TWO CROSS-PLATFORM CSOUND-BASED PLUGIN GENERATORS

Victor Lazzarini
An Grúpa Theicneolaíocht Fuaimé agus Ceoil Dhigitigh
National University of Ireland, Maynooth, Ireland
Victor.Lazzarini@nuim.ie,

Rory Walsh, Martin Brogan
Music Department
Dundalk Institute of Technology
rory.walsh@dkit.ie
broganmartin@yahoo.co.uk

ABSTRACT
This article describes two new plugin generators for the development of effects and instruments using the Csound audio programming language, csLADSPA and csVST. It discusses the basics of these two systems, exploring some of their internal aspects. The text continues on to show three basic examples, two plugin effects and one synthesis instrument. These demonstrate both time-domain and spectral processes. The article concludes with a look at specific details of operation of the two plugin generators.

1. INTRODUCTION
The availability of a variety of plugin models and host applications has provided fertile ground for the development of audio effects. Such systems allow programmers to concentrate on the signal processing tasks at hand, as they provide all the infrastructure for audio input and output, as well as a model for the creation of software components. In this article, we will go a step further by providing the plugin developer with a full complement of signal processing operators, unit generators, from a widely adopted system, Csound[1]. In this scenario, a plugin designer can avail of already existing components to build his/her own effects or instruments.

The generators presented here support the creation of plugins under two major systems: a Free-software one, the Linux Audio Developer’s Simple Plugin API (Application Programming Interface) (LADSPA)[7] and a commercial system, the Virtual Studio Technology (VST) [8]. Plugins for both systems can be created using the same code, interchangeably, as they are solely based on the Csound language.

2. THE PLUGIN GENERATORS

2.1. The inside bits
csLADSPA and csVST are not themselves the plugins, but plugin generators. They will create, in a host, any number of plugins written by the user. Each one of these will be linked to a particular Csound orchestra/score. Plugins are created when csLADSPA (or csVST) is loaded by the host. In the case of csLADSPA, the generator will then provide one plugin for each Csound code file found in a pre-defined directory. For csVST plugins, the Csound code will be appended to the dynamic-loadable module (MS-Windows) or included in a bundle (OS X).

As mentioned above, the number of audio channels for a plugin and its controls is determined by the Csound code. A number of extra XML tags have been defined for some of these tasks. Once plugins are created, they will generally be displayed on a list by the host, from where they can be instantiated and run.

A basic model of how the plugins work is shown in below (fig.1). The host application loads the csLADSPA or csVST-created plugin. This will instantiate a Csound object and compile its associated orchestra/score code.
When the user processes audio through it, the plugin will route the input audio to that instance of Csound. This will then process this audio and return it to the plugin which will then route that audio to the host application. Figure 2 shows a plugin in operation in the host application Audacity.

For this to happen, calls are made to the Csound::PerformKsmps() method, which is responsible for processing or generating one vector full of samples. These, in turn, are shifted in and out of the main Csound buffers to/from the host buffers. In the case of synthesizer instrument plugins (csVSTi), MIDI communication between the host and Csound is also provided through API calls. In this case, Csound instruments will be instantiated by realtime MIDI events sent in by the host.

### 2.2. Plugin tags

Plugins will use the unified file format (CSD) for reading Csound code. In these files, the user must specify some basic information about the plugin, in the form of XML tags (table 1). This is done by adding a section in the CSD file whereby the user can specify things like the name of the plugin, the author, etc. It is in this section that the user can also specify the control ports they will need in order to interact with their Csound code when running the plugin through a host. This section will be defined inside `<csLADSPA>`, `<csVSTfx>` or `<csVSTi>` for LADSPA, VST effects and VST instruments, respectively.

<table>
<thead>
<tr>
<th>Tags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the plugin as it will appear in the host application</td>
</tr>
<tr>
<td>Maker</td>
<td>Author of plugin</td>
</tr>
<tr>
<td>UniqueID</td>
<td>ID given to plugin, each plugin should use a unique ID.</td>
</tr>
<tr>
<td>Copyright</td>
<td>Copyright/Licence notice</td>
</tr>
<tr>
<td>ControlPort</td>
<td>The name of the control as it appears when the plugin is ran and the name of the channel which Csound will retrieve the data on. The two names should be separated by a '</td>
</tr>
<tr>
<td>Range</td>
<td>Plugin max/min range. Again the two values are separated by a '</td>
</tr>
</tbody>
</table>

Table 1. Plugin tags

ControlPort and Range tags are optional, but the former always requires the latter. Examples of how these tags are used can be seen in the next section.

### 3. EXAMPLES

In the following section two csLADSPA plugins are presented. The first illustrates the mechanisms for communication between the host and the csLADSPA plugin. The other plugin makes use of some of the more advanced opcodes included with Csound5, i.e., the PVS[2] opcodes.

#### 3.1. An Adaptive FM effect

The first example features an effect based on Adaptive Frequency (AdFM) [3]. This technique uses an input audio signal as a carrier in a frequency modulation synthesis algorithm. The signal is put through a variable delayline and modulated with a sinusoidal oscillator. The AdFM effect is implemented by a User-Defined Opcode. This plugin will require two control ports, one for the index of modulation and another for the carrier:modulator frequency ratio:

```
<csLADSPA>
Name=AdFM
Maker=Lazzarini, Walsh & Brogan
UniqueID=1054
Copyright=None
ControlPort=Modulation Index|index
Range=0|5
ControlPort=Car-mod ratio|ratio
Range=0.5|2
</csLADSPA>

<CsoundSynthesizer>
<CsInstruments>
opcode DFM,a,akki
setksmps 1
as,krt,knx,ifn xin
kcps,kamp ptrack as,1024
adt oscili knx/($M_PI*kcps),kcps/krt,ifn
adp  delayr 1 /* delay line */
```

In this example, the `setksmps` opcode sets the number of samples per second (ksmps) to 1, which is used by the `opc` opcode to determine the rate of the modulation. The `delayr` opcode is used to delay the signal by 1 sample.
As previously mentioned the means of communication between the plugin and the instance of Csound is provided by the software bus. In Csound we can use the chnget opcode to retrieve data from a particular bus channel. In the case above this data is used to control the different parameters as defined by the Range tag in the <csLADSPA> section of the above code.

### 3.2. A spectral manipulation plugin

Csound5 comes with a host of new Phase Vocoder Streaming, PVS, opcodes. These opcodes provide users with a means of manipulating spectral components of a signal in realtime. This plugin averages the amp/freq time functions of each analysis channel for a specified time.

```plaintext
<csVSTfx>
Name=PVSBlur
Maker=Lazzarini, Walsh & Brogan
UniqueID=5961
Copyright=None
ControlPort=Max Delay|del
Range=0,1
ControlPort=Blur Time|blur
Range=0,10 &log
</csVSTfx>
<CsInstruments>
ksmps=32
nchnls=2

opcode PVSBl,aaaa,kkkki
ka, kc, km, kndx, ifn, xin
a1 oscili kndx/(2*$M_PI),km,ifn
a2 tablei a1,ifn,1,0.25,1
a3 tablei a1,ifn,1,0,1
aae, abe hilbert a2
aao, abo hilbert a3
ac oscili ka/2,kc,ifn
ad oscili ka/2,kc,ifn,0.25
aee = aae*ac + abe*ad
sou = aao*ac + abo*ad
ael = aae*ac - abe*ad
sol = aao*ac - abo*ad
xout aee,sou,ael,sol
endop
</CsInstruments>
</CsScore>
</CsoundSynthesizer>
```

### 3.3. A Split-Sideband synthesis instrument

The following code will create a synthesiser, using the Split-Sideband algorithm[4], that accepts MIDI messages sent from the host application. MIDI is routed from the host to the plugin through the use of the –midi-key-cps flag which is passed to the CsOptions section of the csd file. This sends all note-on messages MIDI byte 2 to p-field 4, converting them to Hz, where they can easily be retrieved in the instrument code. As standard envelopes do not work well with indefinited duration events, we will use the extended ‘r’ family of envelopes. These can sense a turnover event (such as a MIDI noteoff) and extend the performance time of the current instrument. In the case of linenr, an exponential decay is applied to the endpoint of the envelope (its ‘release”).

```plaintext
<csVSTi>
Name=SpSB Synth
Maker=Lazzarini, Walsh & Brogan
UniqueID=5961
Copyright=None
ControlPort=Car-Mod Ratio|kratio
Range=0.5,2
ControlPort=Mod-Index|kindx
Range=0,20
</csVSTi>
<CsOptions>
-rtmidi=null -M0 -m0d --midi-key-cps=5 \
--midi-velocity-amp=4
</CsOptions>
<CsInstruments>
ksmps=32
nchnls=2

opcode SpSB,aaaa,kkkki
ka, kc, km, kndx, ifn, xin
a1 oscili kndx/(2*$M_PI),km,ifn
a2 tablei a1,ifn,1,0.25,1
a3 tablei a1,ifn,1,0,1
aae, abe hilbert a2
aao, abo hilbert a3
ac oscili ka/2,kc,ifn
ad oscili ka/2,kc,ifn,0.25
aee = aae*ac + abe*ad
sou = aao*ac + abo*ad
ael = aae*ac - abe*ad
sol = aao*ac - abo*ad
xout aee,sou,ael,sol
endop
</CsInstruments>
</CsScore>
</CsoundSynthesizer>
```
4. USING THE PLUGIN GENERATORS

The specific details on creating the above plugins in various host applications depend on the type of plugin desired. csLADSPA plugins are by far the easiest to generate. In order to use the code examples above in a LADSPA plugin the end-user must place the csLADSPA dynamic module and the .csd file in the folder pointed to by the LADSPA_PATH environment variable. The csLADSPA library will automatically detect all Csound files and load them as separate plugins.

The mode of operation of csVST will depend on the target platform. On Microsoft Windows, we must make a copy of the generic csVST DLL and then append an appropriate .csd file to the end of the binary data. This method is similar to the one used in the Cabbage[6] application builder to generate Csound-based freestanding programs. A simple C++ utility program, included with the csVST source code, demonstrates how this works. An alternative is to use Rory Walsh’s Lettuce Csound frontend[9], which provides an option to ‘export as VST’. The program will scan the .csd text, check to see if it needs to create an effect or an instrument and then creates a new plugin accordingly.

On OSX users must create a new plugin bundle and place their .csd file inside it. csVST then looks through the current bundle and loads the first .csd it finds. Bundles may not contain multiple .csd files may not run as expected. Included in the csVST source is a simple Python language[2] script which automates this process.

5. CONCLUSION

The current versions of csLADSPA and csVST perform adequately and has been tested by students in the National University of Ireland Maynooth and at Dundalk Institute of Technology, Ireland. The system has been used both as a creative tool and as a pedagogical utility used in the teaching of DSP techniques. csLADSPA, in particular, has been tested using a number of hosts and has demonstrated to be a great companion to Paul Davis’ Ardour. csLADSPA is Free Software and is included in the standard Csound 5 distribution (http://csound.sf.net).

The csVST plugin generator kit will be soon available as a binary distribution for the Windows and OSX platforms and as source code for all Csound 5 support platforms.

6. REFERENCES