mlrVST User Guide

Ewan Hemingway
info@ewanhemingway.co.uk

May 19, 2012

Contents

1 Introduction

mlrVST is an audio plugin for manipulating audio & MIDI samples using the monome controller. It is inspired heavily by both Brian Crabtree's brilliant and original `mlr` application, and the fantastic `mlrV` reboot by Trent Gill (galapagoose) and Michael Felix (%).

What mlrVST brings to the table is the stability and simplicity of the VST. Interaction with DAWs such as Ableton is made much simpler, no setup is required with rewire etc. You can easily resample the audio from other sources within the DAW such as other synths or external audio inputs.

One of the important shifts in direction from previous incarnations is towards an experience where minimal mouse / screen interaction is required. Each row on the monome can act as a control strip for changing volume, playspeed etc whilst a modifier button is held. For full details on this and other new features, please refer to the manual (section 2.3).

The plugin is completely cross-platform[1], the full feature-set should be available to Windows, Mac and Linux users. Finally, mlrVST is written in fully commented, open-source C++ and as such aims to be accessible to anyone who wants to hack in new features of their own (see Part 3 for details).

1.1 Installation / Setup

Windows To install the plugin in Windows, just copy `mlrVST.dll` to your VST plugins folder. For example, in Ableton Live this folder can be found in `Options` → `Preferences` → `File/Folder` → `Plugin Sources`.

Mac First extract the `mlrVST.vst` file from the archive. This should be placed in the folder `~/Library/Audio/Plug-Ins/VST` where it should be visible to Ableton and other hosts.

Linux Coming soon!

1.1.1 OSC

Unfortunately, for now the plugin still uses the old **OSC** spec (so MonomeSerial users can skip this section). Those with serialosc will need to use the monome-serial patch to emulate this, with prefix `mlrVST` and standard input:output ports of 8080:8000.

[1] Linux support is lower priority during initial development due to unfamiliarity but any help is welcomed - working builds have been successfully produced

1.2 Quickstart

If you want to just start playing with mlrVST, start by dragging a sample onto the first horizontal strip. This strip maps the sample to one of your monome's rows. The top row is reserved for various functions, but the remaining rows can be used to play back.

So now press a button on the 2nd row to start playing. You should hear that the sample starts playing from the point at which you pressed, e.g. pressing half way along the row starts the sample half way through. This should be you up and running!

2 Manual

2.1 The Basic Idea

The basic idea of mlrVST is that each row of the monome is mapped to an audio sample in the program. Pressing a button on that row will jump you to that point in the sample, e.g. pressing the 5th button on a 8x8 sized monome will jump you to half way through.

Figure 2.1: Schematic of basic operations

The top row of the device is reserved for various mappings (such as recording and resampling): these can be customised using the `Mappings` button. The remaining rows can be used to control samples.

2.2 The Interface

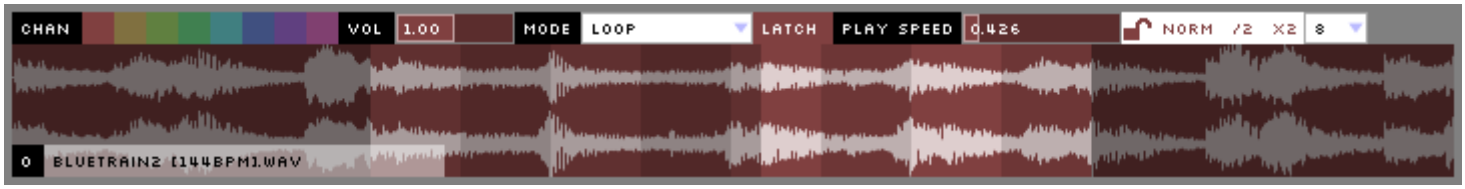

Figure 2.2: An example SampleStrip

The main interface consists of a number of `SampleStrips` such as the one in Fig. 2.2. These strips map onto the rows of the monome, bearing in mind that the top row of the device is reversed for mappings. Each strip has the following settings (left to right along the top of Fig. 2.2):

Channel If multiple strips have the same channel, only one can be playing at once. For example, this could be used creatively if you wanted to switch back and

forth between two strips without stopping and starting. Channels are specified by colour and have their own volume control.

Volume Each individual strip also has it's independent volume.

Mode There are several playmodes available:

- **loop** - this just loops the full sample endlessly. You can also loop over a certain range of the sample (called *inner looping*) by holding the button representing the leftmost loop point, *then* pressing the button for the rightmost point.
- **play chunk** - the sample is broken up into a set of chunks (for each column on the monome): this mode just plays the chunk associated with that column.
- **play to end** - fairly self explanatory: play to the end of the sample and then stop.

Latch By default button presses are "latched", i.e. if a button is lifted, playback still continues. This option allows the user to only have the sample playing back while the button is pressed down.

Playspeed This varies the speed at which the sample is played back (higher speeds will sound more high pitched).

Lock playspeed (padlock icon) By default the playspeed will be adjusted so that the sample stays in sync with everything else if the tempo is changed, or a if different selection is made. To keep the playspeed locked at one value, set this to locked.

Norm / **Rev** This just controls whether we play the sample forwards or backwards.

x2 /**2** These double or half the playspeed respectively. In musical terms, this is equivalent to changing the pitch by an octave.

Number of chunks This sets how many parts the sample is divided up into. By default this is the same as the number of columns, but you might change it if you wanted to use samples of a different time signature.

Using the mouse You can also use the mouse to select part of the sample. If `[Ctrl]` is held, only the edge of the loop nearest the mouse will be changed. This is useful if you just want to make an adjustment to the selection. `Double clicking` will select the whole sample. If `[Ctrl+Shift]` is held, the selection will snap to 16ths of the sample. To select a sample, `[Right click]` and hold: here you can choose to select an audio sample or one of the recordings / resamplings.

2.3 Monome / MIDI / OSC Mappings

One of the new features in mlrVST is the introduction of modifier buttons. When held, these turn each row into a set of controls. This means the musician can control things like playspeed or volume without leaving the monome. By default, the 2 leftmost buttons on the top row are modifier buttons. Each button can enable a different set of maps: to customise these, click on the `MAPPING` button.

Coming soon: MIDI / OSC mapping.

3 Developer Information

3.1 Compiling Yourself

The mlrVST plugin is written using the cross-platform JUCE library in C++. As as result the code should work on Win, Mac and Linux without too much trouble, though a couple of minor changes may need to be made if you want to add platform dependent code (font rendering for example).

First you will need to download both the latest version of JUCE and the VST SDK from Steinberg (SDK version 2.4). JUCE uses an internal program called the Introjucer to produce build targets and generally manage the project. The first step should be to build and run this (it can be build using the appropriate project files in `/juce/extras/Introjucer/Builds`).

Once this is done, you can open the `mlrVST.jucer` project file using the Introjucer to inspect the project settings etc. New files should be added here so they can be added to all of the build targets. Also at this point you should make sure that the paths to JUCE and the VST SDK are correctly specified (for example as in Fig. 3.1).

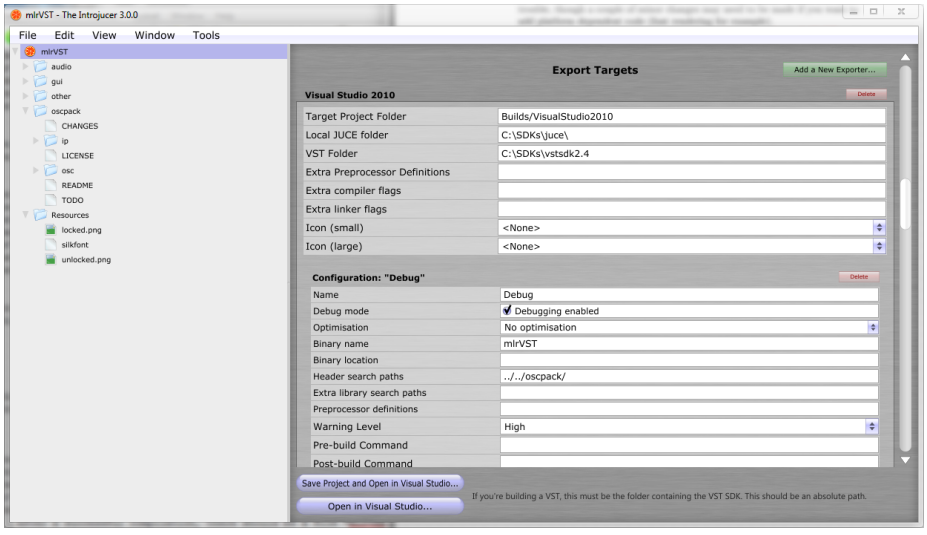

Figure 3.1: Introjucer settings for Visual Studio 2010

Windows Just open the `mlrVST.sln` file in the `/builds/Visual Studio 2010` folder and you should be ready to go. After a successful compilation, there should be a file `mlrVST.dll` in the Debug (or Release) folders - this can just be copied into your host's VST folder.

Mac First open the `/Builds/MacOSX/mlrVST.xcodeproj` project file to load the project into Xcode. Apple's LLVM compiler finds an error with one of the VST files `audioeffect.cpp` which you can simply autofix. Alternatively, use the GCC compiler. If you want to build AU plugins, you will need to follow the instructions here.

Linux Apparently builds fine. There is a `Makefile` generated, but I've not checked this thoroughly recently.

3.2 Guide to Key Classes

3.2.1 Audio Processing

PluginProcessor The guts of the application are in this class. This is what deals with the VST code, handling the overall audio processing. As the `PluginEditor` is destroyed when you close the GUI, everything that wants to be persistant must be saved here. This is why you may see what looks like redundancy between similar objects (why we need `SampleStrip` and `SampleStripControl` for example). The sample pools are also stored in this class, i.e. the objects which store samples in memory.

SampleStrip Each row on the monome controls a `SampleStrip`. This class handles the audio processing for that Strip, and stores the relevent parameters (e.g. playspeed, volume). Any changes to a `SampleStripControl` (the GUI analog) should update the corresponding `SampleStrip` parameters using `setSampleStripParam()`.

3.2.2 GUI

PluginEditor This is the main GUI class that handles drawing the components, anything visual really. This should really just forward any commands to the `PluginProcessor` class and not do very much itself save for handling user input.

SampleStripControl These represent the rows of the monome that are used to manipulate samples. This class allows the user to use the GUI to change parameters (such as volume, the selection of the sample etc), and does so by relaying this information to the `SampleStrips`.

3.2.3 Other

OSCHandler Basic class which processes any OSC messages to / from the monome. Nothing too exciting here, but when we update to the new OSC spec changes will need to be made here.

3.3 General points for developers

The code is still in the beta stage: I have found in Windows that if a Release build hangs in your VST, a restart may be required. If you're running the plugin from Visual Studio (using the `PluginHost.exe` included in utilities) then you can handle crashes without needing a restart.