

About VSTunel

Some software music instruments like the Moog Modula V (from Arturia, pictured) can be difficult to use due to the number of controls they have. In a real-life musical setting, and especially during live performance, the goal is usually to make the changes in sounds or sound parameters as easy and intuitive as possible, so that the musician can better concentrate on other tasks that might matter more, such as technical precision or communication with others. I have observed that when I want to make changes to the instrument and its sound using the software GUI, these tend to repeat themselves - e.g. setting the filter value to a higher value for the chorus and returning it back for a verse, or turning on the distortion on during a solo and back when the solo is finished. In these types of situations, a clogged GUI can be counter-productive: you always have to locate the (often very small) control element desired, which can take some time, precisely when you do not want to risk a mistake.



Figure 1 - The rather sophisticated Moog Modular V GUI (© Arturia)

This was my motivation for building VSTunel, a MIDI VST module that allows the user to use a simple graphic interface (see below) to control assigned parameters via the MIDI interface.

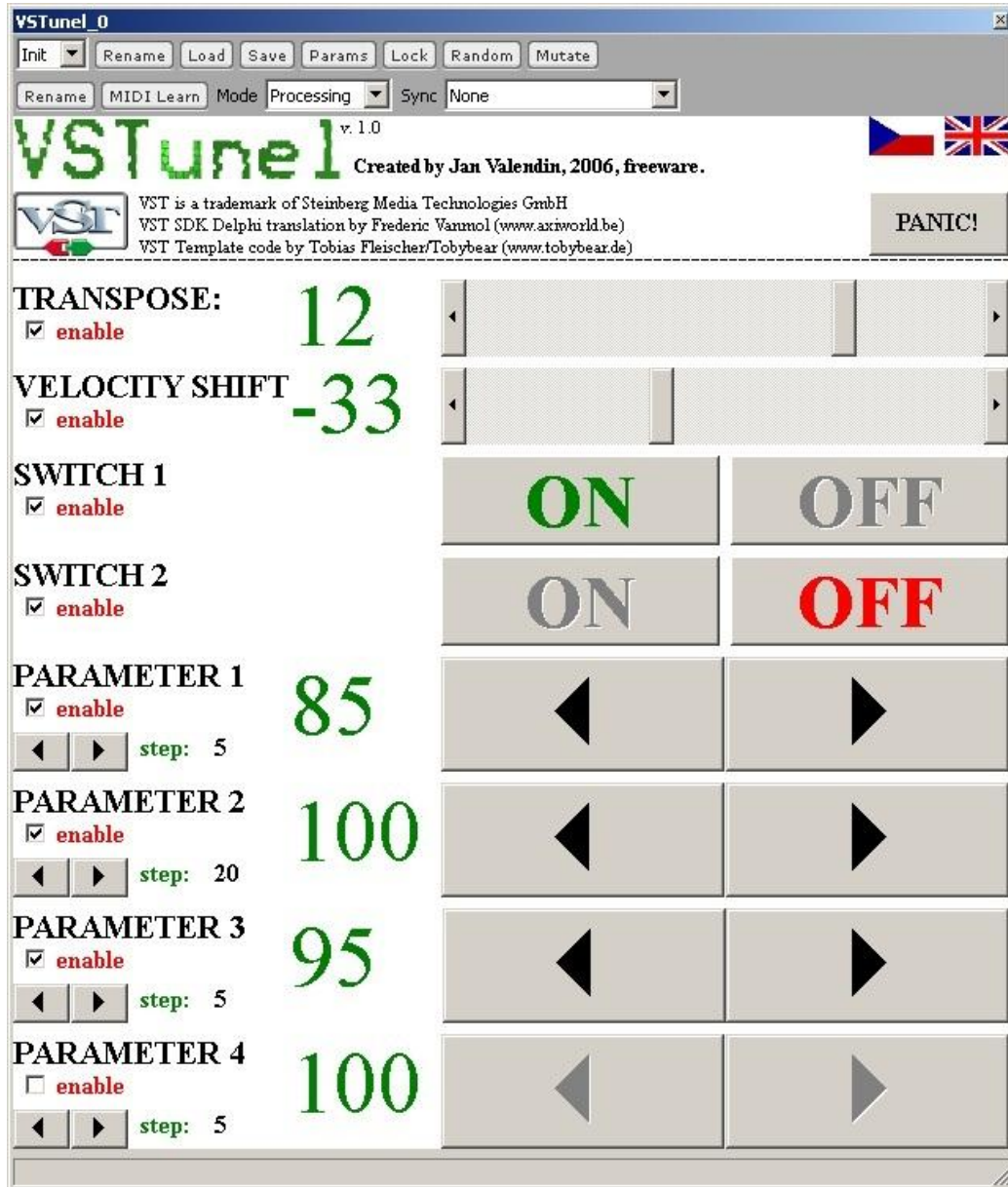


Figure 2 - The VSTunel GUI

From top to bottom, the player can use 8 different channels to alter the MIDI messages as they pass through the module from the electronic control instrument to the software instrument, or to generate controlling messages that are sent to the software instrument.

On the top there is a Transpose slider that adds the chosen value of semitones to the incoming MIDI note values, ranging from -2 octaves to +2 octaves.

The Velocity shift slider adds or subtracts velocity (how forceful the attack was) in a similar fashion to the Transpose controller.

Next are two switches, which allow the musician to change two different Yes/No values of the software instrument. These two cause the module to generate MIDI_CC (*Controller Change*) messages that are sent to the software instrument.

Last four control elements can be used to control various parameters of the software synthesizer, depending on the user's assignment. Each sends a MIDI_CC message with a different channel number. The user can set up the size of each step and then using the arrow buttons he may adjust the parameter up and down to his liking.

Each of the controls can be enabled and disabled using the associated *Enable* checkbox to prevent accidents.

At the very top of the window is the *Panic* button, which allows the player to send a batch of MIDI_OFF messages to the software synth to get rid of "stuck" notes. (Note: this used to be a common issue with older/low end MIDI interface hardware, especially when a lot of notes would get pressed at the same time, random notes would get "stuck" and hold until the user would press the same note again. Modern USB MIDI interfaces do not seem to suffer from this issue anymore.)

Top right are two flags, which allow to change the labels in the interface from Czech to English and back.

The buttons have been made rather large so that the player will not have a problem with the occasional misplaced click.

If the player needs more controls than a single instance of VSTunel can provide, he can run more of them without them interfering with each other.

Programming notes

VSTunel was developed in Borland Delphi and it is organized as a single dynamic linked library. The library is used to acquire the supported version of the VST interface and create an instance of the Aplugin class. There are two other modules included in the library.

The uPlugin.pas file describes the Aplugin object, descendant of the TVSTTemplate class. It uses the following procedures:

- Constructor `Aplugin.Create` is used to define the number of audio inputs and outputs (`numInputs`, `numOutputs`), the number of parameters (`numParameters`), the name of the module as displayed in the host program and other abilities of the module (`canDos`). These abilities feature `sendVstEvents`, `sendVstMidiEvent`, `receiveVstEvents`, `receiveVstMidiEvent` (the module can create and receive VST events and VST MIDI events). These are all included in VST version 2.0 and higher. There is also the option to assign other capabilities of the module to the `Property` array.
- The `Aplugin.initializeParameters` is used to set up the properties of all parameters, like the initial, maximal and minimal values, the parameter names, and the behavior upon change. Using the `smoothingFactor` value we can for example control the speed of the module's reaction on a parameter change, i.e. how quickly the value of the parameter changed reaches the final value from the initial one.
- Method `Aplugin.processMIDI` takes care of the processing of all incoming MIDI messages. The two parameters that use sliders in the GUI, `Transposition` and `Velocity`, will affect the incoming MIDI messages accordingly, before they are sent out to the software synthesizer. The procedure is also used to generate new MIDI messages as a reaction to the incoming ones.
- Method `Aplugin.editorIdle` is used to access the GUI elements in the other direction, i.e. when it is necessary to change the control element so that it reflects a change of a parameter.
- The `Aplugin.parameterChanged` allows us to react on a change of one of the parameters. Whenever the user clicks on one of the buttons or moves one of the two sliders, the plugin will generate `MIDI_CC` messages based on the change.
- The `Aplugin.processAudio` method has been implemented only for compatibility reasons, as the `VSTunel` module has no use for any incoming audio signal, but some hosting programs require effect modules to have at least a stereo audio in and a stereo out port. In the `VSTunel` module, the audio data are sent to the output automatically without any processing, so the audio passes through the module unchanged.

The `uEditor.pas` file describes the visual aspect of the module, the GUI. In the core is the `TpluginEditorWindow` form representing the main window of the module with the control elements. The unit contains procedures and functions that are triggered by changes in the user interface. The important ones are:

- Method `TpluginEditorWindow.UpdaterTimer` facilitates the visual change of the control elements that matches the actual state of the parameters and their inner value. This happens only if there has been a change, that is, the `Effect.editorNeedsUpdate` flag is set to `True`.
- Method `TpluginEditorWindow.OnEditorOpen` is called whenever the module window is opened or re-opened to initialize all elements based on the current values.

Installation and setup

The user is only required to put the VSTunel.dll file representing the module into his VST Plugin directory. In the Windows XP environment this is typically located in “C:\Program Files\Steinberg\Vstplugins,” but some host programs have their own locations. Most hosts will also automatically re-scan the directory with the plugins upon launch.

After the module is made available to the host program, it needs to be connected to a MIDI-IN port of the computer’s audio device. This is the port into which the MIDI controller (i.e. electronic keyboard) sends the MIDI signals the player produces on a physical interface (i.e. keyboard). The output port of the module needs to be assigned to the software synthesizer of choice. The module can work without the MIDI input, in case the user does not need to process or filter the incoming messages and only wants to generate control signals via the GUI. The connection process is easy enough in host programs which support manual wiring of the active modules, like the Plogue Bidule (pictured). In others it is necessary to use it as a *Send Effect*.

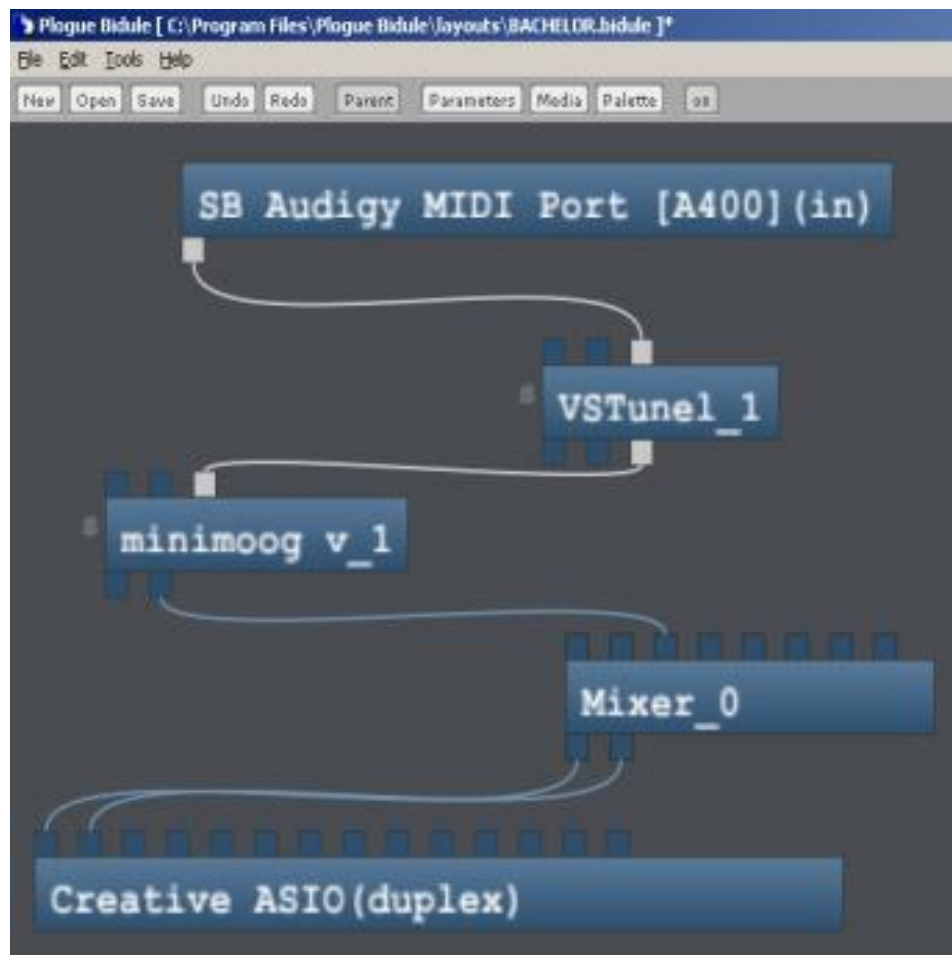


Figure 3 - Typical setup of the VSTunel module