

# Developing LADSPA Plugins with Csound

**Victor Lazzarini**

National University of Ireland, Maynooth  
Maynooth  
Co. Kildare  
Ireland  
victor.lazzarini@nuim.ie

**Rory Walsh**

Dundalk Institute of Technology  
Dundalk  
Co.Louth  
Ireland  
rory.walsh@ear.ie

## Abstract

Csound is one of the most powerful audio programming languages available to electroacoustic composers today. Its origins can be traced directly back to Max Matthews in Bell Labs and since its inception it has grown to become one of the most extensive computer music toolkits in development. This paper will describe a new toolkit for the development of LADSPA plugins using the Csound audio programming language. The toolkit itself was developed using the new Csound API and the Linux Audio Developers Simple Plugin API. This text will explore the implementation of said toolkit and conclude with examples of the toolkit in use.

## Keywords

Computer Music, Audio Plugins, Musical Signal Processing

## 1 Introduction

csLADSPA is a new toolkit for the development of LADSPA[1] plugins using the Csound audio programming language. csLADSPA provides musicians with no low-level programming experience with a simple albeit powerful toolkit for the development of audio plugins. The main goal of this project keeps in line with one of the main objectives of the LADSPA project i.e., to create a 'simple' architecture for the development of plugins. It was of the utmost importance to the author that the end-user need only a rudimentary knowledge of Csound in order to get started with csLADSPA. It is expected that a novice user will in time explore more of Csound to create complex plugins.

### 1.1 The Csound host API

An API (application programming interface) is an interface provided by a computer system,

library or application, which provides users with a way of accessing functions and routines particular to the control program. Essentially APIs provide developers with a means of harnessing an existing applications functionality within a host application.

The Csound API[2] can be used to start any number of Csound instances through a series of different calling functions. The API also provides mechanisms for two way communication with an instance of Csound through the use of a 'named software bus'. In short, the Csound API makes it possible to harness all the power of Csound in ones own application.

### 1.2 LADSPA Plugins

The LADSPA framework came to fruition in early 2000 following the combined efforts of Paul Davis, Richard W.E. Furse, and Stefan Westerfield[3]. LADSPA provides developers with a simple way of developing audio plugins which can be loaded by a wide range of host applications. In terms of implementation, all LADSPA plugins must:

- Declare a plugin structure (for audio buffers, etc.).
- Instantiate the plug-in by calling a user defined function which returns a LADSPA\_Handle data type.
- Register the plugin using a user-defined 'ladspa\_descriptor()' function.
- Connect ports to data locations using a user-defined connect function.
- Process blocks of samples in a run function.
- Free memory

## 2 Inside csLADSPA

The csLADSPA library was written in C++[4]. It is programmed using the same basic structure as any LADSPA plugin. Calls are made to the Csound API at different stages in the execution of the plugin. The Plugin declares a data structure, which includes a pointer to an instance of the Csound class, an array to hold control values and an array to hold the names of the software bus channels which will be used for parameter control. The steps taken in the actual operation of the plugin are as follows:

- When the plugin is loaded, all Csound (.csd) files which reside in the plugin folder are parsed. This data is assigned to the various members of the LADSPA descriptor structure. This step generates a series of plugins based on the Csound source code files created by user.
- When a plugin is instantiated, csLADSPA creates a Csound instance and compiles the respective Csound source code.
- This is followed by the connection of ports to data locations and the assignment of control values to an array of floats. This can be assessed from inside the run function, which is called by the host application.
- When the host runs the plugin, blocks of samples are processed in a 'run' function. This accesses the Csound instance low-level IO buffers, through calls to Csound::Spin() and Csound::Spout(), in order to route the selected audio to it. Finally, inside a processing loop a call is made to Csound::PerformKsmmps(), which does the actual signal processing.

A basic model of how the plugins work is shown in below (fig.1). The host application loads the csLADSPA plugin. When the user hits the process button the csLADSPA library will route the selected audio to an instance of Csound. Csound will then process this audio and return it to the csLADSPA plugin which will then pass that audio to the host application.

### 2.1 Getting started

In order to get started writing csLADSPA plugins the end-user must place the csLADSPA library and all csd plugin files in the folder pointed to by the LADSPA\_PATH environment variable.

LADSPA\_PATH must be set in order for csLADSPA library to work. The csLADSPA library will automatically detect all Csound files and load them as separate plugins. In order to keep things simple, csLADSPA only works with the unified Csound file format.

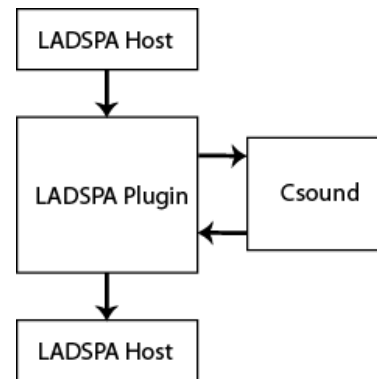


Figure 1. The csLADSPA model

### 2.2 csLADSPA tags

In order for csLADSPA to load the Csound files in the LADSPA\_PATH the user must specify some basic information about the plugin. This is done by adding a section at the top of the Csound 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. Every csLADSPA plugin must specify the following:

Tags	Description
Name	The name of the plugin as it will appear in the host application
Maker	Author of plugin
UniqueID	ID given to plugin, each plugin should use a unique ID.
Copyright	Copyright/Licence notice

If users wish to add controls to their plugins they can do so using the following tags:

Tags	Description
ControlPort	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 ' ' symbol.
Range	Plugin max/min range. Again the two values are separated by a ' ' symbol. If users wish to controls to respond logarithmically they can add a '&log' after they specify the range values.

Note that if a user uses the `ControlPort` tag they must *always* place a `Range` tag underneath it. Examples of how these tags are used can be seen in the next section.

### 3 Examples

In the following section three `csLADSPA` plugins will be presented. The first two plugins will illustrate the mechanisms for communication between the host and the `csLADSPA` plugin. The third and final plugin will illustrate a more complex process which makes use of some of the more advanced opcodes included with `Csound5`, i.e., the `PVS[5]` opcodes.

#### 3.1 A basic gain plugin

Given that most plugin SDKs come with a simple 'gain' example we'll start here too. Here is the full code to a simple gain example.

```
<csLADSPA>
Name=Gain Plugin
Maker=John Doe
UniqueID=1049
Copyright=None
ControlPort=Gain|gain
Range=0|2
</csLADSPA>
<CsoundSynthesizer>
<CsInstruments>
sr = 44100
ksmps = 10
nchnls = 1

instr 1
  kGain chnget "gain"
  ain in
  out ain*kGain
endin

</CsInstruments>
<CsScore>
i1 0 3600
</CsScore>
```

```
</CsoundSynthesizer>
```

As previously mentioned the means of communication between the plugin and the instance of `Csound` is provided by the named software bus, in this case the name given to the software channel is 'gain'. In `Csound` we can use the `chnget` opcode to retrieve data from a particular software bus. In the case above this data is used to multiply the output signal by a value between 0 and 2, as defined by the `Range` tag in the `<csLADSPA>` section of the above code.

#### 3.2 A simple flanging plugin

Flanging is a commonly used digital audio effect. It's created by mixing a signal with a time-varying delayed version of itself. In `Csound` this can be done by using the `vdelay` opcode. To control the amount of delay one can use a low frequency oscillator or LFO. This plugin will need two control ports, one for the flange depth and one for the flange rate. Here is the full code for a simple 'flanger' plugin.

```
<csLADSPA>
Name=Flanger
Maker=John Doe
UniqueID=1054
Copyright=None
ControlPort=Flange Depth|depth
Range=0|1
ControlPort=Flange Rate|rate
Range=0|10
</csLADSPA>
<CsoundSynthesizer>
<CsInstruments>
sr = 44100
ksmps = 10
nchnls = 1

instr 1
  kdelttime chnget "depth"
  krate chnget "rate"
  ain in
  al oscil kdelttime, 1, 1
  ar vdelay3 ain, al+kdelttime, 1000
  out ar+ain
endin

</CsInstruments>
<CsScore>
f1 0 1024 10 1
i1 0 3600
</CsScore>
</CsoundSynthesizer>
```

#### 3.3 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. In the following example the opcodes `pvsanal`, `pvsblur` and `pvsynth` are used to manipulate

the spectrum of the selected audio. The plugin averages the amp/freq time functions of each analysis channel for a specified time.

```
<csLADSPA>
Name=PVSBlur
Maker=John Doe
UniqueID=1056
Copyright=None
ControlPort=Max Delay|del
Range=0|1
ControlPort=Blur Time|blur
Range=0|10 &log
</csLADSPA>
<CsoundSynthesizer>
<Csinstruments>
sr = 44100
ksmps = 10
nchnls = 1

instr 1
imaxdel chnget "del"
iblurtime chnget "blur"
asig in
fsig pvsanal asig, 1024, 256, 1024, 1
ftps pvsblur fsig, 0.2, 0.2
atps pvsynth ftps
out atps
endin

</Csinstruments>
<Cscore>
f1 0 1024 10 1
i1 0 3600
</Cscore>
</CsoundSynthesizer>
```

#### 4 Conclusion

The current version of csLADSPA performs 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. It has been fully tested in real-time using Jack-Rack[6] and in non-realtime mode using Audacity[7]. With regards to future developments the authors are currently working on a better error-checking system and multichannel support is also being investigated.

csLADSPA is Free Software, available for download from [www.eur.ie/csLADSPA.htm](http://www.eur.ie/csLADSPA.htm)

#### References

- [1] <http://www.ladspa.org>.
- [2] John ffitc. 2004. On The Design of Csound5. Proceedings of the 3rd Linux

Audio Developers Conference. ZKM, Karlsruhe, Germany.

- [3] Dave Phillips, Linux Audio Plug-Ins: A Look Into LADSPA:  
<http://www.linuxdevcenter.com/pub/a/linux/2001/02/02/ladspa.html>
- [4] Bjarne Stroustrup. 1991. The C++ Programming Language, second edition. Addison-Wesley, New York.
- [5] Victor Lazzarini, Joseph Timoney and Thomas Lysaght. 2006. Streaming Frequency-Domain DAFX in Csound 5. Proc. of the 9th Int. Conf. on Digital Audio Effects (DAFX) 2006, Montreal, Canada. pp.275-278.
- [6] <http://jack-rack.sourceforge.net/>
- [7] <http://audacity.sourceforge.net/>