Real-time audio plugin architectures

a comparative study

—

IRCAM – Centre Pompidou

Vincent Goudard and Remy Muller

September 8, 2003

Abstract

This document is born after a study of the existing standards for real-time audio plugins. It aims at giving an idea of the underlying structure behind all these incompatible standards, and pointing out differences when they occur.

Contents

Chapter 1

Introduction

Currently, there is more than ten audio *plugin standards*, basically one per major host manuacturer such as Steinberg or Digidesign but Microsoft and Apple have also developped multimedia APIs[1] integrated to their OS[2] that include audio plugins. These standards share a lot of features and behaviours while remaining incompatible. Many converters have been released these last years for breaking those incompatibilities, but they have some unwanted drawbacks: they increase the CPU charge and they are often limited to an OS. In addition, porting plugins from an OS to another does take time especially when dealing with graphics or filesystem access. Many people have tried for their own use to develop tools to abstract from the plateforms and the standards when developping plugins, thus reducing the cost of developpement and easing developers' life. Unfortunately few of these tentatives have been released publically.

The purpose of this document, beyond plugin formats, is to identify what is the specificity of audio plugins, what is common to all standards and what may differentiate them.

We do not pretend to write a tutorial for writing cross-plateform/cross-standards plugins nor to be exhautive but rather give an overview of what one may encounter when trying to target different plateforms and standards, and look for a common model that can be extracted.

[1] Application Programmers Interface.
[2] Operating System.

Chapter 2

What is an audio plugin?

What is its purpose?

A plugin is intended to extend an audio creation environment by the use of dedicated third party libraries. Thus any user should be able to find plugins that fits perfectly their needs to customize this environment. Typically plugins can be considered as *virtual devices* based on the analogy with hardware devices such as reverbs, compressors, delays, synthetizers, samplers or beatboxes that can be connected together in a modular way. This allows host manufacturers to focus on the conviviality and efficiency of their products while specialized manufacturers rather focus on the *Digital Signal Processing* part.

Approaches

We can distinguish 3 approaches to do plugins

- Plugins integrated into a multimedia API like Apple's Audiounits and Microsoft's DirectX.
- Plugins related to a reference host like VST, RTAS, MAS[1]. Some of them can be found on different plateforms like VST or RTAS.
- Modules attached to modular softwares like the "Max family"[2], Buzz or EyesWeb.

We should also consider LADSPA on GNU-linux which is the de-facto standard for that plateform but that can not be attached to any of the above categories.

[1] respectively cubase, protools and digital performer

[2] Cycling 74's Max/Msp, Ircam's jMax an CRCA's Pure Data.

2.1 General model

A plugin treats an **audio signal** then it needs to be **controlled** and in order to work in a host environment, host and plugin have to know their repective setup & properties.
The audio stream is usually treated by blocks (buffers) inside whom samples are synchronous to the samplerate. In general, control parameters have a lower refresh rate and can be either treated between audio blocks – on-demand – if the value has changed, or queued with time-stamps relative to a position in a block for audio-rate resolution. Properties are mostly static flags but some can be dynamic though.

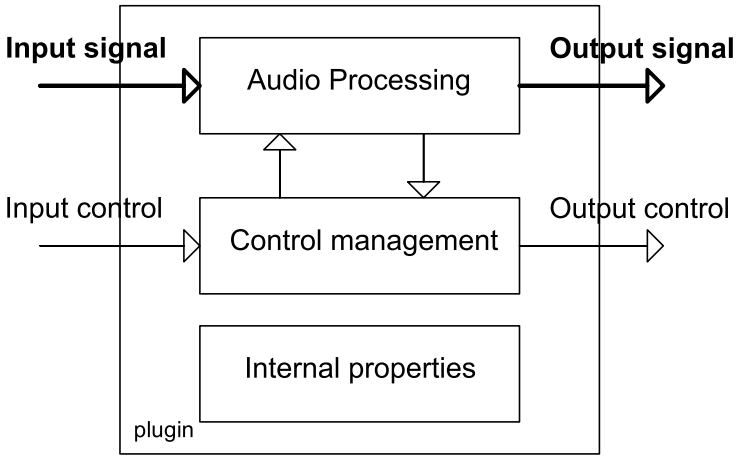

Figure 2.1: A common model behind musical plugin architetures

Setup & properties

Plugins have to know about the host's capabilities like sending/receiving events, providing multichannels busses... In addition there are lot of situations where we would appreciate that plugins can know more about the *musical context* (i.e. tempo, score-position, metric...) than just audio and control data.
The plugin has to give information about its ins/outs, their specificities such as side-chain, mono, stereo or surround and many things such as its latency[3] or its tail-time[4]. There is also some hosts that allow plugins to give them commands

[3]An algorithm like a Fourrier Transform may introduce a pure delay in the signal chain because it needs a minimum number of sample to start. In a sequencer (not in real-time!) this pure delay can be compensed by the host by sending data to the plugin in advance.

[4]Basically all the algorithms based on convolution may introduce a tail i.e. output samples even when there is no more input data (e.g. a delay, a reverb a FIR filter...)

such as *play, stop or set tempo* but it is not really wide-spread.

2.2 Inputs – Outputs

To understand what is relevant to determine the number and the kind of IOs, we may analyse what plugins algorithms does around 3 different of kind functionnalities – analysis (**A**), transformation (**T**) and synthesis (**S**) – that can be either alone or mixed together in the same entity.

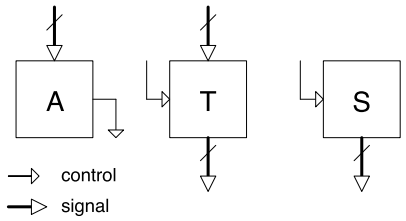

Figure 2.2: functions

Moreover, plugin setups highly depend on what the host allows and how the plugin will be attached to its environment. While most hosts only support mono and stereo (even if surround begins to be popular with the recent success of the DVD and the famous 5.1 surround format[5]), we can also distinguish between effects that will be treated as **inserts** (integrated in a chain on a single bus), **sends** (they can be shared by many voices in a mixing-console and the output is summed on the master bus), **instruments** (no audio input) or simply integrated into a modular host.

Audio

The number of **audio inputs and outputs** vary from one plugin to another, they can be grouped into busses that can be mono, stereo, surround... but they mostly depends on the host capabilities. Moreover, in some standards – the number is still growing –, it is possible to define side-chain[6] (see figure 2.3).

Control

Though it is possible to build a plugin that has no way to be controlled, like a phase inverter, it is of limited interest in a musical context where tweaking the

[5]left, center, right, rear left, rear right plus subwoofer

[6]a side-chain channel can be analyzed or used directly to modify the main audio stream processing.

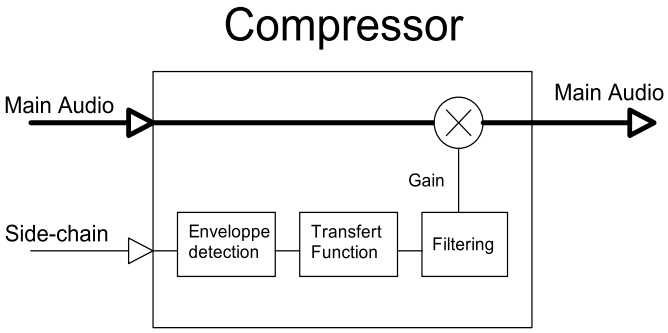

Figure 2.3: A side-chain example: a compressor

parameters or sending notes is part of the creation. You may want to both change the plugin's parameters and record their variation for later playback. Hence most plugins should provide control input and output at the same time. Most of the time hosts and plugins notify each other of parameters' evolution in a simple way but one can also find callback and polling mechanisms. In addition sending **control data** between plugins can sometimes be performed either directly via parameters (LADSPA) or by an event-oriented protocol like MIDI (VST).

2.3 The “process”

All plugins have this function in common often called “process”. This is where the most important thing happens: *audio processing*. All the current standards treat audio as input and output buffers (of given length), that can be given as floating-point sample arrays, or by more sophisticated structures with additionnal information. Processing can be in-place, in which case in and output buffers are the same (i.e. they share the same memory location, processing is thus destructive.), or buffer-to-buffer, where they are different in this case it is possible to have either accumation (send effects) or replacing (inserts). Parameters are usualy known before the process starts, but you may have to update them by hand. For simple audio processing like biquadratic filtering, it should be enough to use directly the buffer provided, but when doing FFT processing for example you may want to re-buffer audio samples to fit your internal buffer size in which case you may introduce additionnal latency in the signal chain.

2.4 The user interface

The GUI should be considered at least as important as the audio processing algorithm because it is the filter thru wich your plugin will be perceived. It is a key-point for the conviviality and usability of a plugin, everything should be fast and easily tuneable in a precise way with a convenient visual feedback. In particular, parameters' mapping can really make the difference in the way people feel about how your plugin sounds.
The GUI can be either generated automatically by the host with the help of the information or hints that the plugin can provide about itself (its parameters type, their range and units..., and the type of control that should be used – slider, combo-box, switch, knob, 2D controler, ...–) or provided by the plugin manufacturer to reach a higher level of customization.
However we will skip the GUI topic in this document. We consider control parameters as the plugin interface without taking account the way these parameters are changed by the user.

2.5 Plugin developpement

The API defines the layer through which plugins and hosts can see each other and also the tools that can be used to ease the developpement step.
The Host needs different information than the user about plugins such as their ID, their capabilities, their setup, the number of parameters or their names. The host needs to receive computer-readable data in the way it is programmed to. In addition most of the SDKs[7] provide a set of tools, functions, utilities. In the same effort, they often allow the use of OOP[8] languages such as C++, ObjectiveC or Pascal to ease the modelisation step and the tasks distribution inside a same plugin.

[7]Software Developpement Kits.
[8]Object Oriented Programming.

Chapter 3

The signal stream

3.1 Nature of the signal stream

Most of the time, the signal stream is represented as floating-point samples in the intervall $[-1; 1]$ full-scale. However some standards supports fixed-point samples[1] for historical or technical[2] reasons .
For efficiency purpose, those samples are packed into buffers whose size can be fixed or variable from one call to the other. Though sample-by-sample processing[3] is already used in some DSP libraries, there isn't already any standard doing the processing that way. The samplerate is supposed to be constant (and known) at least during the length of the buffer and may change at the buffer-rate though it is unusual[4].
Audio buffers can either be surrounded by additionnal information like time-stamps[5](DXi, EyesWeb, AU), samplerate, buffer-size... or transmitted (most of the time) as simple arrays.
Some plugin API – DX, AU – allow channels interleaving for compatiblity with files types and streaming protocol, but it is quite marginal and tends to disappear at least for real-time processing. The most common (and easiest) way is to transmit channels as separated mono buffers.

[1] e.g. The CD format uses 16-bit signed integer samples allowing signal to be coded in the intervall $[2^{15} - 1; -2^{15}]$ while the DVD uses 24-bit ones: range=$[2^{23} - 1; -2^{23}]$

[2] DirectX, AudioUnits and TDM supports fixed-point samples because their plugins may deal directly with files, cd-rom, dvd-rom, soundcards or fixed-point DSP while other standards only deals with hosts for whom the floating point representation is more convenient.

[3] It allows feedback and recursion between objects and is very useful for 'physical modeling'

[4] typically values like 44,1kHz for the CD, 48kHz for DAT or video and 96kHz for the DVD are used.

[5] It can be used for synchronisation purpose by the host scheduler when many complex audio paths are present.

3.2 Signal processing

Here we come to most important part of a plugin, almost every plugin standards have the same method called `process()` to do the audio processing, only the way it is called and the arguments are different between standards. This method is either called directly by the host or by the next plugin in the chain in the case of graph-oriented hosts. Input and output buffers are, most of the time, provided by the host and can either be the same or different to allow in-place or buffer-to-buffer processing but the plugin can't assume one or another and should work correctly in both cases or specify if it suports it (LADSPA).

As a major difference with hardware Digital Signal Processors, this method is assumed to be non interruptible. Therefore parameters interpolation (if needed) or other sample-accurate processing has to be done inside the process and can't be done automatically since processing audio by buffers prevent from audio-rate control as soon as the buffer-size exceeds 1 sample.

3.3 General considerations about Digital Signal Processing

This section doesn't plan to explain rules of 'audio processing algorithm optimization'. We just want to explicitely note that since audio-rates are measured in tenth of kHz and that plugins may handle several channels, the number of samples to process can become huge, thus one should pay special attention to the DSP algorithm complexity[6].

Without going into assembler coding, one should avoid memory allocations, conditional expressions inside loops[7], too many non-static-inline-function calls or intensive float–int conversion among other general programming tips. Note that in a lot of case down-sampling during the analysis step (e.g. in envelope detection) can save precious CPU time for other purpose.

[6] In particular algorithms whose complexity is exponential or polynomial with the number of sample to process, shouldn't be used in a real-time context.

[7] It can break pipe-lining optimizations

Chapter 4

Control

The control interface allow the user to act on parameters, to tune the behaviour of the audio processing function. This can be done in real-time, so that the user can directly hear the consequence of his changes on the audio signal transform. There are different kinds of parameter, that require different handling. We can consider three main categories of parameters:

- **Continuous parameters:** These parameters can be characterized by the fact that they do not introduce a discontinuity in the audio stream [1], when they change of a small amount. As a consequence, these parameters can be interpolated. These parameters are obviously represented by numerical values and can be considered 'passive', in the sense that they are just used as a variable value in a generic computation. As examples: a filter's cutoff-frequency, a volume gain, a modulation frequency or a clipping threshold are continuous.

- **Events:** These parameters, sometime referred to as **messages** or **commands**(EyesWeb), are characterized by their discrete nature. These messages can be of any type, but should belong to a set of predefined messages, known from the plugin. They are not directly used in computation, but are rather interpreted to trigger a specific action, without breaking the audio stream process. As examples: the choice between 'sawtooth', 'sine', and 'square' for a modulator signal, a MIDI note-on event or the number of echoes in a delay.

- **Setup parameters:** These parameters affects the plugin's configuration. They are non-realtime because they involve computationally expensive operations, like memory allocation, that definitely break the audio signal stream. As exemples: the choice of a impulse response file for a convolution plugin or a fft-size in a frequency domain transform.

[1] A dicontinuity in the audio stream is characterized by audible 'zipper noise' or 'clicks'

This distinction between these three categories is not always that clear and well defined, but these three kinds of parameter correspond to different handlings.

With the increase of complexity of the plugins, which require more and more internal parameters to get finer acoustic results[2], there is an ongoing need of mapping and conversion routines, so that the user can still control them in an ergonomic manner. The parameters available to the user and the host (**external parameters**) can differ from the parameters used in the process computation (**internal parameters**). There may exist a conversion layer between them.This conversion can be of various kind, such as scaling a gain expressed in dB to a linear scale, mapping of frequency, gain and resonance to filter coefficients or clipping of the parameter to its range.

4.1 External parameter declaration

4.1.1 Mandatory declarations

The external parameters should be declared, to allow the host and/or the user to modify them. However, the host and the user do not need the same information to handle them. The minimum required set of parameters properties that the host should know contains, at least, the two following:

- **The parameter ID**: This ID is used as a selector among the parameter structure. Parameters ID are signed or unsigned long integers in all major audio plugin standards, sometimes of an enumerated type (DXi, VST, RTAS).
- **The type**: LADSPA and VST make use of 32-bit float only for all control data, and VST restrict the range to [0,1]. Other standards like DXi and AudioUnits implement all parameters as 32-bit float for efficiency, but keep a field in a parameter info structure, that defines whether to interpret the value as an integer, floating-point value, boolean, or enumeration (integer series). We find a similar mechanism in EyesWeb, but with three base format: double, int, and character string, plus a 'Parameter type' flag.

4.1.2 Optional declarations

Most plugin standards do offer much more information about parameters, to prevent the host and the user from manipulating them wrongly, and to let the user

[2] e.g. in physical-modelling of instruments, reverb rooms...

access them in a friendlier way. Such information may be stored in a 'ParameterInfo' structure (DXi, AU), or suggested as defined flags or 'hints' (LADSPA).

- **The name or label** is interesting for the user who do not speak like machines... The parameter name can be part of a parameter information structure, like in DXi, EyesWeb and AU. LADSPA stores the port names (audio AND control) in an array. Parameter's name is not mandatory in VST, but a 'GetParameterName' method exists in the API, that the developper can implement.

- **The units** gives more sense to the values. Few standards offer this feature: DXi,and EyesWeb stores the unit of each parameter in the ParamInfo structure. In AU, the units is part of the type: for example the type 'angle' is expressed in degrees or radians, the type 'frequency' in Hertz... and the parameter's mapping is chosen with respect to this.

- **The range**: It gives hints about what values this parameter is assumed to take. This is useful to prevent the plugin from doing error-leading computations, like log(0), negative frequency for a filter ... etc. It also helps the user to grasp the meaning of the parameter he's handling. Almost all standards allow to specify the range of the parameters, except VST for which the range is normalized between [0;1] and conversion has to be done by hand.

- **A default value**: It can initialize the parameter to a relevant value, within the parameter range, at instantiation time. This feature is implemented in LADSPA, DXi, EyesWeb AU, MAS. In VST, it is stored in the default preset.

- **The mapping**: It helps manipulating the parameter in a more ergonomic way. For example, frequency and gain are often more convenient when handled with logarithmic mapping. Mapping may be linear (all standards), logarithmic (LADSPA, AU), boolean (LADSPA, DXi, AU), indexed (LADSPA, DXi, AU)... VST provides conversion tools for GUI-display only.

- **Automation**[3]: Automatable parameters have to be declared as such by the apropriate mean:

  - AudioUnits: Flag in the parameter information structure.
  - DXi: Index of automated parameters should be less than `NUM_AUTOMATED_PARAMS` within the parameters enumeration.
  - MAS: Automated parameters should be public.

[3]See 4.2.2 for explanation.

	Name/Label	Units	Range	Default	Mapping
VST	Not mandatory, but method in the API	—	[0,1] fixed	—	conversion tools for display only
AU	In a ParamInfo structure	In a ParamInfo structure	Min, max stored in ParamInfo struct	In a ParamInfo structure	Many many!
LADSPA	In the PortNames array	—	min and max suggested as hints	suggested as hint, relative	lin,log,int, bool,SR-multiple
MAS	???	???	???	???	No
DXi	In a ParamInfo structure	In a ParamInfo structure	Min, max stored in ParamInfo struct	In a ParamInfo structure	—
EyesWeb	In a ParamInfo structure	In a ParamInfo structure	Flags HAS_MINMAX and values	Yes	—
RTAS	???	???	???	???	???

Table 4.1: Available parameter information

  - VST: During run-time, one should use the specific method `setParameterAutomated`.

Many other meta-information can be found in some standards:
EyesWeb specifies with the help of flags, whether the parameter can be changed at design-time, at runtime (and if so, if it can be exported), and if it is initially disabled.
AudioUnits provides a really rich API, with many predefined types of parameters, which have their own units, mapping, and range. This include indexed, boolean, percent, second, phase, cent, decibel, hertz, pan, a general type which is float between 0.0 and 1.0, and others. The developper can also add his own types.

4.2 Parameter update

The parameter update mechanism should efficiently take in account the fact that the parameter update rate[4] is asynchronous in nature, in general significantly lower than the audio stream, and that many changes could occur at the same time. Some parameters are known in advance (e.g. like MIDI events in a sequencer track), and some are not (e.g. live actualisation through hardware knobs).
The parameter update can be represented as a two steps operation: external parameters which have changed must be sent to the plugin; then, a conversion – if necessary – must make the corresponding internal parameters available to the process in a correct format. The figure 4.1 illustrates this mechanism.

4.2.1 Update mechanism

The most simple way to update parameters is the one implemented in LADSPA, and consists in a **polling mechanism**. At connection time, the host gives the

[4]There is no 'rate' in the strict sense, but we used this word for convenience.

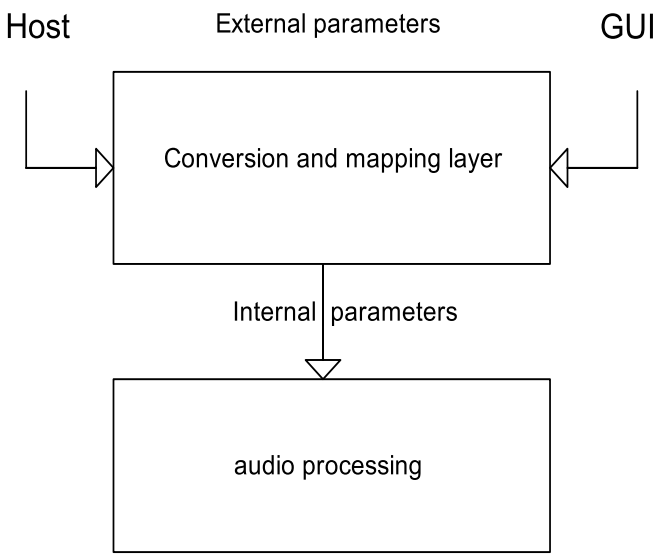

Figure 4.1: Internal and external parameters

plugin the adresses where it will write the external parameter values. The plugin can retrieve these values during the process function – and only then –, assuming they may have changed. Everything that should be performed for mapping and conversion of these external parameters can only be done then.

In the second approach (VST, jMax), the host or the GUI calls a specific plugin method, usually called `setParameter`. This method can only be called between two buffer processings. Necessary conversions and mappings should be implemented there by the plugin developer, so that internal parameters are up-to-date when the process function is called for the next buffer processing.

As a third case, some standards (DXi, AU) provide a conversion layer in the API between the plugin and the host or the GUI. Conversion and mapping are automatically done with the help of the parameter information structure. However, it is still necessary to retrieve the value of the parameters in the process function, and additionnal conversion have to be done here (e.g. computing filter coefficients from the cutoff frequency and the resonance, or checking mutual consistency of parameters).

Ways of handling events differ among plugin standards. A common and simple solution is to treat this set of value just like other parameters: reading and writing the value of the message with the `GetParameter` and `SetParameter` methods, and deciding on what should be done with a '`case`' or '`if`' condition. It can also be

directly treated in the `SetParameter` method (EyesWeb).

4.2.2 Automation

The automation allow the user to record changes of parameters along a sequenced timeline, and play them back. These changes can be either recorded inline (i.e. in realtime, during playback) or offline (e.g. by editing graphically a curve on a sequenced track). The automation recording consists in taking the parameter's value every timeslice.
Most standards tends to allow this (DXi, AU, MAS, RTAS, VST) since it is a quite powerful tool. The parameters declared as automated parameters should be 'realtime-able', i.e. they should not introduce too heavy computations or memory allocation.

4.2.3 Parameters encapsulation and meta data

Some header-information might be provided to the plugin during runtime, in addition to the new parameter's value. Precisely, the time at which change occur, and the way the value should be modified can be specified.

Type of data

As it has been mentionned in the 'Parameter declaration' section, some standard make use of a unique -or a restrained set of- data type for transport. In this case, a flag attached to the parameter in a parameter-info structure, specifies the type, into which the value should be casted.

Timestamps

The user who controls the interface can act at any time, disregarding what the plugin is doing. Thus, more than one change can occur during the time interval a timeslice is being processed. When considering punctual events such as audio attacks, synchronicity can play a major role in the audio rendition, and need more precision than the timeslice, which can last more than 50 ms.
Hence, timestamps can be attached to the parameter changes, to specify the precise sample accurate time at which they should occur. These events can be queued, their timestamp converted to the sample position within a buffer timeslice, and then be performed with the right time distribution during the next buffer processing.
In general, timestamping of parameters is limited to midi messages. Only EyesWeb make use of it for other control parameters.

Interpolation

On the other hand, for most parameters considered previously as 'continuous', such a precision does not matter, since the audio timeslices are really small, but the continuous nature of the parameters should be respected to avoid 'zipper noise' in the audio output signal. Therefore, various standards have implemented a way of interpolating the parameter' values to smooth the change, and avoid gaps in the audio stream that would generates these 'clicks'. A common way of doing is to take in account the last value of the given parameter, and perform an interpolation between the new value and the previous one. Different kind of interpolation may be available to perform the interpolation such as linear (DXi, EyesWeb, AU) , quadratic (DXi) or sine (Dxi).

4.3 Parameter persistence and presets

Persistence of parameters is a saving of the parameters values, that enable the user to find these same values – and not the default ones – when the user re-open the plugin. It means that when the user close the host and relaunch it for a new session, the persistent parameters adjustments will be the same as when he quitted. This feature is useful when manipulating plugins with lots of control values, like an equalizer for example.
Presets are actually an extension of the persistence system to the user's will: the user can decide to store different sets of parameters that he likes, usually identifying them by a name. The way to store the preset depends on the presets size: if the preset is small, it can be stored by the host (VST) or in a register (DXi); whereas if the preset is bigger (e.g. it contains a waveform, or a picture), it will be stored in a separate file as a bytestream [5].

4.4 MIDI

MIDI (Musical Instrument Digital Interface) is a protocol that transmits information about how music is produced. It is asynchronous but quantized at a maximum rate of 31,25 kbit/s[6], and encode control values that fit in both *event* (e.g. 'note on/off') and *continuous parameters* (e.g. 'pitch bend') categories. As a major difference with GUI interaction, MIDI messages usually use time-stamps, allowing sample accurate rendering. One can distinguish different families among

[5]The path of this file will be saved the same way a small preset would have been saved.

[6]This rate was the standard for communication with hardware devices. However, hosts generally overcome this limitation internally: an adaptative resolution of 480 'ticks' per quarter-note is usual.

musical plugins using MIDI: synthesizers, MIDI-controlled effects, pure MIDI effects[7] and analysis plugins – though those last ones aren't formalized in any standard for now – depending on the kind of input and output data. In this document, we will only deal with synthesizers and MIDI-controlled effects.

4.4.1 Synthesizers

Synthesizers (or 'instruments') appeared in the second generation of plugin standards. As harware ones, they get MIDI messages as input, usually have no audio input, and output audio data on many pins compared to chained effects. They often have a lot more parameters, and many of these ones are mapped to MIDI continuous controllers for a convenient live interaction.

4.4.2 MIDI-controlled effects

Integration of synthetizers into standards has allowed the interaction with plugins via MIDI. It allows to 'play' an effect as an intrument, using the incoming audio as a primary material to generate sound. They are often intented for musicians more than sound-engineers. As examples we can cite filters whose cut-off frequency is tuned to MIDI notes, or live-samplers using the incoming audio as a wave-table.

[7]MIDI effects consist in taking MIDI messages as data input and process them. No audio data is implied in the process. Most common MIDI effects are arpegiators, delays and echoes...

Chapter 5

Host environment integration

In order to work together, host and plugin have to know about each other both static and dynamic information. Some are fundamental other are optionnal, the options can vary a lot between standards. They can be either asked or told at any time, but construction-time – or opening-time – is more usual. We describe below the most important of them.

5.1 General information

5.1.1 Meta information

For most standards it is possible to provide a plugin name, a vendor name, a description and a category. One should also have a way to add version number to a plugin as well as to have a way to ask the host's name and version for compatibility purpose[1] It is much more convenient than providing dedicated versions of the same plugin for different versions and kind of host.

5.1.2 Audio setup

Audio pin properties

In this document, we assume that an audio input or output (pin) can be made of mono-channels grouped together that have to be treated as one entity. It makes sens when dealing with mono, stereo or surround channels. In a plugin graph or in a host context, a plugin can negotiate its connexions with other plugins or directly with the host. For that purpose, a plugin need to specify the total number of audio channels it supports and the way they have to be grouped if necessary. Character strings can often be provided to name the pins (see table 5.1). Often there is only one input and one output pin with the same propeties

[1] e.g. Plugins can declare their abilities depending on what hosts support and also depending on known bugs about one particular host.

(classical chained inserts), but in the case of instruments, multiple output pins are common (e.g. one by MIDI channels or one by drum sound in order to be compressed/equalize separately) and with spacializer, panner or down mixing plugins, input and output pin properties can be different (e.g. mono in - surround out). Moreover some channels can be tagged as side-chain.

	Type	Channels	Name	Sidechain	Interleaved	IO switch.
VST	32-bit float	any	yes	yes	no	in theory
AU	32-bit float	any		no	both	yes
LADSPA	32-bit float	any	yes	no	no	no
RTAS	32-bit float	any		yes	no	no
DIRECTX	WaveFormatEx	any	yes	no	Both	in theory
MAS	32-bit float	11x11 max	yes	yes	no	no
EYESWEB	32-bit float	any		no	no	no
MAX-like	32-bit foat	any	no	yes	no	yes

Table 5.1: Audio pins properties

Audio processing properties

A plugin can notify the environnement about its DSP properties. These properties include:

- The buffer processing type : 'in place' or 'buffer to buffer' as well as 'accumulation' for send effects or 'replacing' for inserts (LADSPA, DXi, EyesWeb)
- The generation of a tail : The tail is the part of the processed audio signal, added at the end of the stream processed[2]. (DXi[3], VST)
- The latency : The latency is the pure delay introduced by the computation[4]. (VST)
- The real-time quality : In some plugin GUI, one can choose a lower quality (typically obtained by down-sampling) for the audio signal preview, to get faster computation. (VST, RTAS)

[2]A tail will typically be generated in effects like reverbs, delays, time-streching ...

[3]DXi does this dynamically in the process function.

[4]e.g. to compute a FFT at 44100Hz with 512 samples of hopesize, the latency would be 512/44100 = 11,5 ms.

5.2 Runtime information

Audio plugin can often be used within a sequencer, where several tracks are arranged along a common timeline. During runtime, some plugin – MAS, VST, DX– can send or ask for time-information[5] about the position of the current timeslice being processed within this timeline. It is very useful to set parameters to musical meaningful values (e.g. a number of quarter note instead of a time in millisecond) or even sync some transformation patterns on the current position in a mesure.
It is even possible with some standard – MAS – to send control information to the host, related to the sequencer run, like 'start', 'stop', 'goto locator', 'rewind' ...thus enabling beat-tracking algorithms to synchronize a host on a recorded performance track.
One may also be prepared with standards like VST, DirectX, that the sample-rate or the buffer size may change during runtime. Non power of 2 or even size of buffer are possible, so be careful.

5.3 Acces to the plugin

5.3.1 Plugin Location

Although plugins are usually stored at a common place, there is no real standard location for plugins storage. Still, one will often find a recommended path for the plugins location, to make things cleaner. The possibility for having multiple possible paths actually depends on the host. For some standards, it can be modified, either by adding new locations in an environnement variable (LADSPA) or by changing the path, or in the system register (DirectX), or browsing for a plugin during runtime. However this depends more on the host's strategy and on the OS than on the standard.

5.3.2 Plugin ID

In order to be known by the host, a plugin should provide a unique way to be identified among other plugins: a '**unique ID**'. Depending on the scope in which the host is assumed to seek to find the plugins, the ID may be a simple value, a more complex registration key.
This ID can be made of 1 or many value(s), that is(are) assumed to be chosen different from other plugins' ID, **by the plugin developer**. In this case, it is usually a long integer, or a combination of long integers: this is the case is LADSPA, AU, VST, RTAS, MAS,... In the case of multiple IDs, one may find the manufacturer ID (AU, MAS, RTAS), a variation ID (MAS), a plugin-type

[5]Time information include tempo (bpm), time signature, temporal and/or musical position

ID (AU) ... Note that a plugin ID can be registered by the host manufacturer to become 'official' and avoid clashes with other third party developer's IDs.
It is noteworthy to notice that the shared library files in which the plugins are compiled might contains several plugins – this can be done in LADSPA, AU, MAS and VST (using a plugin shell[6]) –. In this case, every plugin inside this library should have a unique ID.
The other solution is specific to DirectX, and consists in **registering the plugin-object in the system register** through GUID's (Globally Unique ID), which are 128-bit registration keys **automatically generated by the system**. This solution has the advantage of avoiding identity clashes between plugins, because of developpers ignorance of all already existing plugins ID.The DirectShow filters used by Cakewalk and SonicFoundry as 'DirectX effects', and by EyesWeb for its modules use this mechanism.

5.3.3 Entry points

The plugin code stored within a *Shared Object* or *Dynamic Linked Librairy* file [7] that may contain one or several plugins. The host application accesses the plugin by calling the *Entry Point* function(s). The user then accesses the plugin thru the host's interface in a totally integrated way. The Entry Point function will instantiate the plugin and provide information to the host.

[6] from one entry visible in the host you can access all the plugin from a manufacturer
[7] These libraries are *.so files in Linux and *.dll in Windows

Chapter 6

Conclusion

As we have seen through this document, from an abstract point of view, plugins standards looks very similar and share a lot of functionnalities. Some are very quick and simple to develop, like LADSPA, leaving the hard work to the host and the user (GUI, mapping), while other like DirectX or RTAS, have a harder learning curve because they give (too?) many possibilities to the plugin manufacturers (Custom GUI, handling of control surfaces or onboard DSP, automation, MIDI...). Between we can find AudioUnits and VST trying to make ends meet through easy APIs to start with, but allowing still many possibilities for (quite) experienced programmers.
However when looking into details, we can raise many differences. These differences are mostly dependent on the context in which the plugin standards were born. For historical, technical and commercial reasons, host manufacturers started on specific plateforms, targeting different kind of users and/or activities among audio-engineers, musicians, professionals, amateurs, post-production, multimedia, video .... These initial constraints still remains present in standards because of backward-compatibility, despite the fact that since those early times companies and host have evolved, changed of plateform or even of customer's target.
In this specific context, the GMPI[1] group was formed, at the end of 2002, on the initiative of Ron Kupper from Cakewalk and under the authority of the MMA[2], to design a new open and cross-plateform plugin standard, that would meet everyone needs, and put in common all the experience accumulated by people since the first host and plugin systems. However there is still a long way before we can use and develop for a unique kind of plugin that would be easy and quick to develop, highly scalable and efficient. In the meantime, developping for different standards and plateforms is still necessary and time-consuming. Furthermore the choice of the supported standards is fundamental to determine the audience that will be targeted.

[1] Generalized Musical Plugin Interface
[2] MIDI Manufacturer Association

Appendix A

Ressources

Most of the ressources we used to write this study are available on the Internet. Here are the links to the web sites and SDK's of the principal standards:

- AudioUnits Site: `http://developer.apple.com/audio/`
- DirectX – DXi Site: `http://www.thedirectxfiles.com/`
- EyesWeb Site: `http://infomus.dist.unige.it/eywindex.html`
- LADSPA Site: `http://www.ladspa.org/`
- jMax Site: `http://www.ircam.fr/jmax/`
- MAS Site: `http://www.motu.com/`
- Max/MSP Site: `http://www.cycling74.com/`
- PureData Site: `http://www.pure-data.org/`
- RTAS Site: `http://www.digidesign.com/`
- VST Site: `http://ygrabit.steinberg.de/`