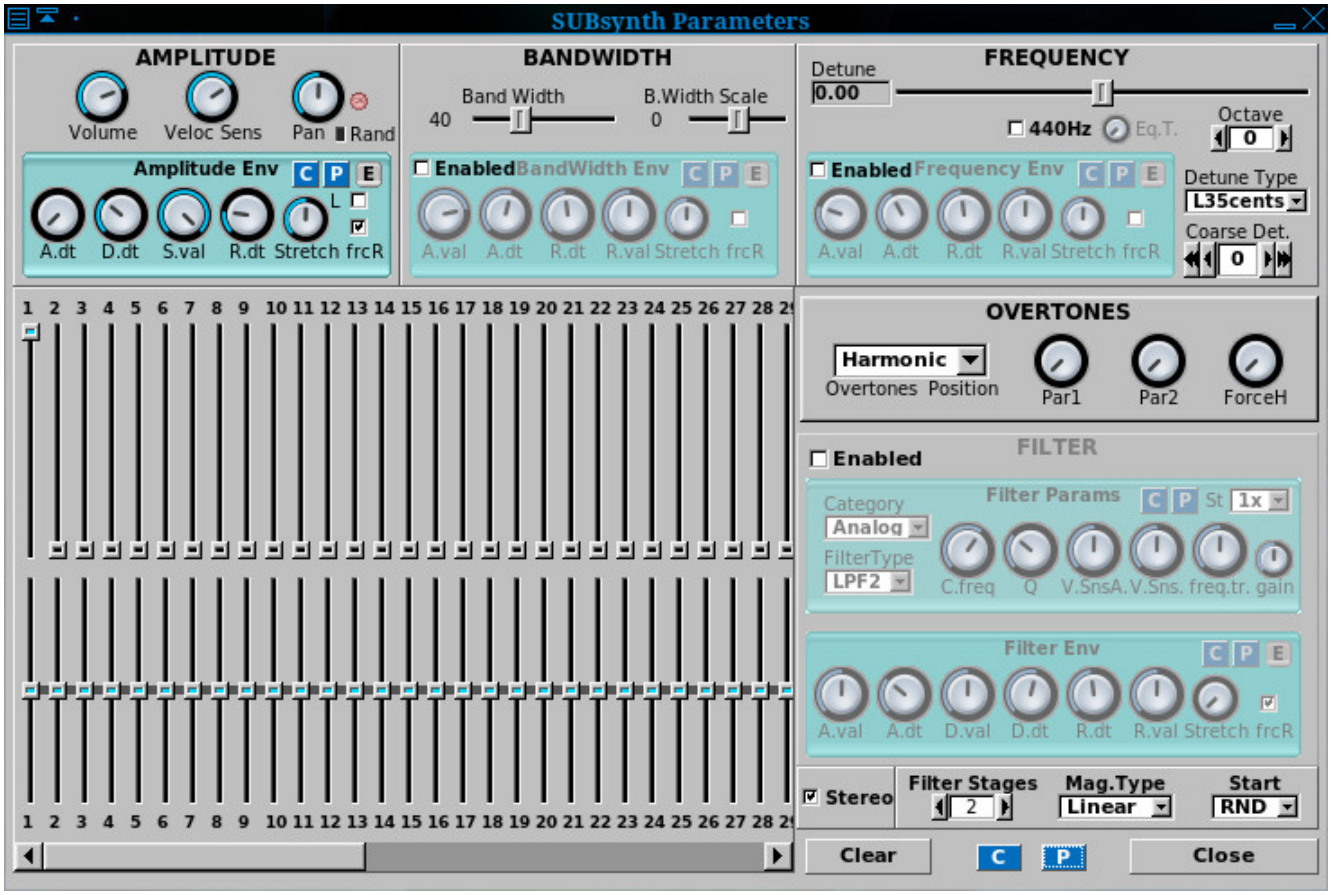

Figure 141: SUBsynth Edit Dialog

This dialog, though very complex, consists of a number of stock sections that are described elsewhere in this manual. Some descriptions are repeated here, though.

1. **AMPLITUDE** (section)
2. **BANDWIDTH** (section)
3. **FREQUENCY** (section)
4. **OVERTONES** (section)
5. **FILTER** (section)
6. **Harmonics** (section)
7. **Clear**
8. **C**
9. **P**
10. **Close**

12.1 SUBsynth / AMPLITUDE

1. **Volume**
2. **Vel Sens**
3. **Pan**
4. **Rand**
5. **Reset (panning)** (red button)
6. **Amplitude Env** (stock sub-panel)

1. Volume. SUBsynth Volume.

Values: `1 to 127, 64*`

2. Vel Sens. Velocity Sensing function, rightmost/max to disable.

Values: `1 to 127, 64*`

3. Pan. Global panning, leftmost/zero gives random panning.

Values: `1 to 127, 64*`

4. Rand. Indicator for activation of random panning.

5. Reset (panning). Reset Panning.

6. Amplitude Env. Amplitude Envelope. See section ..... for this stock sub-panel.

12.2 SUBsynth / BANDWIDTH

1. **BandWidth**
2. **B.Width Scale**
3. **Bandwidth Env**

1. BandWidth. SUBsynth Bandwidth. Sets the bandwidth of each harmonic.

Values: `1 to 127, 40*`

2. B.Width Scale. SUBsynth Bandwidth Scale. Sets how the bandwidth of each harmonic is increased according to the frequency. The default (0) increases the bandwidth linearly according to the frequency. This setting is kind of a "frequency stretch" parameter.

Values: `-64 to 63, 0*`

3. Bandwidth Env. SUBsynth Bandwidth.

1. **Enabled**
2. **A.val**
3. **A.dt**
4. **R.dt**
5. **R.val**
6. **Stretch**
7. **frcR**
8. **C**
9. **P**
10. **E**

1. Enabled. Enable the panel.

2. A.val. Attack value. We need to figure out what this means.

Values: `0 to 127, 64*`

3. A.dt. Attack duration. Attack time.

Values: `0 to 127, 40*`

4. R.dt. Release time.

Values: `0 to 127, 60*`

5. R.val. Release Value. Actually present only on the Frequency Env sub-panel.

Values: `0 to 127, 64*`

6. Stretch. Bandwidth Stretch. On lower notes make the bandwidth lower.

Values: `0 to 127, 64*`

7. frcR. Forced release. If this option is turned on, the release will go to the final value, even if the sustain level is not reached.

Also present in this sub-panel are the usual **C**opy and **P**aste buttons that call up a copy-parameters or paste-parameters dialog, as well as a button to bring up the editor window.

Values: `Off, On*`

12.3 SUBsynth / FREQUENCY

1. **Detune**
2. **FREQUENCY Slider**
3. **440Hz**
4. **Eq.T**
5. **Octave**
6. **Detune Type**
7. **Coarse Det.**
8. **Frequency Env**

1. Detune. Frequency Detune Indicator

2. FREQUENCY Slider. Frequency Slider.

Values: `-35 to 34.99`

3. 440Hz. Frequency 440Hz. Fixes the base frequency to 440Hz. One can adjust it with detune settings.

4. Eq.T. Frequency Equalize Time. Sets how the frequency varies according to the keyboard. Set to the leftmost setting for a fixed frequency.

5. Octave. Frequency Octave. Octave Shift.

6. Detune Type. Frequency Detune Type. Sets the "Detune" and "Coarse Detune" behavior

7. Coarse Det. Frequency Coarse Detune, "C.Detune".

8. Frequency Env. Frequency Envelope Stock Sub-Panel.

1. **Enable**
2. **A.value** or **A.val**

3. **A.dt**
4. **R.dt**
5. **R.val**
6. **Stretch**
7. **frcR**
8. **C**
9. **P**
10. **E**

See section ....

12.4 SUBsynth / OVERTONES

The harmonics settings controls the harmonic intensities/relative bandwidth. Moving the sliders upwards increases the relative bandwidth. Please note that, if one increases the number of harmonics, the CPU usage increases. Right click to set the parameters to default values.

1. **Overtones Position**
2. **Par1**
3. **Par2**
4. **ForceH**

1. Overtones Position. Subsynth Overtones Position.

Values: `Harmonic, ShiftU, ShiftL, PowerU, PowerL, Sine, Power, Shift`

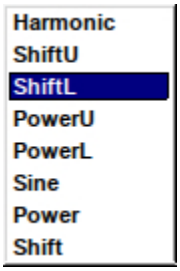

Figure 142: Harmonic Type Dropdown

2. Par1. Subsynth Overtones Par1.

Values: `0 to 127`

3. Par2. Subsynth Overtones Par2.

Values: `0 to 127`

4. ForceH. Subsynth Overtones ForceH.

Values: `0 to 127`

12.5 SUBsynth / FILTER

1. **Enabled**
2. **Filter Params** (stock sub-panel)
3. **Filter Env** (stock sub-panel)

4. **Stereo**
5. **Filter Stages**
6. **Mag. Type**
7. **Start**

1. Enabled. SUBsynth Filter Enabled.

2. Filter Params. Filter Params. See section 6.2.5 "Filter Parameters User Interface" on page 74, which describes this stock sub-panel.

3. Filter Env. Filter Params. See section 6.5.5 "Envelope Settings for Filter" on page 89, which describes this stock sub-panel.

4. Stereo. SUBsynth Stereo. Make the instrument stereo. The CPU usage goes up about 2 times. This item isn't really a **FILTER** item, it is just located in that same area.

5. Filter Stages. Filter Stages. Filter Order. Sets the number of filter stages applied to white noise. This parameter affects the CPU usage.

Values: `0, 1, 2*, 3, 4, 5`

6. Mag. Type. Magnitude Type. Sets the type of magnitude settings (linear versus dB values)

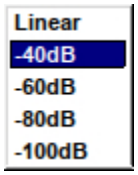

Figure 143: SUBSynth Magnitude Type Dropdown

Values: `Linear, -40dB, -60dB, -80dB, -100dB`

7. Start. Start Type. How to start the filters.

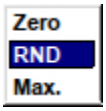

Figure 144: SUBsynth Start Type

Values: `Zero, RND, Max.`

12.6 SUBsynth / Harmonics

This section consists of 64 sliders to control the amplitude of the narrow noise band at a given harmonic, and 64 sliders to control the bandwidth of each band.

The top row of SUBsynth sliders sets the *relative* amplitude. This used of the word "relative" is an important distinction, as the overall level of the output is normalised; all actual levels will be dependent on whichever is the highest.

The bottom row sets the bandwidth of each harmonic. If one has just the fundamental, and drops the bandwith to the minimum, one gets very nearly a sinewave. Set it to maximum and it is very obviously filtered noise.

13 Kit Edit

The *Yoshimi* Kit dialog is a dialog for creating a set of drums or layered instruments. It provides a way to use individual voices and synth blocks to create drumlike sounds, or complex layered sounds. Within this window one can create drum kits, layered instruments, or one can combine more instruments into one instrument.

Is this true of *Yoshimi*?:

Item 0 is a special type: it cannot be disabled (but it can be muted), to edit it one must use "ADs edit" or "SUBs edit" from the part window.

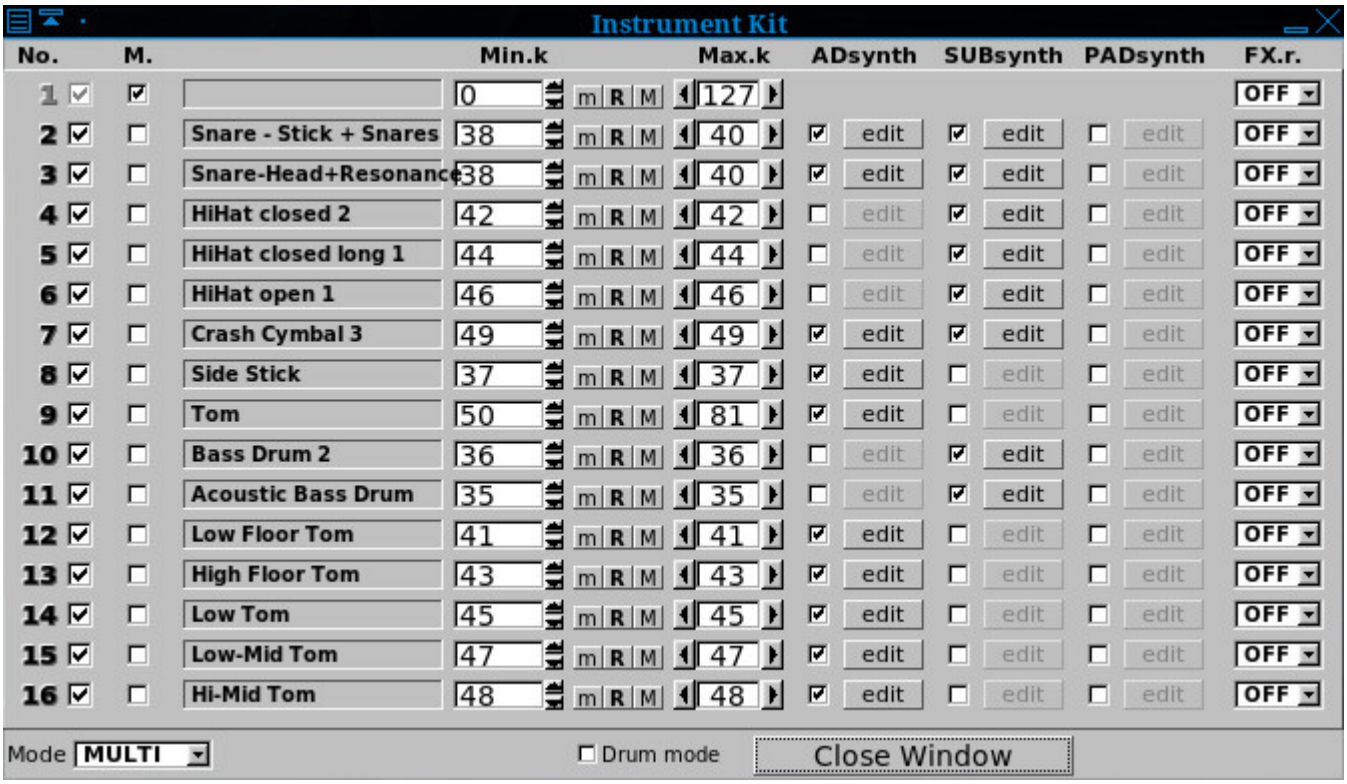

Figure 145: Kit Edit Dialog

1. **Rows 1 to 16.** This dialog contains 16 identical rows containing the following elements, in the order given:
    1. **No.**
    2. **Enable**
    3. **M.**
    4. **Instrument Name**
    5. **Min.k**
    6. **m** (set minimum note)
    7. **R** (reset default note range)
    8. **M** (set maximum note)
    9. **Max.k**
    10. **ADsynth**
        1. **Enable**

        2. **edit**
    11. **SUBsynth**
        1. **Enable**
        2. **edit**
    12. **PADsynth**
        1. **Enable**
        2. **edit**
    13. **FX.r**
2. **Mode**
3. **Drum mode**
4. **Close Window**

Some items described in ZynAddSubFX that aren't seen in any diagrams:

1. Kit Mode. Enable the kit mode.
2. Protect the kit. when loading an instrument, only item 0 will be changed, Other items will remain untouched. This allows one to combine more instruments. If one wants to add more instruments to the kit, one must copy the item 0 to another item, because the item 0 will be replaced. If one loads master settings or clearx the instrument/master setting, the kit is cleared .
3. Swap/Copy. Swap two items or copy a item to other item.

1. No. Kit Row Number. Kit Item Number. A simple label to indicate the instrument number in the kit.

2. Enable. Kit Row Enable.

Value: `Off*, On`

3. M. Kit Row "M". Mute an item of the kit.

4. Instrument Name. Kit Instrument Name.

5. Min.k. Kit Instrument Minimum Key. Sets the minimum key of the item of the kit.

6. m. Sets the minimum note of this instrument to value of the last note pressed.

7. R. Resets the minimum and maximum notes to their default values.

8. M. Sets the maximum note of this instrument to value of the last note pressed.

9. Max.k. Kit Instrument Maximum Key. Sets the maximum key of the item of the kit.

10. ADsynth. Kit ADDsynth. A checkbox is provided to enable/disable this synth component, and an edit button is provided to edit the component.

11. SUBsynth. Kit SUBsynth. A checkbox is provided to enable/disable this synth component, and an edit button is provided to edit the component.

12. PADsynth. Kit PADsynth. A checkbox is provided to enable/disable this synth component, and an edit button is provided to edit the component.

13. FX.r. Kit Effect. Chooses the Part Effect (PartFX) to process the item (OFF means that is unprocessed).

Values: `OFF, FX1, FX2, FX3`

14. Mode. Kit Mode.

- **OFF** means no kit is enabled, so one only has the Add, Sub, and Pad sounds in the Instrument Edit window.

- **MULTI** means all the kit items will sound together regardless of their note ranges.
- **SINGLE** means only the lowest numbered item will sound in a given note range. There will be no overlap.

For example: Item 0 has **Min.k** set to 0 and **Max.k** set to 60, and Item 1 has **Min.k** set to 40 and **Max.k** set to 127.

In **SINGLE mode**, only Item 0 will sound in the note range 0 to 60, and Item 1 will sound in the range 61 to 127.

In **MULTI** mode, only Item 0 will sound in the range 0 to 40, both items will sound from 41 to 60, and only Item 1 will sound from 61 to 127.

Values: `OFF*, MULTI, SINGLE`.

15. Drum mode. Kit Drum Mode. If drum-mode is set, then microtonal tuning is ignored for this kit, otherwise it could make drum sounds very unpredictable!

16. Close Window. Close.

14 Banks Collection

In this section, we attempt to collect and summarize all of the existing banks for *Yoshimi* and *ZynAddSubFX* that we can find. Many of them are supplied by the two projects.

Between all of the collections, there is a large amount of duplication. There is also semi-duplication, with slight variations on the same basic instrument. Various Linux distributions which package *ZynAddSubFX* and *Yoshimi* might add some banks to their versions of these packages. Thus, there are far more sound settings than we can discuss and categorize.

One thing we're looking for is a good General MIDI (GM) bank for *Yoshimi*. As part of our *Yoshimi Cookbook* [3], we include a basic General MIDI bank for. However, there are number of patches with no good implementation in it.

14.1 Yoshimi Banks

Yoshimi comes with the following banks, which may be found in `/usr/share/yoshimi/banks` as installed by the installer. In this case, it is the Debian installer.

1. **Arpeggios**. Also in ZynAddSubFX.
2. **Bass**. Also in ZynAddSubFX.
3. **Brass**. Also in ZynAddSubFX.
4. **chip**.
5. **Choir_and_Voice** Also in ZynAddSubFX, slightly different bank name.
6. **Drums**. Also in ZynAddSubFX, but with only one drum kit included.
7. **Dual**. Also in ZynAddSubFX.
8. **Fantasy**. Also in ZynAddSubFX.
9. **Guitar**. Also in ZynAddSubFX.
10. **Misc**. Also in ZynAddSubFX.
11. **Noises**. Also in ZynAddSubFX.
12. **Organ**. Also in ZynAddSubFX.
13. **Pads**. Also in ZynAddSubFX.

14. **Plucked**. Also in ZynAddSubFX.
15. **Reed_and_Wind**. Also in ZynAddSubFX, slightly different bank name.
16. **Rhodes**. Also in ZynAddSubFX.
17. **Splited**. Also in ZynAddSubFX, slightly different bank name.
18. **Strings**. Also in ZynAddSubFX.
19. **Synth**. Also in ZynAddSubFX.
20. **SynthPiano**. Also in ZynAddSubFX.
21. **The_Mysterious_Bank**. Also in ZynAddSubFX, slightly different bank name. ZynAddSubFx has three more mysterious banks (see next section).
22. **Will_Godfrey_Collection**.
23. **Will_Godfrey_Companion**.

14.2 Additional ZynAddSubFX Banks

ZynAddSubFX comes with the following banks, which may be found in the source code [28] or installation packages of this project. *ZynAddSubFX* has some of the same banks (as far as we can tell) as *Yoshimi*, but with the following additions:

1. **Companion**.
2. **Cormi_Noise and Cormi_Sound** [4].
3. **Laba170bank**.
4. **olivers-100**. Some very good instruments, including sitar and steel drums.
5. **the_mysterious_bank**.
6. **the_mysterious_bank_2**.
7. **the_mysterious_bank_3**.
8. **the_mysterious_bank_4**.

14.3 Additional Banks

Here are some additional banks we have found, or have built ourselves. It often happens that, later on, a site is no longer available. Or we forget from whence we got the banks. In these cases, the banks are stored in the `contrib/banks` directory of this project.

1. **Alex_J** The site seems to be gone/expired. So one will find these in the "contrib/banks" directory for safekeeping.
2. **Bells** We have no idea where we got this one. Lost track of that information.
3. **C_Ahlstrom** These are mine, but not yet made into a systematic bank. They are included with this document.
4. **Chromatic Percussion** Not sure where we got this at this time.
5. **Drums_DS** Not sure where we got this at this time.
6. **Electric Piano** Not sure where we got this at this time.
7. **Flute** Not sure where we got this at this time.
8. **folderol collection** [6], also found at [24].
9. **Internet Collection** Not sure where we got this at this time.
10. **Leads** Not sure where we got this at this time.
11. **Louigi_Verona_Workshop** The site seems to be gone/expired. So one will find these in the "contrib/banks" directory for safekeeping.
12. **Misc Keys** Not sure where we got this at this time.

13. **mmxgn Collection** [10]
14. **Piano** Not sure where we got this at this time.
15. **RB Zyn Presets** Not sure where we got this at this time.
16. **Vanilla** See [24] for this bank, and for some demonstration files of *ZynAddSubFX* sounds, and some other nice links.
17. **VDX** Not sure where we got this at this time.
18. **x31eq.com** [16]
19. **XAdriano Petrosillo** Not sure where we got this at this time.
20. **Zen Collection** Not sure where we got this at this time.

15 Non-Registered Parameter Numbers

This section comes from the source-code documentation file `Zyn nrpn.txt` or the *ZynAddSubFx* online manual [25] and the *Using_NRPNS.txt* document that accompanies the *Yoshimi* source code.

Yoshimi implements System and Insertion effects control in a manner compatible with *ZynAddSubFX*. As with all *Yoshimi*'s NRPNs, the controls can be sent on any MIDI channel.

15.1 NRPN / Basics

NRPN stands for "Non Registered Parameters Number". NRPNs can control all System and Insertion effect parameters. Using NRPNs, *Yoshimi* can now directly set some part values regardless of what channel that part is connected to. For example, one may change the reverb time when playing to keyboard, or change the flanger's LFO frequency. The controls can be sent on any MIDI channel (the MIDI channels numbers are ignored).

The parameters are:

- **NRPN MSB** (coarse) (99 or 0x63) sets the system/insertion effects (4 for system effects or 8 for insertion effects). We abbreviate this value as `Nhigh`.
- **NRPN LSB** (fine) (98 or 0x62) sets the number of the effect (first effect is 0). We abbreviate this value as `Nlow`.
- **Data entry MSB** (coarse) (6) sets the parameter number of effect to change (see below). We abbreviate this value as `Dhigh`.
- **Data entry LSB** (fine) (26) sets the parameter of the effect. We abbreviate this value as `Dlow`.

One must send NRPN coarse/fine before sending Data entry coarse/fine. If the effect/parameter doesn't exists or is set to none, then the NRPN is ignored.

It's generally advisable to set NRPN MSB before LSB However, once MSB has been set one can set a chain of LSBs if they share the same MSB. The data CCs associated with these are 6 for MSB and 38 for LSB. Only when an NRPN has been established can the data values be entered (they will be ignored otherwise). If a supported control is identified, these data values will be stored locally (if needed) so that other NRPNs can be set. Whenever either byte of the NRPN is changed, the data values will be cleared (but stored settings will not be affected). If either NRPN byte is set to 127, all data values are ignored again.

In *Yoshimi* NRPNs are not themselves channel-sensitive, but the final results will often be sent to whichever is the current channel. *Yoshimi* also supports the curious 14-bit NRPNs, but this shouldn't be

noticeable to the user. In order to deal with this, and also some variations in the way sequencers present NRPNs generally, if a complete NRPN is set (i.e. `Nhigh`, `Nlow`, `Dhigh`, `Dlow`), then the data bytes can be in either order, but must follow `Nhigh` and `Nlow`.

(In these notes, where practical we also list the 14 bit values in square brackets.)

After this, for running values, once `Dhigh` and `Dlow` have been set if one changes either of these, the other will be assumed. For example, starting with `Dhigh` = 6 and `Dlow` = 20:

Change `Dlow` to 15 and *Yoshimi* will regard this as a command `Dhigh` 6 + `Dlow` 16 Alternatively change `Dhigh` to 2 and *Yoshimi* will regard this as a command `Dhigh` 2 + `Dlow` 20. This can be useful but may have unintended consequences! If in doubt change either of the NRPN bytes and both data bytes will be cleared.

Additionally there is 96 for data increment and 97 for decrement.

Data increment and decrement operation enables one to directly change the data LSB by between 0 and 63. To change the MSB add 64 to cover the same range. Setting 0 might seem pointless, but it gives an alternative way to make an initial setting if one's sequencer doesn't play nice.

Although data increment and decrement are only active if a valid NRPN has been set, they are otherwise quite independent single CCs. For example:

```
Start Value       Command value   Result
----- -----       --------------  ------
LSB     5           inc 20      25
MSB     7           inc 68      11
LSB     128(off)    inc 1       1
MSB     126         dec 74      116
MSB     128(off)    dec 65      127
```

A small example (all values in this example are hex):

```
B0 63 08 // Select the insertion effects
B0 62 01 // Select the second effect (remember: the first is 00 and not 01)
B0 06 00 // Select the effect parameter 00
B0 26 7F // Change the parameter of effect to the value 7F (127)
```

WARNING: Changing of some of the effect parameters produces clicks when sounds passes thru these effects. We advise one to change only when the sound volume that passes through the effect is very low (or silence). Some parameters produce clicks when they are changed rapidly.

Here are the effects parameter numbers (for Data entry, coarse). The parameters that produces clicks are written in red and have (AC) after their entry (always clicks). The parameter that produces clicks only when they are changed fast are written in blue and have a (FC) after the entry (Fast Clicks). Most parameters have the range from 0 to 127. When parameters have another range, it is written as "(low...high)".

Here are the basic formats:

1. Send NRPN:

- MSB = 64 (same as for vectors)
- LSB = 0

2. Send Data MSB (6); all value ranges start from zero, not 1.
   - 0 : data LSB = part number
   - 1 : data LSB = program number
   - 2 : data LSB = controller number
   - 3 : data LSB = controller value
   - 4 : data LSB = part's channel number (15 to 127 disconnects the part from any channel)
   - 5 : data LSB = part's audio destination, one of 1 = main L&R; 2 = direct L&R; 3 = both; all other values are ignored
   - 7 : data LSB = main volume (not yet implemented)
   - 35 (0x23) : data LSB = controller LSB value (not yet implemented)
   - 39 (0x27) : data LSB = main volume LSB (not yet implemented)
   - Other values are currently ignored.

Other values are currently ignored by *Yoshimi*.

15.2 NRPN / Effects Control

15.2.0.1 Reverb

- 00 - Volume or Dry/Wet (FC)
- 01 - Pan (FC)
- 02 - Reverb Time
- 03 - Initial Delay (FC)
- 04 - Initial Delay Feedback
- 05 - reserved
- 06 - reserved
- 07 - Low Pass
- 08 - High Pass
- 09 - High Frequency Damping (64..127) 64=no damping
- 10 - Reverb Type (0..1) 0-Random, 1-Freeverb (AC)
- 11 - Room Size (AC)

15.2.0.2 Echo

- 00 - Volume or Dry/Wet (FC)
- 01 - Pan (FC)
- 02 - Delay (AC)
- 03 - Delay between left and right (AC)
- 04 - Left/Right Crossing (FC)
- 05 - Feedback
- 06 - High Frequency Damp

15.2.0.3 Chorus

- 00 - Volume or Dry/Wet (FC)
- 01 - Pan (FC)

- 02 - LFO Frequency
- 03 - LFO Randomness
- 04 - LFO Type (0..1)
- 05 - LFO Stereo Difference
- 06 - LFO Depth
- 07 - Delay
- 08 - Feedback
- 09 - Left/Right Crossing (FC)
- 10 - reserved
- 11 - Mode (0..1) (0=add, 1=subtract) (AC)

15.2.0.4 Phaser

- 00 - Volume or Dry/Wet (FC)
- 01 - Pan (FC)
- 02 - LFO Frequency
- 03 - LFO Randomness
- 04 - LFO Type (0..1)
- 05 - LFO Stereo Difference
- 06 - LFO Depth
- 07 - Feedback
- 08 - Number of stages (0..11) (AC)
- 09 - Let/Right Crossing (FC)
- 10 - Mode (0..1) (0=add, 1=subtract) (AC)
- 11 - Phase

15.2.0.5 AlienWah

- 00 - Volume or Dry/Wet (FC)
- 01 - Pan (FC)
- 02 - LFO Frequency
- 03 - LFO Randomness
- 04 - LFO Type (0..1)
- 05 - LFO Stereo Difference
- 06 - LFO Depth
- 07 - Feedback
- 08 - Delay (0..100)
- 09 - Left/Right Crossing (FC)
- 10 - Phase

15.2.0.6 Distortion

- 00 - Volume or Dry/Wet (FC)
- 01 - Pan (FC)
- 02 - Left/Right Crossing
- 03 - Drive (FC)
- 04 - Level (FC)
- 05 - Type (0..11)
- 06 - Invert the signal (negate) (0..1)

- 07 - Low Pass
- 08 - High Pass
- 09 - Mode (0..1) (0=mono,1=stereo)

15.2.0.7 EQ

- 00 - Gain (FC)

All other settings of the EQ are shown in a different way. The N represent the band ("B." setting in the UI) and the first band is 0 (and not 1), like it is shown in the UI. Change the "N" with the band one likes. If one wants to change a band that doesn't exist, the NRPN will be ignored.

- 10+N*5 - Change the mode of the filter (0..9) (AC)
- 11+N*5 - Band's filter frequency
- 12+N*5 - Band's filter gain
- 13+N*5 - Band's filter Q (bandwidth or resonance)
- 14+N*5 - reserved

Example of setting the gain on the second band in the EQ module:

- The bands start counting from 0, so the second band is 1 =<N=1.
- The formula is 12+N*5 =<12+1*5=17, so the number of effect parameter (for Data entry coarse) is 17.

15.2.0.8 DynFilter

- 0 - Volume
- 1 - Pan
- 2 - LFO Frequency
- 3 - LFO Randomness
- 4 - LFO Type
- 5 - LFO Stereo Difference
- 6 - LFO Depth
- 7 - Filter Amplitude
- 8 - Fitler Amplitude Rate Change
- 9 - Invert the signal (negate) (0..1)

Click behaviour of DynFilter has not yet been tested.

15.2.0.9 Yoshimi Extensions

If the Data MSB bit 6 is set (64) then Data LSB sets the effect type instead of a parameter number. This must be set before making a parameter change.

- 0 - Reverb
- 1 - Echo
- 2 - Chorus
- 3 - Phaser

- 4 - AlienWah
- 5 - Distortion
- 6 - EQ
- 7 - DynFilter

For Insert effects, if the Data MSB bit 5 is set (32) then Data LSB sets the destination part number. 127 is off and 126 is the Master Output. A complete example:

- 99 - 8 ˜insert effects
- 98 - 3 ˜number 4 (as displayed)
- 6 - 32 ˜set destination
- 38 - 126 ˜Master Out
- 99 - 8 *
- 98 - 3 *
- 6 - 64 ˜change effect
- 38 - 4 ˜Alienwah
- 99 - 8 *
- 98 - 3 *
- 6 - 0 ˜Dry/Wet
- 38 - 30 ˜value

Notes (*): these repeats are not needed as far as *Yoshimi* is concerned, but some sequencers get unhappy without them. Change just a parameter on an exisiting system effect:

- 99 - 4 ˜system effects
- 98 - 0 ˜the first effect
- 6 - 1 ˜Pan
- 38 - 75 ˜value

15.3 NRPN / Dynamic System Settings

Almost all dynamic setup (i.e. that doesn't require a restart) can now be done via NRPNs, so a MIDI file can manage *Yoshimi* starting from a pretty random state, and set up important features like Bank and Program change behavior and the number of available parts.

In parallel with this setup, there is a command to list all of these settings. One can also list the available bank roots, the banks in any root, and instruments in any bank, along with their numeric IDs. These IDs can then be used with normal MIDI CCs to get exactly the instrument you want at any time.

This arrangment looks positively steam-punk, but is actually very easy to use, requiring only a command line interface and any utility that can send MIDI CCs. NRPNs aren't special. They are simply a specific pattern of CCs. *Yoshimi*'s implementation is very forgiving, doesn't mind if you stop halfway through (will just get on with other things while it waits), and will report exactly what it is doing. So ...

... If *Yoshimi* has been started from the command line (but not necessarily in the no-GUI mode), all of the system settings that don't require a restart can now be viewed by sending the appropriate NRPN. Most of them can also be changed in this way.

To access this functionality, set NRPN MSB (CC 99) to 64 and NRPN LSB (CC 98) to 2 (8130).

After that send the following DATA values. Commands with LSB x don't actually use DATA LSB, but one still needs to send it (unless it has already been set by a previous command in this control group).

Table 2: Dynamic System Commands

DATA MSB	DATA LSB	Setting
2	LSB key	Set master key shift, 52<=key<=76 (-36 to +36)
7	LSB volume	Set master Volume 'volume'
64-79	LSB key	Set channel-based (n-64) key shift
100	LSB >63	Send reports to Reports window, otherwise to stderr.
108	LSB x [13824]	List settings of all vectors.
109	LSB x [13843]	List the following:
		Reports destination
		Current Root path
		Current Bank
		CC to control Root path change
		CC to control Bank change
		Accept Program change (enabled/disabled)
		Activate part when program changed (enabled/disabled)
		CC to control Extended Program change (instruments 128-159)
		Number of available parts
110	LSB x [13970]	List all root paths
111	LSB path [14224]	List all banks in Root ID 'path'; path=127 for current root)
112	LSB bank [14351]	List all instruments, current root, bank 'bank'
		(bank = 127 for current bank in current root)
113	LSB root	Set CC to control Root path change (root>119 disables)
114	LSB bank	Set CC to control Bank change (bank>119 disables)
115	LSB >63	Enable Program change otherwise disable
116	LSB >63	Enable activation of part when program changed
117	LSB extprog	Set CC control Extended program change (extprog>119 disables)
118	LSB parts	Set number of available parts (parts = 16, 32 or 64)
119	LSB x [15113]	Save all dynamic settings

16 Vector Control

This section comes from the source-code documentation file `doc/Vector_Control.txt`.

Vector load and save work from the command-line, for a complete vector set, with all mappings, instruments, etc. One can independently decide which channel to load and save from, so one can actually build up a vector set in (say) channel 3, then later decide to use it in channel 7. The vector settings file has the extension `.xvy` standing for *Xml / Vector / Yoshimi*.

16.1 Vector / Basics

Vector control is a way to control more than one part with the controllers. It is a little bit reminiscent of the "vector" control knob on the Yamaha PSS-790 consumer MIDI synthesizer. Vector control is only possible if one has 32 or 64 parts active. Setup is per MIDI channel, so one can have totally different vector behaviour on, say, channel 1 and channel 5.

Vector control has been extended so that there are four independent 'features' that each axis can control. One is fixed as *Volume* (if enabled) but the other three can be any valid CC, and can also be reversed. The vector 'sweep' CCs are split out very early in the MIDI chain, and the new CCs created are fed back in before any other processing. The result of this is that once we eventually get MIDI-learn implemented, the control possibilities will expand dramatically. *(Will notes: "sorry about the extreme delay :(")*

In vector mode parts will still play together but the vector controls can change their volume, pan, filter cutoff in pairs, controlled by user-defined CCs set up with NRPNs.

One must set the X axis CC before the Y axis, but if one doesn't set the Y axis at all, one can run just a single axis. If one has only 32 parts active, Y settings are ignored.

For example: parts 1 and 17 can be set as x1 & x2 (volume only) while parts 33 and 49 can be y1 & y2 (pan only).

Independently of this Parts 2 & 18 could use filter and pan from another CC.

16.2 Vector / Vector Control

Setting up vector control is currently done as follows. In the required channel send:

- NRPN MSB (99) set to 64
- NRPN LSB (98) set to 1 [8192]
- Data MSB (6) set mode:
  - 0 = X sweep CC
  - 1 = Y sweep CC
  - 2 = enable X features
  - 3 = enable Y features
  - 4 = x1 instrument (optional)
  - 5 = x2 instrument (optional)
  - 6 = y1 instrument (optional)
  - 7 = y2 instrument (optional)

Setting CC for X enables vector control; any value outside the above list disables it.

Data LSB (38) value to set features:

- 1 = Volume (fixed)
- 2 = Pan (the default)
- 4 = Filter Cutoff (Brightness, it is the default)
- 8 = Mod Wheel (the default)
- 0x12 = 18 = Reversed Pan
- 0x24 = 36 = Reversed Filter Cutoff
- 0x48 = 72 = Reversed Mod Wheel

The feature numbers are chosen so they can be combined. So, 5 would be Volume + Brightness and 19 would be Volume + Reversed Pan.

Setting the sweep CC for the X axis enables vector control. It also sets, but doesn't enable the default X axis features. Setting the sweep CC for the Y axis sets, but doesn't enable the default Y axis features. If one doesn't enable any features, not a lot will happen.

The feature numbers are chosen so they can be combined. So, 5 would be Volume + Brightness and 19 would be Volume + Reversed Pan.

Optional settings. The first part, the number, is the MSB value. The second part is the LSB, the parameter value to set. Note that the instrument IDs are for instruments in the current bank.

- 4 = x1 instrument ID
- 5 = x2 instrument ID
- 6 = y1 instrument ID
- 7 = y2 instrument ID
- 8 = set CC for X feature 2
- 9 = set CC for X feature 4
- 10 = set CC for X feature 8
- 11 = set CC for Y feature 2
- 12 = set CC for Y feature 4
- 13 = set CC for Y feature 8

The IDs are for instruments in the current bank. Any data MSB value outside the above list disables vector control. Sweep CCs and feature CCs are sanity-checked.

An Example. From channel 1, send the following CCs:

```
CC      Value
99       64
98        1
 6        0
38       14
98        1 *
 6        1
38       15
98        1 *
 6        2
38        1
98        1 *
 6        3
38        2
```

This sequence will set up CC 14 as the X axis incoming controller, and CC 15 as the Y axis incoming controller, with X set to volume control and Y set to pan control.

One can either go on with the NRPNs to set the instruments (this will load and enable instruments from the current bank), or enable and load them by hand. For channel 1 this would be part 1 and 17 for X and part 33 and 49 for Y.

The (*) CCs ensure that the data bytes are reset each time. This is not really necessary for the earlier commands, but should be done if one sets the instruments with NRPNs as well, otherwise one will try to set them twice.

16.3 Vector / Command Line

This section covers material that could be in the command-line section (see section 17 "The Yoshimi Command Line Interface" on page 188), but is really too detailed to cover there. The examples here, to set up vectors from the command line, are provided by Will.

Assuming we want just a single axis on channel 1 (which is channel 2 in the GUI), first we need to make sure we have enough parts available:

```
yoshimi> set available 32
Available parts set to 32
```

The next command must aways be the first command, as everthing else depends on it. It's the command that *enables* vector control. The `x` token denotes the "x axis", and the `cc` token, followed by 14, is the incoming sweep CC (control change) that will vary the features one sets.

```
yoshimi > set vector 1 x cc 14
Vector channel set to 1
```

Note that, according to the list of MIDI CC's at `http://nickfever.com/music/midi-cc-list`, CC 14 is undefined, normally. It is thus available for *Yoshimi* to assign for its own purpose.

There are four vector features currently available:

- **1** is fixed as *volume*.
- **2** is *pan* by default.
- **3** is *brightness* by default.
- **4** is *modulation* by default.

We will select *volume* for this example. Let's enable this feature:

```
yoshimi Vect Ch 1 X > set features 1 enable
Set X features 1 en
```

Next, one needs to set the instruments that will be used. The instruments can only be selected from the instruments in the current bank. Therefore, assuming the current bank is the "*Will Godfrey Companion*", let's set up two instruments:

```
yoshimi Vect Ch 1 X > set program left 20
Loaded 20 "Bubbles" to Part 1

yoshimi Vect Ch 1 X > set program right 120
Loaded 120 "Ghost Ensemble" to Part 16
```

The `left` token merely assigns instrument 20 to a "virtual" left side of the X axis, and the `right` token assigns instrument 20 to a "virtual" right side of the X axis.

(Chris asks: How did the part numbers 1 and 16 come about? What are the rules? Why are 32 parts needed, if only 1 to 16 are involved? Why do we need to have 64 parts when we add the Y axis below? Can we set intermediate values between "left" and "right" and "up" and "down" to get some really weird morphs?)

If one now sweeps the the controller assigned to CC 14, the sound will morph between these two instruments.

To continue on to using the other axis as well, one needs to have 64 parts available:

```
yoshimi Vect Ch 1 X > /set available 64
Available parts set to 64
```

Note the slash, which lets the user immediately access the topmost command level, where the "available parts" setting can be performed. Then:

```
yoshimi > set vector y cc 15
Vector 1 Y CC set to 15
```

This command sets up the Y axis to be controlled by MIDI CC 15, which is, again, a CC that is normally undefined. We will use *panning* (feature 2) for this vector, which is defined on the Y axis:

```
yoshimi Vect Ch 1 Y > set features 2 enable
Set Y features 2 en
```

Analogous to the "left" and "right" virtual directions used above for the X axis, the Y axis used the "up" and "down" virtual directions:

```
yoshimi Vect Ch 1 Y > set program down 107
Loaded 107 "Angel Harp" to Part 32

yoshimi Vect Ch 1 Y > set program up 78
Loaded 78 "Brassy Flutter" to Part 48
```

Notice that the directions left, right, up, and down match the directions provided by a traditional joystick.

So we have now set up a vector sound where MIDI CC 14 morphs the sound through a continuous linear combination of two different instruments, and MIDI CC 15 morphs the sound between two other instruments. One can then save one's cool vector sound to a file:

```
yoshimi Vect Ch 1 Y > save vector CoolSound
Saved channel 1 Vector to CoolSound
```

The file extension for the save vector sound file is `.xvy`, and this extension is added automatically. The final name of the file is `CoolSound.xvy`.

(Chris asks: Where is this file saved? Is there a way to modify this location?)

At any time one can reload this vector sound file from the command-line:

```
yoshimi> load vector channel 0 CoolSound
Loaded Vector CoolSound to channel 0
```

If there is no channel number provided, then the vector sound will be loaded to the same channel as it was saved from:

```
yoshimi> load vector CoolSound
Loaded Vector CoolSound to source channel
```

17 The Yoshimi Command Line Interface

Applies through version: 1.3.9 May 2016

Yoshimi provides a "no GUI" or "command-line" mode of operation where some aspects of the application can be controlled via textual commands. This mode is useful for blind people, for example. To access this mode, add the `-i` or `--no-gui` command-line option when starting *Yoshimi* on the command-line. But note that, when starting *Yoshimi* on the command-line, the "command-line" mode of operation is available at the same time as the GUI, as well.

One of the main features of recent *Yoshimi* releases is improved non-GUI accessibility. In a command line environment, almost all the 'running' commands are available, but none of the instrument editing ones are... yet! One can decide what MIDI/audio setup is wanted, list and set roots and banks, load instruments into any part, change a part's channel, set main volume and key shift, and set up vector control. A number of first-time defaults have been changed to make this feature easier.

When starting from the command line, an argument can be included for a new root path to be defined to point to a set of banks fetched from elsewhere. This will be given the next free ID. A future upgrade will allow the ID to be set to any valid one when it is created, mirroring the GUI behaviour.

Once running, almost all dynamic setup (i.e. doesn't require a restart) can now be done within the terminal window. There is also extensive control of roots, banks, parts and instruments including the ability to list and set all of these.

Additional controls that are frequently taken for granted in the GUI, but otherwise get forgotten, are *master key shift* and *master volume*. The most important parts of vector control are exposed to the command line.

The command-line mode provides extensive error checking and feedback. Note the change in nomenclature from "Parameters" to "Patch Set", which is visible in the main screen, and also reflected in the command line. The prompt will always show what *command level* one is on, along with relevant information. At the CLI prompt, when effects are being managed, the preset number is also shown on the prompt so you'll typically see something like:

```
yoshimi part 2 FX 1 Rever-7 >
```

One will also get a confirmation message, but, for clarity, those are not included in the examples below. Here is an example session:

Starting from the `yoshimi` prompt:

```
yoshimi> s p 2 pr 107
yoshimi part 2 >
```

This command **s**ets **p**art number 2 to **pr**ogram number 107 from the *current* instrument bank. *Yoshimi* is now on part 2 as the current part (indicated by the prompt), and all subsequent commands will relate to this "level". At this level, one can change the current part simply with:

```
yoshimi part 2 > s 4
yoshimi part 4 >
```

Yoshimi is now on part number 4. Now set an effect:

```
yoshimi part 4 > s ef ty re
yoshimi part 4 FX 0 rever-0 >
```

This command **s**ets the part's **ef**fect 0 (implicit) to **ty**pe **re**verb.

Note that many settings parameters are optional, and if you omit them, either a default or last-used value will be assumed. Also, names are truncated to 5 characters so the prompt line doesn't get unmanageably long.

From here you can set a preset for this effect:

```
yoshimi part 4 FX 0 rever > s pre 3
```

Currently, the **pre**sets are not shown in the prompt, but one will get a confirmation message.

Settings that follow in a direct command "path" through several levels can be made all at once, and one will be left at the appropriate level. Thus, summarizing some of the above commands:

```
s p 4 ef 2 ty re
yoshimi part 4 FX 2 rever >
```

One cannot combine `type` and `preset` as they are both at the same level. To go back one level, use the ".." command:

```
yoshimi part 4 FX 2 rever > ..
yoshimi part 4 >
```

To go back to the top command level, use the "/" command:

```
yoshimi part 4 > /
yoshimi >
```

These two special level-movement commands can also be put on the front of any other command. Starting where we were before:

```
yoshimi part 4 FX 2 rever > .. s vol 70
yoshimi part 4 >
```

Part 4 volume is now at 70, and *Yoshimi* is once again at the "part level", not the "part FX level". Also note that the space after the ".." is optional.

The help menus and lists are also partially context sensitive. This feature should help avoid clutter and confusion.

As well as an immediate history, *Yoshimi* maintains a single command history file for all instances of *Yoshimi* that records any non-duplicated loads or save. Thus, provided one makes a normal command-line exit, the last commands will be available on the next run of *Yoshimi*.

Originally this section described the currently implemented commands, but as the command set is very much a moving target, it is simpler to just ask one to run *Yoshimi* and type the "?" command.

Commands with "*" in the description need the setup to be saved, and *Yoshimi* restarted to be activated.

Note that *Yoshimi*'s command-line can also load and save states, patchsets, and scales, and can list recent histories. Vector load and save is also supported from the command-line. That's a complete vector set, with all mappings, instruments, etc. One can independently decide which channel to load and save from, so that one can actually build up a vector set in (say) channel 3, then later decide to use it in channel 7. It has the extension `.xvy`, standing for "Xml/Vector/Yoshimi". It will eventually be integrated with the saved states.

Another small detail is that all of the minimum command-line abbreviations are now **C**apitalised in the help list.

More features will be added, and the organisation of them may be changed slightly. If any configuration settings are changed, either at the command-line or in the graphical user-interface, one will be given a warning when exiting, with the option to continue running so one can save the changes.

17.1 Command Level

A command level is simply a position in the hierarchy of commands that cover some aspect of *Yoshimi* functionality.

The levels that currently exist are:

- **Top Level**
- **System Effects**
- **Insertion Effects**
- **Part**
- **Part Effects**
- **Vector**

Ones that we're pretty sure will be added are:

- **Scales (microtonal)**
- **Controllers**
- **Addsynth**
- **Addsynth Voice**
- **Subsynth**
- **Padsynth Harmonics**
- **Padsynth Envelopes**

Any level that has a direct numerical content will be changeable simply with "set (n)" once you are at that level. The level will, of course, be indicated by the text in the *Yoshimi* prompt.

For example, one can have 0 to 15 vector channels, so from the *Top* level, the following command will return the default (0 or the last-used number):

```
set vector
s ve
```

Given this level (the *Vector* level), the following commands...

```
set 5
s 5
```

...will then switch to vector channel 5. However, at the start, one could have gone straight there with:

```
set vector 5
s ve 5
```

A more detailed discussion of command-line vector control is presented in section 16.3 "Vector / Command Line" on page 186.

17.2 Command Table

When running from the command line, these commands (see table 3 "Yoshimi Text Commands, Part 1" on page 193and table 4 "Yoshimi Text Commands, Part 2" on page 194) can be entered after the 'up and running' message. The commands are *not* case-sensitive. The commands can be abbreviated to the first three letters of each command, or, in some cases, just one letter. The commands that are available depend on the current "context" of the command line. There are a group of commands that are always available. These are

- `?` or `help`
- `List`
- `RESet`
- `EXit`

Apart from these basic commands, the command line works on a system of context levels, and normally only the commands relevant to that "level" will be available.

The brief descriptions in the following table can be obtained using the "help" command in the *Yoshimi* command-line mode. More detailed descriptions are given in the section following the table.

The `paths` command no longer exists. Instead we have `add` and `remove`, so instead of `path add` it's `add root`. Also, one can now add and remove banks using the same structure. `paths show` has now been moved into lists as `list roots`. These changes make roots/banks/instruments more consistent.

Table 3: Yoshimi Text Commands, Part 1

`..`	Step back up one command level.
`/`	Step back up to the top command level.
`add bank [s]`	Add a new bank to the current root.
`add root [s]`	Define a new root path and add it to the list.
`remove bank [s]`	Delete the bank from the current root.
`remove root [s]`	De-list the given root path.
`list banks [n]`	List instruments in bank *n* or the current bank.
`list effects [n]`	List effect types for *[*n] or for 'all'.
`list instruments [n]`	List all instruments in bank *n* or current bank.
`list parts`	List number of parts available, and more.
`list roots`	List all available root paths.
`list setup`	Displays the dynamic system settings.
`list history [s]`	Lists recent files (patchsets, scales, and states).
`list vectors [n]`	Lists the settings for enabled vectors, channel *n*.
`load instrument [s]`	Loads an instrument to the current part from a file.
`load patchset [s]`	Load a complete patch set from the file.
`load vector [s]`	Loads a vector setup from a file.
`save patchset [s]`	Save the patch set to a named file.
`save instrument [s]`	Saves the current part to a named instrument file.
`save setup`	Save the current dynamic system settings.
`save vector [s]`	Saves the current vector setup to a named vector file.
`set activate [n]`	Set part-activate on MIDI program change (0 on, 1 off).
`set alsa audio [s] *`	Sets the name of the audio device to look for.
`set alsa midi [s] *`	Sets the name of the MIDI source to look for.
`set available [n]`	Set the number of available parts (16, 32, 64).
`set bank [n]`	Set current bank to ID *n*.
`set ccbank [n]`	Set MIDI CC for bank changes (non-0 or -32 disables).
`set ccroot [n]`	Set MIDI CC for root path changes (above 119 disables).
`set extend [n]`	Set MIDI CC for extended program change (above 119 disables).
`set insert effects [n]`	Set insertion effects for editing.
`set jack midi [s] *`	Sets the name of the JACK MIDI source for Yoshimi.
`set jack server [s] *`	Sets the name of the JACK server for Yoshimi.
`set part [n1] program [n2]`	Load instrument *n2* into part *n1*.
`set part [n1] channel [n2]`	Set the MIDI channel *n2* for part *n1*.
`set part [n1] destination [n2]`	Set audio destination of part *n1* to main, part, or both.
`set preferred audio [s] *`	Set audio connection type (jack/alsa).
`set preferred midi [s] *`	Set MIDI connection type (jack/alsa).

Table 4: Yoshimi Text Commands, Part 2

`set - preset [n]`	Set effect preset.
`set - preset [n]`	Set numbered effect preset.
`set program [n]`	Set MIDI program change (0 disables, otherwise enables).
`set reports [n]`	Set report destination (gui, stderr, show, and hide).
`set root [n]`	Set current root path to ID *n*.
`set shift [n]`	Set master key shift for notes, semitones.
`set system effects [n]`	Set System Effects for editing.
`set vector [n1] x/y cc [n2]`	Sets and enables vector CC. See section below.
`set vector [n1] x/y features [n2]`	Sets channel *n1* X or Y features to *n2*.
`set vector [n1] x/y program [l/r] [n2]`	Loads program *n2* to ch. *n1* X or Y *left* or *right* part.
`set vector [n1] x/y control [n2] [n3]`	Sets `n3` CC to use for X or Y feature `n2` (2, 4, 8).
`set vector [n] [off]`	Disables vector control for channel *n*.
`set volume [n]`	Set the master volume.
`reset`	Return to the start-up conditions, if 'y' selected.
`stop`	Cease all sound immediately!
`?` or `help`	List all commands.
`exit`	Tidy up and close Yoshimi down, if 'y' selected.

Entries marked with an asterisk need to be saved, and *Yoshimi* restarted, to activatate.

Commands are not case sensitive, and an invalid one will print a reminder. Often one needs only the first letter of a command, as long as it is unambiguous. The examples above show their minimum abbreviations in capitals. However, Yoshimi is quite pedantic, and if you type the command in full it must be exactly correct!

All number ranges start from zero. This is different from the GUI where most (but not all) start from one. So CL part 0 is GUI part 1, but CL bank 0 is GUI bank 0.

More commands will be added, and the organisation of the commands may change slightly.

17.3 Command Descriptions

This section describes the command-line commands in more detail. Obviously, some more needs to be written about some of the commands. Note that all the parameters for the `load` and `save` parameters are strings, and the parameters are compulsory, not optional.

1. ... Step back up one command context level. This command can immediately precede another command, so that the second command takes places at the context above the current context.

2. /. Step back up to the top command context level. This command can immediately precede another command, so that the second command takes places at the top context.

3. add bank [s]. Define a new bank, *s*, where *s* is a bank name, and add it to the current root.

4. add root [s]. Define a new root path, *s*, and add it to the list of root paths.

5. remove bank [s]. Delete the bank named *s*, and all its contents, from the current root path.

6. remove root [s]. De-list the root path named *s*.

7. list banks [n]. List the instruments and IDs in bank *n* or the current bank/root.

8. list effects [n]. List effect types for [n]. If the parameter is the word *all*, then list every effect and all its presets along with the preset number.

9. list history [s]. Displays the recently-used files, including patchsets, scales, and states. If no parameter is given, then this command lists all three files in sequence.

10. list instruments [n]. List all instruments and IDs in bank *n* or the current bank/root.

11. list parts. Lists the number of parts available and parts with instruments currently installed along with any enabled with the default sound. Also shows their audio destination: *M* = main L/R, *P* = part L/R, *B* = both, and - = disabled or unavailable. This way one can tell if an instrument patch is installed even if it is not currently usable. To avoid unnecessary list length, the default "Simple Sound" is not shown unless it is enabled.

12. list roots. Displays all defined root paths.

13. list setup. Displays the current dynamic system settings.

14. list vector [n]. Lists the settings for vector on channel *n*.

15. load instrument [s]. Loads an instrument into the current part from the named file. The file-name parameter *s* is mandatory.

16. load patchset [s]. Load a complete patch set from a named file, *s*.

17. load vector [s]. Loads an vector setup from the named file. The file-name parameter *s* is mandatory.

18. save patchset [s]. Saves the current patchset to the file named *s*.

19. save instrument [s]. Saves the instrument of the current part to the named file. The file-name parameter *s* is mandatory.

20. save setup. Save the current dynamic system settings. These settings get saved to the state file (we think).

21. save vector [s]. Saves the vector setup to the named file. The file-name parameter *s* is mandatory.

22. set activate [n]. Set part-activate on MIDI program change. *n*=0 disables this feature, and 1 or any non-zero value enables this feature. This feature applies to command line program change as well.

23. set alsa audio [s]. Sets the name of the audio hardware device ALSA looks for. Requires a restart of *Yoshimi*.

24. set alsa midi [s]. Sets the name of the MIDI source ALSA looks for. Requires a restart of *Yoshimi*.

25. set available [n]. Set the number of available parts (16, 32, 64). Note that 32 and 64 are supported in the newest versions of *Yoshimi*.

26. set bank [n]. Set current bank to ID *n*.

27. set ccbank [n]. Set the MIDI CC for bank changes (anything other than 0 or 32 disables MIDI CC).

28. set ccroot [n]. Set the MIDI CC for root path changes (values above 119 disable this feature).

29. set extend [n]. Set CC value for extended program change (values above 119 disables this feature).

30. set insert effects [n]. Set insertion effects for editing. What are the possible values of *n*?

31. set jack midi [s]. Sets the name of the JACK MIDI source for Yoshimi. Requires a restart of *Yoshimi*.

32. set jack server [s]. Sets the name of the JACK server Yoshimi tries to connect to. Requires a restart of *Yoshimi*.

33. set part [n1] program [n2]. Load instrument *n2* into part *n1*. Example: `set part 4 program 130`

34. set part [n1] channel [n2]. Set the MIDI channel *n2* for part *n1*. If the channel number is greater than 15, no further MIDI messages will be accepted by that part.

35. set part [n1] destination [n2]. Set the audio destination of part *n1* to main (*1*), part (*2*), both (*3*). Also enables the part if not already enabled.

36. set preferred audio [s]. Set the audio connection type. The parameter should be either "jack" or "alsa".

37. set preferred midi [s]. Set the MIDI connection type. The parameter should be either "jack" or "alsa".

38. set - preset [n]. et effect preset. Set numbered effect preset.

39. set program [n]. Set MIDI program change (0 disables, anything else enables).

40. set reports [n]. Sets the report destination or where messages are displayed, and, to some extent, which messages are displayed. Here are the variations on this command that are supported:

- `set reports gui` or `s r g`. All reports are sent to the GUI console window.
- `set reports stderr` or `s r s`. All reports are sent to stderr.
- `set reports show` or `s r sh`. All messages are displayed.
- `set reports hide` or `s r h`. Non fatal low-level messages are discarded.
- `set reports (any other word or nothing at all)` or `s r (other)`. This sets the default condition of sending reports to the CLI and displaying all of them.

41. set root [n]. Set current root path to ID *n*.

42. set shift [n]. Set the master key shift for following notes in semitones (+- octave, 64 for no shift).

43. set system effects [n]. Set System Effects for editing.

44. set vector [n1] x/y cc [n2]. CC *n2* is used for channel *n1* X or Y axis sweep. For X, this also enables vector control for the channel.

The individual features are now numbered 1-4 and can be `enabled` or `reversed` (any other word disables the feature). "Reversed" means that, instead of the X left rising in value with increasing CC value, it decreases. X right does the opposite of course.

Feature 1 is always fixed as 7 (volume) and is not reversible. Features 2 to 4 can also have the outgoing CC changed to any valid one. The vector is just about the only command-line entry that starts from 1.

The original system where bits were ORred together was done to make NRPN control as efficient as possible. That hasn't changed, but log messages refer to the command-line numbering.

A more detailed discussion of command-line vector control is presented in section 16.3 "Vector / Command Line" on page 186.

45. set vector [n1] x/y features [n2]. Sets channel *n1* X or Y features to *n2*.

46. set vector [n1] x/y program [l/r] [n2]. Loads program *n2* to channel *n1* X or Y *left* or *right* part.

47. set vector [n1] x/y control [n2] [n3]. Sets `n3` CC to use for X or Y feature `n2` (2, 4, 8). *n3* is the CC to be used for feature number *n2* on X vector channel *n1*. The *x* is a sort of hidden parameter as the code uses an offset dependent on whether it is *x* or *y*. Also *n1* can be omitted in which case it will use the last defined channel number. Using alternate words and numbers gives a great deal of flexibility like this.

48. set vector [n] [off]. Disables vector control for channel *n*.

49. set volume [n]. Set the master volume.

50. reset. Return to the start-up conditions, if 'y' selected.

51. stop. Cease all sound immediately!

52. ? or help. List commands for current mode. All of the minimum command-line abbrevations are capitalised in the help listing.

53. exit. Tidy up and close Yoshimi down.

18 LV2 Plug-in Support

Yoshimi now runs as an LV2 plugin.

Supported features:

1. Sample-accurate midi timing.
2. State save/restore support via LV2_State_Interface.
3. Working UI support via LV2_External_UI_Widget.
4. Programs interface support via LV2_Programs_Interface.
5. Multi channel audio output. 'outl' and 'outr' have LV2 index 2 and 3. All individual ports numbers start at 4.

Planned feature: Controls automation support. This will be a part of a common controls interface.

Download and build the source code found at the *Yoshimi* site [17], and one will find a file named `LV2_Plugin/yoshimi_lv2.so`

The LV2 *Yoshimi* interface can be run in hosts such as *Ardour 3*, *Carla*, and *QTractor*. We have tested LV2 using the latest versions of *Ardour*, *Muse* and *Qtractor* as LV2 hosts. We also tried *Carla* but couldn't get anywhere with it at all... if someone is familiar with it we'd be interested in what results they get. Slightly to our surprise the one that performed the best in every respect was *Qtractor*. *Muse* got everything correct, but was prone to Xruns on program changes. *Ardour* was very problematical. There were no Xruns but it seemed to have odd timing issues. Also, on two occasions it managed to shorten the decay times of two of the instruments. We don't understand how it managed that. Our reference was the original MIDI file played into a stand-alone *Yoshimi* via `aplaymidi`. This also behaves identically to the file being sequenced by *Rosegarden*.

At some point we hope to document the process of setting up and using the *Yoshimi* LV2 plugin. In the meantime, we include some notes.

If *Yoshimi*'s internal buffer is *smaller* than the JACK buffer, it sounds quite horrible. If it's the same or greater there's no problem. The reason we didn't find this before is that it only affects *Yoshimi* LV2. Also it doesn't apply to any of the released versions. The cause of the problem is that the LV2 code

doesn't have the same looping structure that was added to the standalone routine to deal with exactly this situation (re-entering the audio 'construction' function until the JACK buffer is filled). We hope Andrew can deal with this fairly soon as we don't understand the LV2 code very well. There are valid reasons for wanting different sizes for these buffers. The internal buffer size as well as affecting latency and CPU load also alters the sound in quite subtle ways. It particularly affects the behaviour of filters. One day we may be able to stop this happening, but in the mean time we have to live with it. For our purposes, we find a buffer size of 128 or 256 is best.

19 Yoshimi Man Page

The *Yoshimi* man page is actually the output of the `yoshimi --help` command, which prints out the command-line that are discussed in this section.

Yoshimi 1.3.6, a derivative of ZynAddSubFX - Copyright 2002-2009 Nasca Octavian Paul and others, Copyright 2009-2011 Alan Calvert, Copyright 20012-2013 Jeremy Jongepier and others, Copyright 20014-2015 Will Godfrey and others.

`-a --alsa-midi[="device"]`
Use ALSA MIDI input. From the command line, as well as autoconnecting the main L & R outputs to JACK, with ALSA MIDI one can now auto-connect to a known source.

```
./yoshimi -K --alsa-midi="Virtual Keyboard"
```

ALSA can often manage with just the client name. This command is case sensitive, and quite fussy about spaces, etc., so it's wise to use quotes for the source name, even if they don't seem to be needed.

`-A --alsa-audio[=device]`
Use ALSA audio output.

`-b --buffersize=size`
Set ALSA internal audio buffer size.

`-c --show-console`
Show the console on startup.

`-D --define-root`
Define the path to a new bank root. *Yoshimi* will then immediately scan this path for new banks, but won't make the root (or any of its banks) current. The final directory doesn't in fact have to be 'banks' but traditionally *Yoshimi* has always done this. Also, when running from the command line there is now access to many of the system, root, bank, etc. settings. See section 17 "The Yoshimi Command Line Interface" on page 188.

`-i --no-gui`
Do not show the GUI. See section 17 "The Yoshimi Command Line Interface" on page 188for more information about this mode of operation. Note that the command-line and the GUI can be available simultaneously. Also note that this switch allows *Yoshimi* to run on a dumb terminal or virtual console.

`-j --jack-midi[=device]`
Use JACK MIDI input. From the command line, as well as autoconnecting the main L & R outputs to JACK, with JACK MIDI one can now auto-connect to a known source.

```
./yoshimi -K --jack-midi="jack-keyboard:midi_out"
```

JACK needs the port as well as the name. This command is case sensitive, and quite fussy about spaces, etc., so it's wise to use quotes for the source name, even if they don't seem to be needed.

`-J --jack-audio[=server]`
Use JACK audio output. Connect to the given JACK server if given.

`-k --autostart-jack`
Auto-start the JACK server. Note that this can cause some odd behavior on some systems, so be aware of that possibility.

`-K --auto-connect`
Auto-connect JACK audio. Note that, if the auto-connect feature has been specified in the user-interface, and saved to the *Yoshimi* configuration file, there is then no way to disable this feature from the command-line, at this time.

`-l --load=file`
Load a `.xmz` file.

`-L --load-instrument=file`
Load an `.xiz` file

`-N --name-tag=tag`
Add tag to client-name.

`-o --oscilsize=size`
Set ADDSynth oscillator size (OscilSize).

`-R --samplerate=rate`
Set ALSA audio sample rate.

`-S --state[=file]`
Load saved state from file, where the file defaults to `$HOME/.config/yoshimi/yoshimi.state`

`-u --jack-session-file[=file]`
Load the named JACK session file.

`-U --jack-session-uuid[=uuid]`
Load the named JACK session by UUID.

`-? --help`
Give this help list.

`--usage P`
rovide a short usage message.

`-V --version`
Print program version.

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

From the command line, as well as autoconnecting the main L & R outputs to JACK, with either JACK or ALSA MIDI one can now auto-connect to a known source.

ALSA can often manage with just the client name, but JACK needs the port as well. These commands are case sensitive, and quite fussy about spaces etc. so it's wise to use quotes for the source name, even if they don't seem to be needed.

20 Building Yoshimi

This section describes building and debugging *Yoshimi*. Building *Yoshimi* requires getting the source code, making sure all of the necessary dependencies are installed, and using CMake to set up the build.

The source-code is located at its main location ([17]) or its alternate location ([18]).

Like *ZynAddSubFX*, *Yoshimi* uses CMake as its build system [23]. CMake is a preprocessor that can generate project build setups for Visual Studio, UNIX make, and Xcode.

20.1 Yoshimi Source Code

Get the source code version you want from SourceForge (`http://sourceforge.net/projects/yoshimi/files/1.3/`). Download the desired tar-ball and unpack it in your work area.

Since SourceForge has had some issues, the *Yoshimi* team has wisely hosted the source code at another site as well, `https://github.com/abrolag/yoshimi`. One can grab the whole git repository there using the following command in your work area:

```
$ git clone https://github.com/abrolag/yoshimi.git
```

20.2 Yoshimi Dependencies

As of recent versions of *Yoshimi* 1.3.9, building *Yoshimi* requires C++11. For GNU builds, this requires *gcc 4.7* and above.

To save some wasted time, make sure the *development versions* of the following packages have been installed using your Linux distribution's package manager:

- `pkg-config`
- `libz`
- `fftw3f`
- `mxml`
- `ALSA` (libasound)
- `JACK`
- `Boost`
- `fontconfig`
- `libcairo`
- `FLTK`
- `lv2`

These package names are from *Debian Jessie*:

- `automake`
- `build-essential`
- `cmake-curses-gui`
- `dssi-dev`
- `fluid`

- `libboost-dev`
- `libcairo2-dev`
- `libfftw3-dev`
- `libfltk1.3-dev`
- `libglu1-mesa-dev`
- `libjack-jackd2-dev`
- `libjpeg-dev`
- `libmxml-dev`
- `libncurses5-dev` (new dependency)
- `libreadline-dev` (new dependency)
- `libxft-dev`
- `libxinerama-dev`
- `libxml2-dev`
- `xutils-dev`
- `zlib1g.dev`

LV2 plugin adds one more dependency:

`lv2-dev` with minimum version ¿= 1.0.0.

Other distros may have slightly different names or version numbers, and may even have these installed by default. If in doubt, try looking for just the main part of the name, but with the `-dev` extension where appropriate.

20.3 Build It

The following instructions are for an in-source build. An in-source build is simpler if you just want to build and install *Yoshimi*.

We will also show how to set up for an out-source-build, which keeps the build products out of the way.

The location of `CMakeList.txt` does not appear to be standard, so these instructions differ in details from the build instructions of *ZynAddSubFX*. Basically, the build is based in the project's `src` directory, instead of its root directory.

1. Enter the source directory where the code was unpacked.
2. Generate the project build-files:

```
$ cd src
$ cmake ..
$ cmake .
```

3. Build the code and install it (as root):

```
$ make
# make install
```

Here is how to make an out-of-source debug. Despite what cmake documentation (and Googling) says, using a command like the following *does not work* unless you have `ccmake` installed.

```
$ cmake -DCMAKE_BUILD_TYPE=Debug ..
$ ccmake
```

In Debian Linux, install the `cmake-curses-gui` package to get access to `ccmake`. Or use the shorter `cmake -DBuildForDebug=on ..` command below.

1. Enter the source directory where the code was unpacked.
2. Create a "Debug" or "Release" directory for an out-of-source build:

```
$ cd src
$ mkdir Debug
```

3. Generate the project build-files in the `Debug` directory.

```
$ cd Debug
$ cmake -DBuildForDebug=on ..
$ make
```

The output file, and executable name `yoshimi` is now ready to run (and be debugged).

Here is a debugging use case we used in *Yoshimi 1.3.5.1* and slightly earlier versions. Here is how to verify the bug:

1. Run the following command:

```
$ yoshimi -a -A
```

2. Navigate the following command path: Menu / Instrument / Show Banks
3. Select the **RENAME** button.
4. Select the bank (e.g. Arpeggios).
5. In the file prompt that comes up, click **Cancel**.
6. Observe a "Segmentation fault".

To avoid a lot of debugging, let *valgrind* find the bug for you. Install *valgrind*. Then, in the `src/Debug` directory, run:

```
$ valgrind --log-file=yoshvalgrind.log ./yoshimi -a -A
```

In the log file, one sees that the last good call was in the `Bank :: readOnlyBank()` function. That would be a good place to put a breakpoint.

However, even without *valgrind*, this particular bug is easy to find under the *debugger*. The steps are simple:

```
$ cd src/Debug
$ gdb ./yoshimi
(gdb) r -a -A
```

Then repeat the steps above that trigger the bug. One sees

```
Program received signal SIGSEGV, Segmentation fault.
```

Issue the command "backtrace" at the `(gdb)` prompt. There will be a list of stack frames starting at 0. Frame 1 is in *Yoshimi*, so issue the command "frame 1", a see

```
if (strlen(tmp) > 2) ...
```

`tmp` is a null pointer here; we need to add an initial check for the null pointer there to avoid triggering the crash.

Not worry now, though, the bug has been officially fixed.

20.4 Yoshimi Code Policies

Yes, we actually have *Yoshimi* code policies. Look how many there are! :)

If the version string contains a 4th number this will always be just a bugfix, and will never have features added or changed from the main version. For example:

- yoshimi-1.3.5 Main version.
- yoshimi-1.3.5.1 First bugfix.
- yoshimi-1.3.5.2 Second bugfix. (Surely not!)

We won't accept fixes for spelling errors in the *code*. For a start, from bitter experience it is fatally easy to change two variables to the same name! Also, there's no point, after all they are only a mnemonic for memory addresses etc. 'volume' and 'LFO' could just as well be 'turnyfing' and 'derfingwotwiggles'.

To avoid possible confusion, from now on 'master' will display the last released version number (without bugfix digits) with an 'M' suffix - unless it is a release candidate in which case the suffix will be rc[n]. For example:

- Last release was yoshimi-1.3.5.2
- Master is shown as yoshimi-1.3.5 M

XML files created with this release will have: major version 3 and minor version 5.

If using Fluid to edit GUI files, please close all windows and collapse all menus *before* the last save. I know it's tedious, but it avoids storms of spurious 'changes' that make genuine ones harder to see.

21 Summary

In summary, we can say that you will absolutely love *Yoshimi*. There are some topics that this document does not yet treat:

- Changing the colors and style of the GUI, as seen in some versions of *ZynAddSubFX*. This does not yet seem possible, because the synth part is not truly separate from the GUI, and the *Yoshimi* developers have not found it easy to recode the FLTK GUI.

  A car analogy.
  A sample player is a drive along a straight, wide, almost new highway with only 2 other cars in sight, on a lightly overcast summer's day in a Ford Fiesta at around 40 MPH.
  Yoshimi is a white-knuckle trip over a Swiss mountain pass in a blizzard, at night, facing donkeys, trucks and bandits, while driving an open-frame kit car doing 90 +
  In recent times we've been able to dispose of the donkeys, and the bandits are on the run :)

Although we must admit the new Fiesta are actually pretty slick and fast!

22 References

This section provides references for this *Yoshimi* user's manual, as well as some other information.

Although the *Yoshimi* project is based on SourceForge, it also has a mirror on GitHub, and a mailing list on FreeLists.Org. One can subscribe with an e-mail to: yoshimi-request@freelists.org or by visiting: http://www.freelists.org/list/yoshimi. To post to the list, email to: yoshimi@freelists.org The archive of the old SourceForge mailing list is found at: https://www.freelists.org/archive/yoshimi.

References

[1] Will J. Godfrey *A discussion of making Bank/Root specifications more regular.* http://sourceforge.net/p/yoshimi/mailman/message/33200765/ 2014.

[2] Chris Ahlstrom *A Yoshimi User's Manual* https://github.com/ahlstromcj/yoshimi-doc/ 2015.

[3] Chris Ahlstrom *A Yoshimi Cookbook* https://github.com/ahlstromcj/yoshimi-cookbook/ 2015.

[4] Cormi *Cormi Collection* https://www.freesound.org/people/cormi/ His cormi57.wordpress.com site has a lot of other nice sound material, as well. 2015.

[5] Straulino *A collection of instruments* http://www.straulino.ch/zynaddsubfx/ http://www.straulino.ch/zynaddsubfx/Drums_DS_v2012-12-08.zip http://www.straulino.ch/zynaddsubfx/ZynAddSubFX_Natural_Drum_Kit_Demo_v2012-06-22.ogg

[6] "folderol" *A collection of instruments* http://www.kara-moon.com/forum/index.php?topic=762.0

[7] Will Godfrey *Will Godfrey's Music* http://www.musically.me.uk See the "Bits 'n Stuff" link especially.

[8] Will J. Godfrey *A discussion of licensing issues with Youshimi and ZynAddSubFX components, and FLTK library versions.* http://sourceforge.net/p/yoshimi/mailman/yoshimi-devel/ 2015.

[9] caonoize *ZynAddSubFX by paulnasca: Downloads, Banks, Patches, etc.* http://www.kvraudio.com/product/zynaddsubfx_by_paulnasca/downloads 2015.

[10] mmxgn *Instruments made with zynaddsubfx/yoshimi* https://github.com/mmxgn/instruments-zyn 2011.

[11] Rakarrack team *The download site for the Rakarrack software effects engine.* http://rakarrack.sourceforge.net/ 2015.

[12] LinuxMusicians newsgroup *Ring Modulation in ZynAddSubFX* http://linuxmusicians.com/viewtopic.php?f=1&t=8178 2012.

[13] Manuel Op de Coul ¡coul@huygens-fokker.org¿, *The Scala Musical Tuning Application.* http://www.huygens-fokker.org/scala/ Scala is a powerful software tool for experimentation with musical tunings, such as just intonation scales, equal and historical temperaments, microtonal and macrotonal scales, and non-Western scales. 2014.

[14] Sharphall *How to create drum sounds in ZynAddSubFX or Yoshimi, Part 1* http://sharphall.org/docs/zynaddsubfx_yoshimi_drum_tutorial.php Never got continued, unfortunately.

[15] Gordon Reid *Synth Secrets: Creative Synthesis* http://www.soundonsound.com/sos/allsynthsecrets.htm 1999-2004.

[16] Graham *A small musical website* http://x31eq.com and specifically http://x31eq.com/zasf

[17] Yoshimi team abrolag@users.sourceforge.net *The download site for the Yoshimi software synthesizer.* http://yoshimi.sourceforge.net/ 2015.

[18] Yoshimi team *The alternate location for the Yoshimi source-code.* https://github.com/abrolag/yoshiminow. 2015.

[19] Yoshimi team. *Yoshimi user/developer mailing list* http://www.freelists.org/list/yoshimi 2015.

[20] Yoshimi team. *Yoshimi user/developer mailing list archive* http://www.freelists.org/archive/yoshimi 2015.

[21] Mark McCurry, Paul Nasca (ZynAddSubFX team) *The download site for the ZynAddSubFX software synthesizer.* http://zynaddsubfx.sourceforge.net/ 2015.

[22] Paul Nasca? *The download site for the ZynAddSubFX banks, instruments, parameters, and demos* http://zynaddsubfx.sourceforge.net/doc/instruments/ 2009.

[23] ZynAddSubFX team. *Building ZynAddSubFX* http://zynaddsubfx.sourceforge.net/Doc/#_appendix_b_building_zynaddsubfx 2009.

[24] AMSynth team. *Provides a number of OGG demos of ZynAddSubFX sounds. It also includes the author's own "Vanilla" bank, and links to additional patch collections and demonstration videos.* http://www.amsynth.com/zynaddsubfx.html

[25] Mark McCurry, Paul Nasca *ZynAddSubFX online manual.* http://zynaddsubfx.sourceforge.net/Doc 2015.

[26] Paul Nasca *Original ZynAddSubFX manual, ODT format.* http://linux.autostatic.com/docs/zynaddsubfx_manual-v0.1.odt 2011.

[27] Paul Nasca *Original ZynAddSubFX manual, PDF format.* http://linux.autostatic.com/docs/zynaddsubfx_manual-v0.1.pdf 2011.

[28] Paul Nasca, Mark Murray *The download package for the ZynAddSubFX source-code* http://downloads.sourceforge.net/project/zynaddsubfx/zynaddsubfx/2.5.0/zynaddsubfx-2.5.0.tar.gz

[29] Jeremy ("autostatic") *ZynAddSubFX Manual, good overall of most Zyn settings and knobs* http://wiki.linuxaudio.org/wiki/zynaddsubfx_manual 2011.

Index