TVSTInstrument API

Version 1.3 19 March 2019

In this framework you find Delphi wrapper code to create VST2, VST3 and FruityPlug Instruments.

For this, there’s a translation between a new wrappertype TVSTInstrument and the various platforms.

Instead of having three versions for these platforms you create a derivative from TVSTInstrument.

An example of this is the class TMyVSTPlugin, found in the SimplesynthCommon directory.

In this document the API of TVSTInstrument is discussed.

In the document Description SimpleSynth the working of the simplesynth is explained.

You can read this and that document in any order.

To make the TMyVSTPlugin available as plugin, you have to describe it in a VSTInstrumentInfo record and make it available through the function GetVSTInstrumentInfo. If all goes well your plugin will be instantiated in your DAW. From there you have control over it with this API.

Please use the RMSMyVST2, RMSMyVST3, FruityPlug projects as a starting base as they include the startup code.

Declaration:

The function GetVSTInstrumentInfo must be implemented delivering the following information:

vst3id A unique UUID for you’re a VST3 plugin

vst2id: A unique 4 letter string for a VST2 plugin

cl The class name: here: TMyVSTPlugin. It must me derived from TVST3Instrument.

Ecl The class name for your editor: here: TFormMyVST

isSynth true if this is a synth.

softMidThru if you want to pass all midi events to the next plugin

and a few other information fields.

API. The API is split into two parts:

**Processor**

Methods concerned with the audio processing part. You should not update the UI from a Processor call.

**Controller**

Methods concerned with the parameters, presets and editor

|  |  |
| --- | --- |
| **Processor** | All methods are virtual and optional for processing |
| proc OnSysexEvent(s:string); | s starts with $F0 and ends with $F7 |
| proc OnMidiEvent(byte0,byte1,byte2:byte); | Remark: NOT called from UI Thread |
| proc MidiOut(const b1, b2, b3: byte); | Remark: NOT called from UI Thread |
| proc Process32(samples,channels:integer;  inputp, outputp: PPSingle); | Here you process your audio. Inputp and outputp are arrays of array with the first subscribe the channel and the second the sampleposition. See example. |
| proc OnSamplerateChanged(samplerate:single); |  |
| proc OnPlayStateChanged(playing:boolean;ppq:inte); | Called when the DAW changes play state, or position |
| proc OnTempoChanged(tempo:single); |  |
| proc updateProcessorParameter(id,value) | Virtual method called when a Host parameter changes (NON-UI thread). You should update your ‘Processor’ or ‘Model’ |

|  |  |
| --- | --- |
| **Controller** |  |
| proc OnInitialize; | Called after creation. Here you should add your parameters |
| proc AddParameter(id:integer;title, shorttitle,units:string;min,max,val:double) | Adds a parameter to the system. Each parameter has an unique id and some other values |
| func getParameterAsString(id:integer;value:double):string; | Value is between 0 and 1! |
| proc updateHostParameter(id,value) | Updates a parameter on the Host |
| proc updateEditorParameter(id,value) | Virtual method called when a Host parameter changes (UI thread). You should update your UI. |
| proc ResendParameters | Resends all parameters through UpdateEditorParameter (must be called from UI thread, e.g. when opening your plugin editor) |
| function getEditorClass:TformClass; | Virtual method. You can override the default creation of the editor class (as defined in GetVST3InstrumentInfo) here (normally not needed) |
| proc OnEditOpen | Virtual method. Called when the editor opens. It could be wise to call ResendParameters and to create a mechanism so when a UI element changes you send the changed value to the host. |
| proc OnPresetChange(prgm:integer); | Virtual method. Only needed if you want to show the preset number in the UI. |
| proc OnEditClose | Virtual method. For cleanup, but normally not necessary. |
| proc OnFinalize | Virtual method. For cleanup, you should release all resources, especially timers. |
| Proc OnEditIdle | Called regularly if the Editor is open |

Missing from the API (will be implemented on request)

* Setting/Getting preset names. Setting the preset number (to host)
* MidiOut (for VST3)
* MidiIn stuff like MPC, aftertouch (for VST3)

== internal remarks for Ruud ==

Midi In / CCs and Note On/Off and Parameters

The system generates Midi events and simulated Parameters, but how does it work?

|  |  |  |  |
| --- | --- | --- | --- |
|  | VST2 | VST3\* | FruityPlug |
| OnMidiEvent | HostCallProcessEvents  This seams to be called from UI Thread.. | Notes: Processor thread, not UI.  Parameters (CC):  Processor thread, not UI. | MidiIn procedure  (GM) (GUI / Mixer Thread)=ThreadSafe |
| UpdateEditorParameter | HostCallSetParameter NOT from UI gets Cached and send in OnIdle.  Also calls to UpdateHostParameter are treated the same way | Controller thread, UI thread | See VST2 |
| UpdateProcessorParameter | HostCallSetParameter and  UpdateHostParameter call this, on NON-UI | Processor thread, not UI. | See VST2 |
| UpdateEditorMidiEvent | Not used | Parameters(CC)  Controller thread, UI thread | Not used |

In VST3 MIDI CC is constructed with Parameter Changes.

There is a BIG difference between OnMidiEvent for VST3 vs. VST2/Fruityplug..

My plugins process OnMidiCC as follows:

* Parameters CC event: -> Update Model (=Processor) and call UpdateHostParameters
* Note Events -> Update Model (=Processor) and UI::Keyboard but this does not use the UI

“Thus”, this works with all formats: VST2 and FP use the Loopback in HostCallSetParameter. VST3 works because UpdateEditorMidiEvent gets called.

* Note that in Fruity Loops the parameter trick does not work (and MIDI CC don’t get called) and that in Reaper I get ‘faultly’ receive on a ParameterCC message (luckily!)

So, for now MIDI support is at least to say ‘buggy’…but it’s not my fault.. (nor Steinberg)

OK, I changed this for VST3. There is now an OnIdle thingie…