

Reaper Plugins for the  
LilypondToBandVideoConverter  
(Version 1.2)

Dr. Thomas Tensi

2024-08-12

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Overview . . . . .	3
1.2	Available Scripts . . . . .	3
<b>2</b>	<b>Installation of the Plugins</b>	<b>5</b>
<b>3</b>	<b>Description of the Plugins</b>	<b>6</b>
3.1	Motivation for the Plugins . . . . .	6
3.2	AdaptTracksToLTBVConventions . . . . .	7
3.3	AdjustSoXEffectNames . . . . .	11
3.4	ExportLilypond . . . . .	12
3.5	ImportMidi . . . . .	14
3.6	MakeRegionsFromRegionStructureTrack . . . . .	16
3.7	NormalizeStructuredVoiceTracks . . . . .	17
3.8	SelectLTBVCAudioSources . . . . .	18
<b>4</b>	<b>Demo Song</b>	<b>22</b>
<b>5</b>	<b>Coping with Errors</b>	<b>23</b>
	<b>References</b>	<b>23</b>

# 1. Introduction

## 1.1 Overview

The “Reaper Plugins for the LilypondToBandVideoConverter” package provides Lua plugins for being used in the Reaper DAW [Reaper]; they assist in using the LilypondToBandVideoConverter tool chain for generating notation videos from arrangement text files.

LilypondToBandVideoConverter (or short `ltbvc`, [LTBVC]) is a command line audio processing tool written in Python for Unix, Windows and Mac OS that uses standard command-line tools to convert a music piece (a song) written in the lilypond notation to

- a PDF score of the whole song,
- several PDF voice extracts,
- a MIDI file with all voices (with additional preprocessing applied to achieve some humanization),
- audio mix files with several subsets of voices (specified by configuration), and
- video files for several output targets visualizing the score notation pages and having the mixes as mutually selectable audio tracks as backing tracks.

Because that command-line oriented approach is a bit tedious, the current package provides several scripts for the Reaper DAW to make this process easier.

## 1.2 Available Scripts

The following scripts are provided:

**AdaptTracksToLTBVConventions.lua:**

applies specific theme and layouts to the tracks and routes audio according to LTBVC conventions,

**AdjustSoXEffectNames\*.lua:**

sets the effect names of the SoX effects in either all or the selected tracks to the command line strings required for processing outside of the Reaper DAW,

**ExportLilypond.lua:**

transforms the notes of the selected MIDI item into a textual lilypond note/chord sequence and returns them in a text box,

**ImportMidi.lua:**

scans the current project for tracks with single MIDI items with names conforming to some pattern and replaces those by the corresponding track in an associated MIDI file filtering out unwanted MIDI items before import,

**MakeRegionsFromRegionStructureTrack.lua:**

either makes regions based on a track with structural MIDI items or generates that region structure track with MIDI items from the current regions,

**NormalizeStructuredVoiceTracks.lua:**

scans all tracks with some specific prefix and normalizes their enclosed MIDI items by removing reverb, chorus and delay control codes, setting note velocities to some default and quantizing the note start and end positions, and

**SelectLTBVCAudioSources.lua:**

allows the simple selection of the audio source for each voice by a selection dialog in a Reaper project conforming to LTBVC conventions (for example, to compare the rendering of the MIDI or raw audio file with the refined audio file in the final mix).

## 2. Installation of the Plugins

The installation is as follows:

1. Copy the Reaper Plugins for the LilypondToBandVideoConverter archive from the repository in [LTBVCPlugins] and unpack it to some temporary directory.
2. Close the Reaper application (if open).
3. Copy the lua-files from the archive subdirectory `src` into the `Lua` subdirectory of the Reaper installation (typically in `\Program Files\Reaper\Lua` or `/Applications/Reaper.app/Lua` in MacOS).
4. If helpful, also copy the documentation file from the archive subdirectory `doc` to the `Lua` sub-directory.
5. Restart Reaper. You should now be able to access the scripts as actions in the Actions menu of Reaper. It is helpful to define some keyboard shortcuts for those actions for a quicker access.

Alternatively — and a little bit easier — you can use the ReaPack plugin [ReaPack] and do an automatic install via the `index.xml` file in the repository [LTBVCPlugins]. After the installation via ReaPack all the scripts can be found in the action list of the Reaper installation via the prefix `LTBVC-Plugins_`; so, for example, the lilypond export script has the action name `LTBVCPlugins_ExportLilypond.lua`.

---

## 3. Description of the Plugins

### 3.1 Motivation for the Plugins

Motivation for all the scripts presented here is to allow input and adaptation of MIDI notes of some arrangement in the Reaper DAW, but then easily integrate that into a `ltbvc` pipeline. Part of this approach is also that one can quickly import the tracks of the generated MIDI file from that external pipeline for checking whether the `ltbvc` and the DAW project are in sync.

To be able to do this, one should first organize MIDI items in the DAW corresponding to the voices and the musical structure of the piece into *structured voice tracks*. It is also helpful when the names of those tracks adhere to some naming pattern (because one of the tools relies on that).

When you have your project organized in such a way is then possible to

- reflect the musical structure as regions based on the items in some track (see section 3.6),
- sets the tracks' layouts in the mixer according to LTBVC conventions and routes the audio accordingly (see section 3.2),
- normalize the structured voice tracks by removing unwanted control codes and quantizing the note positions to a raster compatible with the later lilypond export (see section 3.7),
- export MIDI items in the structured voice tracks as lilypond fragments (see section 3.4),
- repeatedly import the MIDI file generated by `ltbvc` into special project tracks e.g. for a detailed comparison with the structured voice tracks (see section 3.5), and
- adjust the effect names of the SoX effects in the DAW to the command line fragments needed when doing an external rendering (see section 3.3) and also make all of them available in a dialog windows.

Figure 1 shows the structuring of the demo song from the `ltbvc`. Note that this Reaper DAW file is also included in the current distribution.

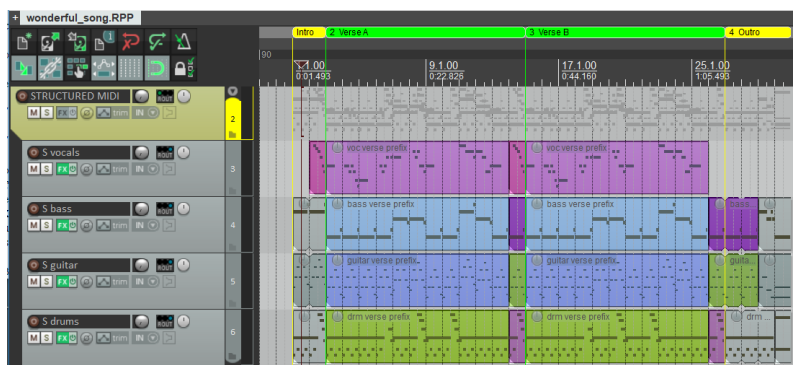


Figure 1: Structured MIDI Representation of Demo Song from ltbvc

## 3.2 AdaptTracksToLTBVCConventions

When a project in a DAW is aimed at emulating the ltbvc tool chain, it is helpful to organize the tracks in a certain manner.

There are seven groups of tracks:

- (optional) tracks with original audio as reference material,
- structured MIDI voice tracks that contain MIDI data for the voices and have loops or aliases indicating the musical structure of voice/song (with MIDI converted to audio by some VST instrument),
- MIDI voice tracks generated by the “midi” phase of the external tool chain (where all structure is expanded, but the MIDI data should be almost identical to the structured MIDI tracks see 3.5), which also have their MIDI converted to audio by some VST instrument,
- raw audio voice tracks generated by the “rawaudio” phase of the external tool chain (where MIDI is rendered as audio but no SoX effects have been applied),
- plain effect voice tracks containing all the SoX effects applied to each voice similar to the “refinedaudio” phase of the external tool chain,
- refined audio voice tracks generated by the “refinedaudio” phase of the external tool chain (as reference tracks), and
- final audio voice tracks where refined audio is panned and has its volume adjusted for the final mix

Each group has its own “folder track” such that they can be individually muted.

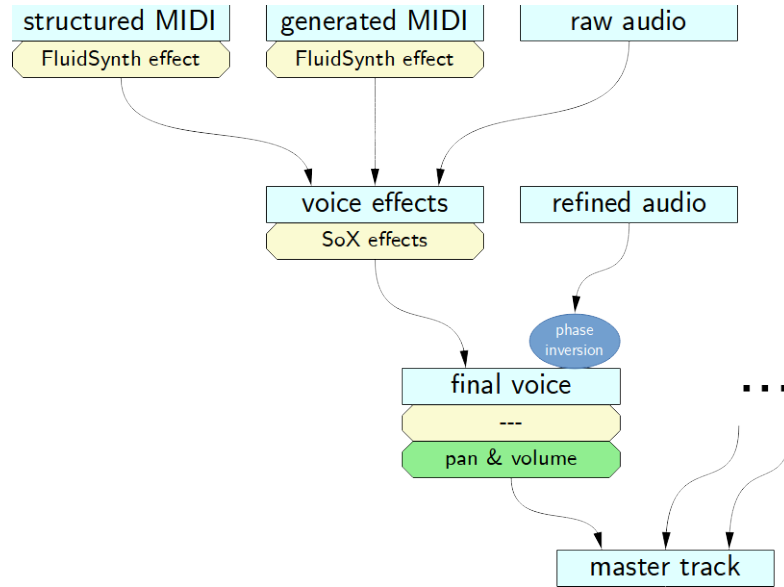


Figure 2: Audio Routing Along Voice Tracks

This project structure seems to be overly complex, but the idea behind that is to factor out commonalities between the tracks.

For each voice the following is done (as shown in figure 2):

- The structured MIDI track, the generated MIDI track and the raw audio track are routed into the plain SoX effect track for that voice. Routing is pre-FX for the raw audio track and post-FX for both the MIDI tracks.

This means that the identical SoX effect chain is applied to all those audio source. Because the MIDI tracks have a FluidSynthPlugin as their MIDI-to-audio converter, in principle all those tracks should provide the same audio signal (apart from the missing humanization in the structured track).

- The output of the original audio tracks, the voice SoX effect track and the refined audio track are both routed into the voice mix track to have pan and volume applied for the final mix. Routing is pre-FX for the refined audio track and post-FX for the effect and original track.

As an additional trick the refined audio track is phase inverted. This means that when both effect track and refined audio track deliver some audio signal, **those should completely cancel out**. This helps when checking whether the internal and external effects chain is identical.

- Finally all voice mix tracks are mixed into the (stereo) master track post-fader.



It is also clear, that not all tracks need all kinds of interaction, for example in the mixer panel. An effect track or a raw audio track does not have to have a volume fader or a pan control because those are handled in the voice mix track. A raw or refined audio track does not need any effects: its media is just routed into another track following.

Hence it is helpful to change the appearance of the tracks accordingly. Reaper allows to customize the track layout as required, so there is a layout style to be preferably applied to an ltbvc project.

The whole procedure looks tedious: routing tracks and setting their layout to reflect their function manually.

Fortunately the script `AdaptTracksToLTBVConventions` does all that. It connects tracks as given in figure 2 via some naming conventions and also applies track layouts to them.

Both naming convention and layout assignment is handled by the script based on a so-called “configuration file”. This is a text file named “LT-BVC\_trackConventions.cfg” that defines several variables and is either located in the script directory or in the directory `.luasettings` in the user’s home directory. An example is provided in the `misc` subdirectory of the distribution.

(For a detailed description of configuration files see the ltbvc documentation [LTBVC].)

The variables relevant for this script are `trackKindToNamePatternMap`, `trackNamePatternWithParentDisabledList`, and `trackNamePatternToConnectionDataMap`.

`trackNamePatternToConnectionDataMap` tells the routing partners for specific name patterns signifying track categories (e.g. effects or raw audio tracks). The variable maps each of those category track name pattern (in convention of the programming language Lua) as the connection source onto a capturing pattern giving the destination track name together with the position of the send and the information whether some phase inversion is done.

For example, the map entry

```
"'^E%s+(%S.*)" : [ '^F%s+%1', 'postFX', true ],"
```

tells that an effect track (starting with an “E” character and some white space followed by the voice name) is connected to a corresponding track starting with an “F” letter and some white space followed by the same track name. Routing is post-FX and a phase inversion is also applied.

Hence the default variable definition matching figure might look like

TRACK KIND	TRACK NAME PREFIX
original track	O_
structured MIDI track	S_
generated MIDI track	M_
raw audio track	RA_
effect track	E_
refined audio track	RF_
mix track	M_

Figure 3: Default Naming Conventions for Tracks

```
trackNamePatternToConnectionDataMap = "{
  '^E%s+(%S.*)' : [ '^F%s+1', 'postFX', false ],
  '^M%s+(%S.*)' : [ '^E%s+1', 'postFX', false ],
  '^RA%s+(%S.*)' : [ '^E%s+1', 'preFX', false ],
  '^RF%s+(%S.*)' : [ '^F%s+1', 'preFX', true ],
  '^S%s+(%S.*)' : [ '^E%s+1', 'postFX', false ]
}"
```

That default naming convention for the track routing assumes track names as given by the table in figure 3.

The variable `trackNamePatternWithParentDisabledList` gives a list of patterns for tracks names that are not directly connected to the master track.

So for example the setting

```
trackNamePatternWithParentDisabledList = "[
  '^E%s+%S.*$', '^RA%s+%S.*$'," ...
]"
```

makes sure that effect tracks as well as raw audio track (with the convention from above!) are not connected to the master track.

`trackNamePatternToColorAndLayoutMap` is a map with each entry being a regular expression string mapped onto a pair of colour and name of some layout in Reaper.

For example, the entry

```
"'^E%s+': [ 0xFF8080, 'gf - Effect Bus' ]"
```

tells that a track with name starting with “E” followed by at least one blank shall be set to colour light blue (=0xFF8080 in BGR-notation) and to a layout named “gf - Effect Bus”.

So the variable could be set to

```
trackNamePatternToColorAndLayoutMap = "{
  '^EFFECTS': [ 0xFF0000, 'gc - Grouping without FX'],
  '^E%s+': [ 0xFF8080, 'gf - Effect Bus' ]
}"
```

When the script is run on the demo project, the routing is adapted (as shown

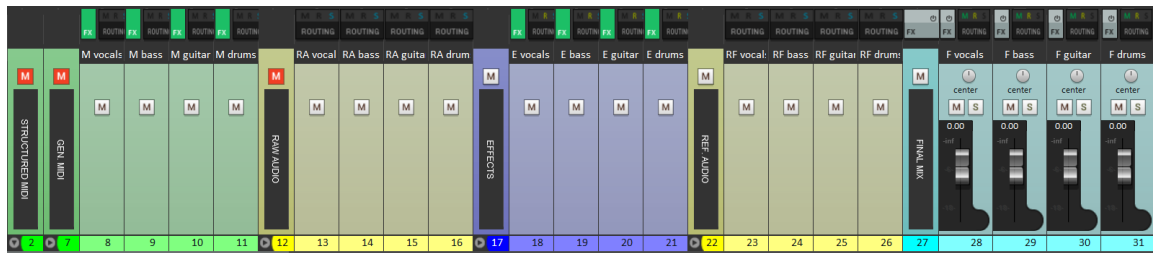


Figure 4: Standardized ltbvc Track Layout in Mixer

in figure 2) and the mixer will look like shown in figure 4.

### 3.3 AdjustSoXEffectNames

In the “refinedaudio” phase of the external tool chain SoX is used for applying audio effects to the raw audio. In the DAW the SoX plugins from [SoXPlugins] emulate the external SoX program bit-identically.

When tweaking the SoX plugins, the command lines for the SoX program can be easily extracted from the DAW. This is done by the scripts `AdjustSoXEffectNamesInAllTracks` and `AdjustSoXEffectNamesInSelectedTracks`, where the former one processes all tracks while the latter processes only the selected tracks.

The scripts do two things:

- The correct command line text for the corresponding SoX effect is used as the name of the SoX plugin in the DAW.
- For those tracks containing SoX plugins, track name and the command line for the effects are written into Reaper’s message window.

Figure 5 shows an example of the command line generation for a SoX phaser effect.

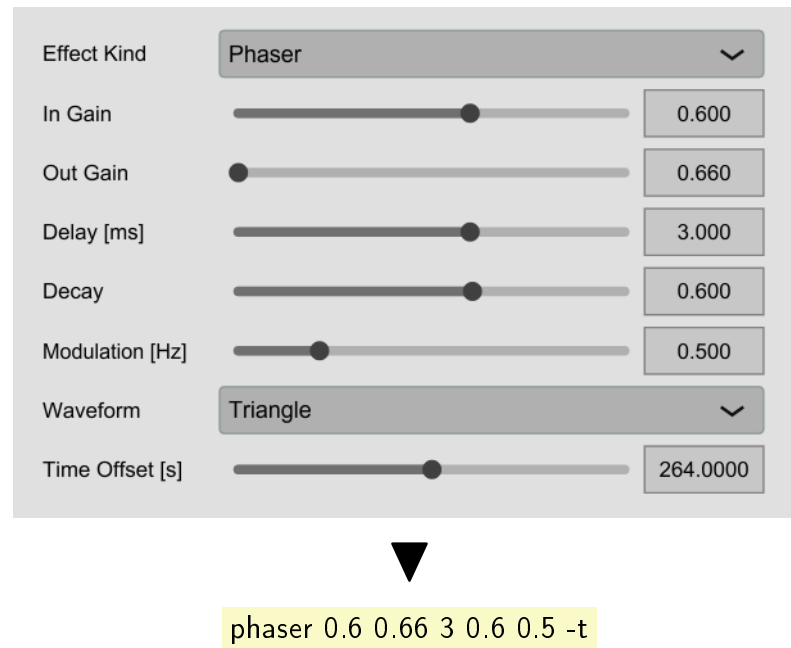


Figure 5: Extraction of Command Line Fragment for SoX Effect Plugin

## 3.4 ExportLilypond

The command here is applied to some selected MIDI item and transforms its notes into a textual lilypond note/chord sequence and finally returns it in the Reaper message box. One can then copy the text into the clipboard and insert it into a lilypond file for the song and later processing by the `ltbvc`.

The notes produced by the script are in English notation. That means for example, an  $f\sharp$  (**f** sharp) note is “fs”, an  $eb$  (**e** flat) note is “ef”. The algorithm analyzes the underlying MIDI notes along the measures and groups them into the least possible number of notes still conforming to score guidelines. Chords are automatically detected.

This generation of notes is dependent on a line in the project settings defining the key. E.g. the line `key=f` in figure 6 defines the key of some song to be “f” (major). Only major keys can be defined, but, however, this only affects whether accidentals used shall be sharps or flats.

All instruments have a default octave defined by their names, where the note sequences start as follows:

- bass, keyboardBottom  $\rightarrow C_1$ ,
- keyboard  $\rightarrow C_2$ ,
- guitar, keyboardTop, strings  $\rightarrow C_3$ ,

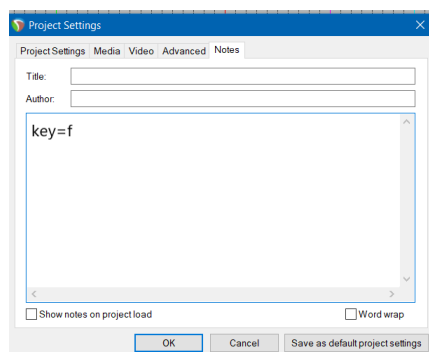


Figure 6: Key Definition for ExportLilypond

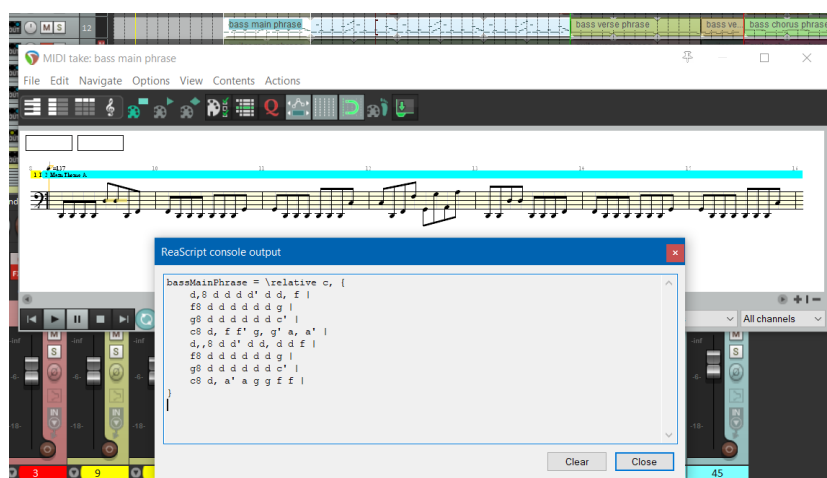


Figure 7: Exporting a Bass MIDI Item

- vocals  $\rightarrow C_4$ , and
- drums, percussion  $\rightarrow$  drum clef

So, for example, a guitar fragment will start with the lilypond text `\relative c'` ( $= C_3$ ), a bass fragment with `\relative c`, ( $= C_1$ ).

Activating the action on some MIDI item puts the resulting MIDI fragment into the message box as shown in figure 7.

Doing the same for some drum MIDI item puts the resulting MIDI fragment also into the message box as shown in figure 8, but uses drum notation instead. This is triggered by the item name starting with either “drums ” or “percussion ”.

The durations in the output are optimized for being conformant to standard notation practice and also switch from and to triplets when appropriate.

A note will be split into parts tied together when its duration is not allowed

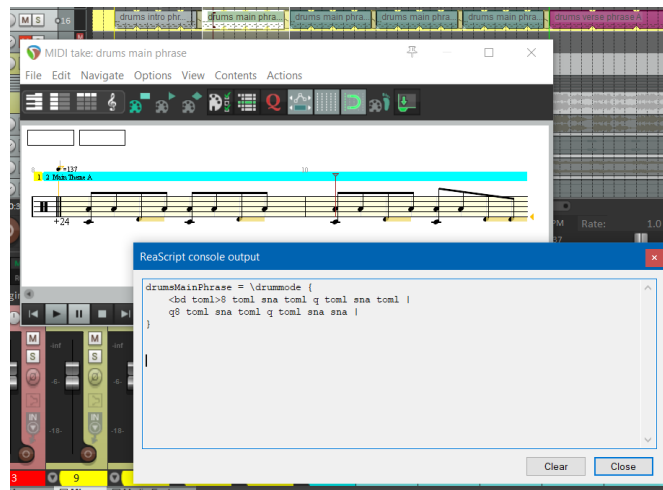


Figure 8: Exporting a Drums MIDI Item

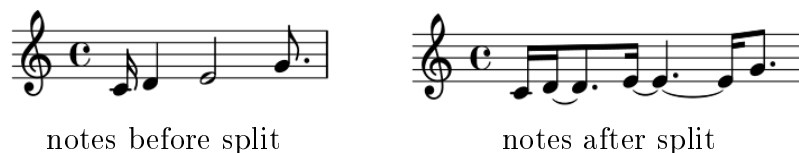


Figure 9: Splitting Notes on Musical Raster Positions

its start position due to notation standards. For example, when in a measure a quarter note follows a sixteenth note, it will be split into a sixteenth note and a dotted eighth note to conform to notation guidelines. Figure 9 shows how a simple note sequence is transformed by the algorithm.

Note that the minimum note duration allowed is a  $32^{nd}$ -note or a  $32^{nd}$ -triplet. If the item converted is not quantized accordingly, typically some strange note durations like “e?77?” will occur in the result where this signifies a note with a duration of 77 MIDI ticks (which is  $77/240$  of a quarter note) that cannot be split into meaningful durations.

## 3.5 ImportMidi

The `ltbvc` pipeline produces a temporary MIDI file with some humanization applied. It is helpful to import that file into tracks repeatedly to see the consequences of changes in the `lilypond` file.

Doing this manually is tedious, especially because MIDI tracks in the project may be at arbitrary positions.

Fortunately the tracks generated by the `ltbvc` have a predictable structure

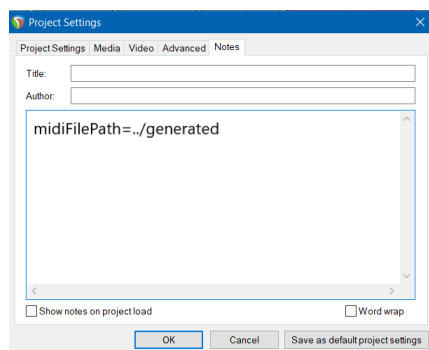


Figure 10: Relative Path Definition for MIDI Import

and naming of tracks. Hence this script scans the DAW project for tracks conforming to that convention: those are tracks with a single MIDI item, where its item name ends with “.mid” and it gives the voice name followed by the name of the MIDI file.

For example, a MIDI item “bass - wonderful\_song-std.mid” is the bass voice in an imported MIDI file “wonderful\_song-std.mid” from the ltbvc.

For the import the location of the MIDI file has to be specified; this is done by the configuration variable `midiFilePath` in the project settings (see figure 10). It gives the relative path of the directory containing the MIDI file from ltbvc.

The processing by the script is as follows:

- The referenced MIDI file is imported into new temporary tracks.
- All existing tracks with matching names (“«instrument name» - «MIDI file name»” are updated from the imported tracks.
- The new temporary tracks are deleted.
- Some filtering is done on the imported MIDI items: pan, reverb and volume control codes are removed (because they shall be provided by DAW controls and effects).
- All those items are set to “locked” (because they should not be changed manually, because they will be overwritten by the next import).

The example song has four MIDI tracks for each of the voices; figure 11 shows those tracks, their items with the appropriated names in the demo Reaper project.

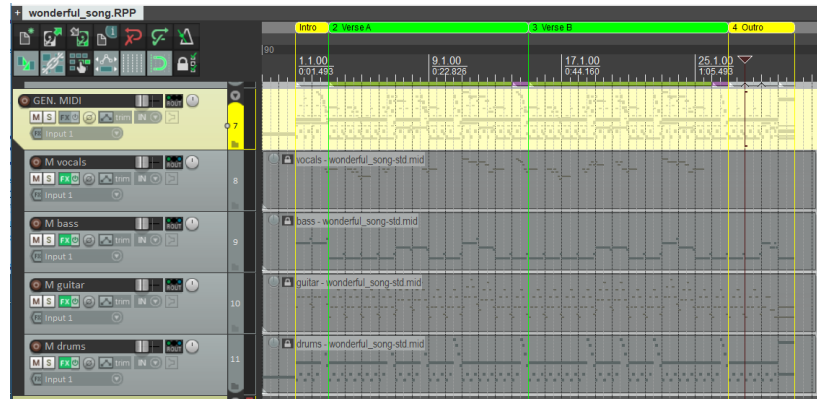


Figure 11: “Updateable” MIDI Tracks of Demo Song

## 3.6 MakeRegionsFromRegionStructureTrack

As mentioned in section 3.4 it is practical to structure the MIDI items according to the song structure. Reaper provides so-called *regions* along the timeline, which are very helpful in organizing a project.

Unfortunately they are a bit tedious to use: duplicating them or coloring them requires many clicks even when you use the “Region/Marker Manager” of Reaper.

The script `MakeRegionsFromRegionStructureTrack` simplifies this at the expense of using another track with the region information encoded into items. It makes regions from the MIDI items in a track called “STRUCTURE” by copying their positions and also reuses their coloring and naming. So you can quickly adjust the MIDI items in the structure track and then regenerate the regions from it.

It is also possible to vice versa generate that “STRUCTURE” track with this script.

When the script is started the dialog of figure 12 appears.

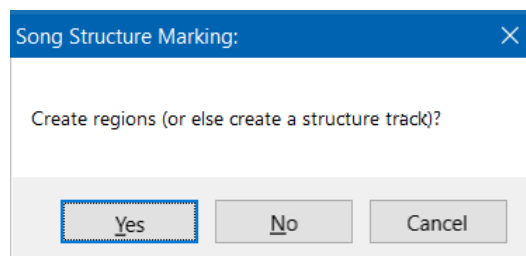


Figure 12: Selection Dialog for `MakeRegionsFromRegionStructureTrack`

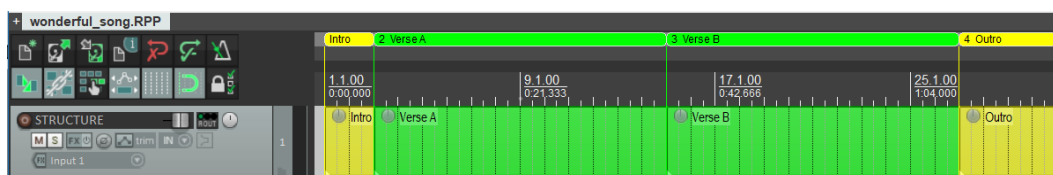
Depending on the selection (assuming you do not select cancel) the following



happens:

- For a selection of “Yes” (create regions) a track called “STRUCTURE” is searched for. First each region is deleted. Then for each MIDI item on that track a new region is created having the same start time, end time, name and color as the item.
- For a selection of “No” (create structure track) a track called “STRUCTURE” is searched for and is created if non-existent. Each existing MIDI item on that track is deleted. Then for each region a new MIDI item is created on that track having the same start time, end time, name and color as the region.

Figure 13 shows the structure track and the generated regions for the demo song.



*Figure 13: Regions and Structure Track in Comparison*

## 3.7 NormalizeStructuredVoiceTracks

The list of MIDI items for the structured approach as describe in section 3.4 should be located in tracks that adhere to some naming pattern.

In that case, the script `NormalizeStructuredVoiceTracks` can be used. It scans all MIDI items in structured MIDI voice tracks and normalizes their note velocity, quantizes note to the raster necessary for later export and removes unwanted MIDI control events.

Only those MIDI items in tracks are considered whose track names conform to a certain naming pattern. That pattern is — as with other scripts — defined in a line in the project settings using the variable `structuredMidiTrackNamePattern` and it specifies the regular expression a voice track name has to match. For example, the line `structuredMidiTrackNamePattern="S_.*"` in figure 14 defines the name pattern for the voice track names to be “S\_.\*”. This means the name of a voice structure track must start with a capital “S” followed by a blank character. Note that that is the default when you do not specify any pattern.

Each MIDI item in the voice structure tracks is transformed as follows:

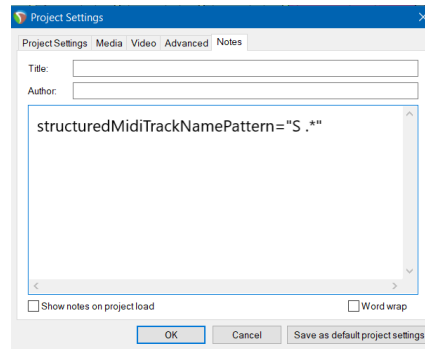


Figure 14: Settings for Selection of Structured MIDI Tracks

- Note velocities are set to a default value (80).
- Note positions and lengths are moved to the raster necessary for later lilypond export by the script `Exportlilypond`.
- MIDI control codes for volume, pan and reverb are removed (because they shall be provided by DAW controls and effects).

## 3.8 SelectLTBVCAudioSources

The track structure described in section 3.2 is very systematic, but it is tedious to switch between the different stages for a single voice.

For example, when you want to compare the MIDI track with effects applied to it with the refined audio track of the same voice and assuming the MIDI track is currently active, one has to

- Scroll to the MIDI track of the voice in the track or mixer panel.
- Mute that MIDI track.
- Scroll to the refined audio track of the same voice in the track or mixer panel.
- Unmute that track.

This gets annoying especially when the effects track of that voice is also involved.

Here the script `SelectLTBVCAudioSources` comes in handy: it presents all available audio sources within an LTBVC project in a selection dialog with a matrix of voices and audio sources. Additionally the final voice track may be muted or soloed for individual analysis.

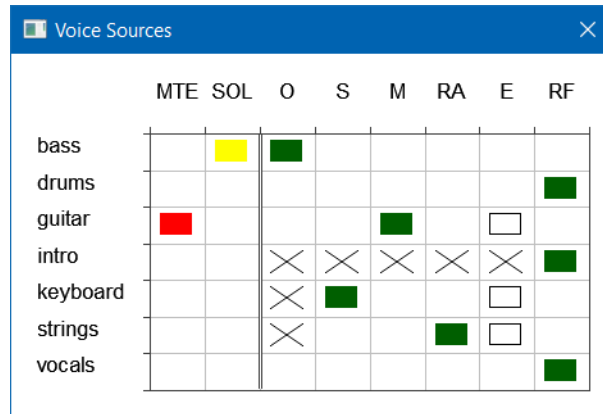


Figure 15: Selection Dialog for Audio Sources in LTBVC project

Typically you will assign this script to some key for quick access and then a selection dialog as shown in figure 15 occurs.

The voices are lined up in rows, the audio source tracks in columns with some code letters for each track category. Additionally the first two columns allow to mute or solo the final track for that voice.

In the above figure the following settings for the voices are active:

- **bass:** uses the “original” track and has its “final” track soloed,
- **drums, vocals:** use the “refinedaudio” track,
- **guitar:** uses the “generated MIDI” track, has additionally its “effects” track active and its “final” track muted,
- **intro:** uses the “refined audio” track and obviously has no other audio tracks available in this project,
- **keyboard:** uses the “structure MIDI” track and additionally has its “effects” track active, and
- **strings:** uses the “generated MIDI” track and additionally has its “effects” track active.

The “effects” track of some voice is automatically switched on and off, when some selected or deselected audio source is routed into them: hence a click into that column does not have any effect. Note that unavailable track kinds for a voice are shown as crossed out.

The audio source selection is changed by clicking on the appropriate cell in the matrix. MUTE and SOLO can be clicked at any time, but the other sources typically behave like radio buttons and deselect an existing selection.

For example, when clicking on some “refined audio” cell, a correspondingly selected “generated MIDI” and “effects” cell will be cleared.

A shift-click *adds* the existing cell to the selections for that voice. This is normally nonsensical, but may be helpful, when some tracks are phase-inverted. For example, the “refined audio” tracks in the typical configuration are: then you can check whether some renderings cancel out.

All the settings for the configuration dialog are again provided by the configuration file “LTBVC\_trackConventions.cfg” (see section 3.2 above).

The variables relevant for this script are `trackNamePatternAndHeaderList`, `trackNamePatternForVoiceEffects`, `trackNamePatternForVoiceFinal`, `trackNamePatternSentToEffectsList`, and `selectionDialogIdleDuration`.

`trackNamePatternAndHeaderList` is a list of pairs. Each pair contains a track name pattern signifying track categories (e.g. effects or raw audio tracks) as its first part and a column header for that category in the selection dialog. The list gives the order of columns in the selection dialog (preceded by the mute and solo columns).

For example, the list entry

```
trackNamePatternAndHeaderList =
...
" [ '^E%s+(%S.*)', 'E' ],"
...
```

tells that an effect track (starting with an “E” character and some white space followed by the voice name) has “E” as the heading in the selection dialog (as shown in the above figure).

`trackNamePatternForVoiceEffects` and `trackNamePatternForVoiceFinal` give the track name patterns for the special effects and final tracks.

For example, the line

```
trackNamePatternForVoiceFinal = "^F%s+(%S.*)"
```

tells that the final tracks start with the letter “F” followed by white space and some non-blank text (the voice name).

`trackNamePatternSentToEffectsList` is a list of track name patterns where those tracks are fed into the effects voice track.

For example, the lines

```
trackNamePatternSentToEffectsList = "["
" '^S%s+(%S.*)', "
" '^M%s+(%S.*)', "
" '^RA%s+(%S.*)' "
"]"
```

tells that tracks with the given name patterns (i.e. “structured MIDI”, “generated MIDI” and “raw audio”) are routed into the effects track and hence have to activate that track when selected.

Finally `selectionDialogIdleDuration` gives the maximum idle duration for the selection dialog in seconds. When nothing happens the dialog automatically closes after that duration. Zero means that the dialog stays open until explicitly closed.

As you might have noticed, in general the same strings for the patterns are often used in several places within the configuration file. So it is helpful to define auxiliary configuration variables for them and use those instead. The example configuration file in the `misc` subdirectory of the distribution uses that technique extensively.

---

## 4. Demo Song

Included in this installation is a Reaper demo file with tracks for the demo song of the `ltbvc`.

The following tracks are contained in the project:

- a region structure track,
- structured voice tracks for all four voices (vocals, bass, guitar and drums) routed to the corresponding effect tracks,
- the MIDI tracks from the generated MIDI file routed to the corresponding effect tracks,
- raw audio tracks (generated by the `ltbvc` pipeline) routed to the corresponding effect tracks,
- effect tracks enhancing MIDI or raw audio tracks, and
- refined audio tracks (generated by the `ltbvc` pipeline)

Because this project file should work with a standard Reaper installation without any additional plugins, the MIDI to audio conversion is done with the stock `ReaSampleomatic` plugin plus some external samples; also the audio effects for refinement are stock effects. This is acceptable for trying out the plugins described in this document, but does not faithfully reproduce the sound of the external pipeline.

For a realistic impression of the external MIDI to audio conversion, one also needs a soundfont player fully compatible with the `fluidsynth` player [`Fluidsynth`] used in the `ltbvc` pipeline. One candidate is the `FluidSynthPlugin` that almost faithfully emulates the external `fluidsynth` player (apart from some deviations described in its documentation [`FluidSynthPlugin`]).

If you want to emulate the SoX effects from the external pipeline (as mentioned in the `ltbvc` documentations) you'll have to install specific SoX emulation plugins (for example, the SoX plugins from [`SoXPlugins`]) and then adapt the effect tracks accordingly.

## 5. Coping with Errors

If one of the scripts does not work as expected or even issues an error message, how can you find out what really went wrong?

All the scripts do some fine-grained entry-exit-tracing of relevant function calls into a log file; its last lines should give you some indication about the error.

The log files are written into a directory given by the environment variables REAPERLOGS, TEMP and TMP (in the order given). If none of those variables is set, the directory “/tmp” is used.

The log file name is “reaper\_” followed by the script name, so, for example, the script exportLilypond writes its log to the file “reaper\_exportLilypond.log” in the directory given by one of the environment variables given above.

Figure 16 shows how a log file looks like. Each line shows either an entry of a function (“>>”), an exit from a function (“<<”) and a log line within a function (“--”) together with a time indication.

```

...
>>Reaper.Project.current
>>Reaper.Project._make
<<Reaper.Project._make: Project(0)
<<Reaper.Project.Current: Project(0)
>>ImportMIDI.findMidiFileFromTracks: Project(0)
>>Reaper.Project.trackList: Project(0)
>>Reaper.Project.trackCount: Project(0)
<<Reaper.Project.trackCount: 21
>>Reaper.Generics.makeList: container = Project(0), kind = Track, count = 21
--Reaper.Generics.makeList: processing element 1
>>Reaper.Project.trackByIndex: self = Project(0), index = 1
>>Reaper.Project.trackCount: Project(0)
<<Reaper.Project.trackCount: 21
>>Reaper.Generics.findElementByIndexRaw: container = Project(0), index = 1
>>Reaper.Track._make
<<Reaper.Track._make: Track(id = '{93AA75D3-5F86-49B1-AFF1-31DC7C0C349C}', name
= 'STRUCTURE')
<<Reaper.Generics.findElementByIndexRaw: Track(id = '{93AA75D3-5F86-49B1-AFF1-31
DC7C0C349C}', name = 'STRUCTURE')
<<Reaper.Project.trackByIndex: Track(id = '{93AA75D3-5F86-49B1-AFF1-31DC7C0C349C
}', name = 'STRUCTURE')
...

```

*Figure 16: Extract from a Log File (for ImportMidi)*

## Bibliography

- [Fluidsynth]     *FluidSynth - Software synthesizer based on the Sound-Font 2 specifications.*  
<https://fluidsynth.org>
- [FluidSynthPlugin]     Thomas Tensi.  
                  *FluidSynth-Plugins*  
<https://github.com/prof-spock/FluidSynthPlugin>
- [LTBVC]             Dr. Thomas Tensi.  
                  *LilypondToBandVideoConverter - Generator for Notation Backing Track Videos from Lilypond Files.*  
<https://github.com/prof-spock/LilypondToBandVideoConverter>
- [LTBVCPlugins]     Dr. Thomas Tensi.  
                  *Reaper Plugins for the LilypondToBandVideoConverter.*  
<https://github.com/prof-spock/Reaper-LTBVC-Plugins>
- [ReaPack]            Christian Fillion.  
                  *ReaPack: Package manager for REAPER.*  
<https://reapack.com>
- [Reaper]             Cockos Incorporated.  
                  *Reaper Digital Audio Workstation.*  
<https://reaper.fm>
- [SoXPlugins]         Dr. Thomas Tensi.  
                  *SoX Plugins - A Reimplementation of the SoX Commandline Processor as DAW Plugins.*  
<https://github.com/prof-spock/SoX-Plugins>