Audio Plugin development with Cabbage

Rory Walsh
Dundalk Institute of Technology,
Co.Louth,
Ireland
rory.walsh@dkit.ie

Abstract

This paper describes a novel new approach to developing cross-platform audio plugins with Csound. It begins with an short historical overview of projects that led to the development of Cabbage as it exists today, and continues with a more detailed description of Cabbage and its use within digital audio workstations. The paper concludes with an example of an audio effect plugins and a simple MIDI based plugin instrument.

Keywords

Cabbage, WinXound, Csound, Audio Plugin, Audio programming languages.

1 Introduction

In an industry dominated by commercial and closed-source software, audio plugins represent a rare opportunity for developers to extend the functionality of their favourite digital audio workstations, regardless of licensing restrictions. Developers of plugins can concentrate solely on signal processing tasks rather than low-level audio and MIDI communication.

The latest version of Cabbage seeks to provide for the first time a truly cross-platform, multi-format Csound-based plugin solution. Cabbage allows users to generate plugins under three major frameworks: the Linux Native VST[1], Virtual Studio Technology (VST) [2], and Apple's Audio Units [3]. Plugins for the three systems can be created using the same code, interchangeably. Cabbage also provides a useful array of GUI widgets so that developers can create their own unique plugin interfaces.

When combined with the latest version of WinXound[4] computer musicians have a powerful, fully integrated IDE for audio software development using the Csound programming language.

1.1 The Csound host API

The main component of the framework presented here is the Csound 5 library[5], accessed through its API. This is used to start any number of Csound instances through a series of different calling functions. The API provides several mechanisms for two-way communication with an instance of Csound through the use of 'named software' buses.

Cabbage accesses the named software bus on the host side through a set of channel functions, notably `Csound::setChannel()` and `Csound::getChannel()`. Csound instruments can read and write data on a named bus using the `chnget/chnset` opcodes.

In general, the host API allows software to control Csound in a very flexible way, without it the system described in this paper would not have been possible.

2 Background

The ability to run open source audio software in tandem with commercial DAWs is not something new to computer musicians. Systems such as Pluggo[6], PdVST[7] and CsoundVST[8] all provide users with a way to develop audio plugins using open source audio languages. CsoundVST is still available for download but it's anything but a lightweight plugin system. Pluggo and PdVst have been discontinued or are no longer under development.

The software presented in this paper may well have been inspired by the systems mentioned above but is in fact an amalgamation of 3 projects that have been rewritten and redesigned in order to take full advantage of today's emerging plugin frameworks. Before looking at Cabbage in its present state it is worth taking a look at the two main projects it is derived from.

2.1 csLADSPA/csVST

csLADSPA[9] and csVST[10] are two lightweight audio plugin systems that make use of the Csound API. Both toolkits were developed so

that musicians and composers could harness the power of Csound within a host of different DAWs. The concept behind these toolkits is very simple and although each makes use of a different SDK, they were both implemented in the very same way. A basic model of how the plugins work is shown below in fig.1.

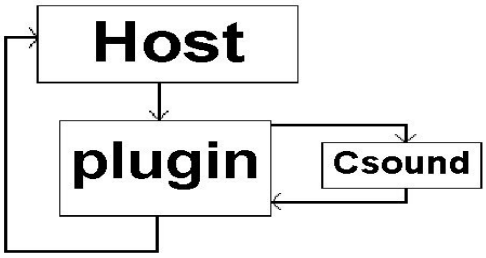

Figure 1. Architecture of a Csound plugin

The host application loads the csLADSPA or csVST plugin. When the user processes audio the plugin routes the selected audio to an instance of Csound. Csound will then process this audio and return it to the plugin which will then route that audio to the host application. The main drawback to these systems is that they do not provide any tools for developing user interfaces. Both csLADSPA and csVST use whatever native interface is provided by the host to display plugin parameters.

2.2 Cabbage 2008

Cabbage was first presented to the audio community at the Linux Audio Conference in 2008[11]. The framework provided Csound programmers with no low-level programming experience with a simple, albeit powerful toolkit for the development of standalone cross-platform audio software. The main goal of Cabbage at that time was to provide composers and musicians with a means of easily building and distributing high-end audio applications. Users could design their own graphical interfaces using an easy to read syntax that slots into a unified Csound text file(.csd). This version of Cabbage had no support for plugin development.

3 Cabbage 2011

The latest version of Cabbage consolidates the aforementioned projects into one user-friendly cross-platform interface for developing audio plugins. Combining the GUI capabilities of earlier versions of Cabbage with the lightweight csLADSPA and csVST systems, means users can now develop customised high-end audio plugins armed with nothing more than a rudimentary knowledge of Csound and basic programming.

Early versions of Cabbage were written using the wxWidgets C++ GUI library.[12] Whilst wxWidgets provides a more than adequate array of GUI controls and other useful classes it quickly became clear that creating plugins with wxWidgets was going to be more trouble than it was worth due to a series of threading issues.

After looking at several other well documented GUI toolkits a decision was made to use the JUCE Class library[13]. Not only does JUCE provide an extensive set of classes for developing GUIs, it also provides a relatively foolproof framework for developing audio plugins for a host of plugin formats. On top of that it provides a robust set of audio and MIDI input/output classes. By using these audio and MIDI IO classes Cabbage bypasses Csound's native IO devices completely. Therefore users no longer need to hack Csound command line flags each time they want to change audio or MIDI devices.

The architecture of Cabbage has also undergone some dramatic changes since 2008. Originally Cabbage produced standalone applications which embedded the instrument's .csd into a binary executable that could then be distributed as a single application. Today Cabbage is structured differently. Instead of creating a new standalone application for each instrument Cabbage is now a dedicated plugin system in itself.

3.1 The Cabbage native host

The Cabbage native host loads and performs Cabbage plugins from disk. The only difference between the Cabbage host and a regular host is that Cabbage can load .csd files directly as plugins. To load Cabbage plugins in other hosts users must first export the Cabbage patch as some form of shared library, dependant on the OS. The Cabbage host provides access to all the audio/MIDI devices available to the user and also allows changes to be made to the sampling rate and buffer sizes. The function of the Cabbage host is twofold. First it provides a standalone player for running GUI based Csound instruments. In this context it functions similarly to the Max/MSP runtime player[6]. Secondly it provides a platform for developing and testing audio plugins. Any instrument that runs in the Cabbage native host can be exported as a plugin.

3.1.1 Cabbage Syntax

The syntax used to create GUI controls is quite straightforward and should be provided within special xml-style tags `<Cabbage>` and `</Cabbage>` which can appear either above or below Csound's own `<CsoundSynthesizer>` tags. Each line of Cabbage specific code relates to one GUI control only. The attributes of each control is set using different identifiers such as `colour()`, `channel()`, `size()` etc. Cabbage code is case sensitive.

3.1 Cabbage widgets

Each and every Cabbage widget has 4 common parameters: position on screen(x, y) and size(width, height). Apart from position and size all other parameters are optional and if left out default values will be assigned. As x/y, width and height are so common there is a special identifier named `bounds(x, y, width, height)` which lets you pass the four values in one go. Below is a list of the different GUI widgets currently available in Cabbage. A quick reference table is available with the Cabbage documentation which illustrates which identifiers are supported by which controls.

```
form caption("title"), pos(x,y), size(width, height),
colour(“colour”)
```

Form creates the main plugin window. X, Y, Width and Height are all integer values. The default values for size are 400x600. Forms do not communicate with an instance of Csound. Only interactive widgets can communicate with an instance of Csound, therefore no channel identifier is needed. The colour identifier will set the background colour. Any HTML and CSS supported colour can be used.

```
slider chan(“chanName”), pos(x,y), size(width,
height), min(float), max(float), value(float),
caption(“caption”), colour(“colour”)
```

There are three types of slider available in Cabbage. A horizontal slider(`hslider`), a vertical slider(`vslider`) and a rotary slider(`rslider`). Sliders can be used to send data to Csound on the channel specified through the “chanName” string. The “chanName” string doubles up as the parameter name when running a Cabbage plugin. For example, if you choose “Frequency” as the channel name it will also appear as the identifier given to the parameter in a plugin host. Each slider that is added to a Cabbage patch corresponds with a plugin parameter on the host side. Min and Max determine the slider range while value initialises the slider to a particular value. If you wish to set Min, Max and Value in one go you can use the `range(min, max, value)` identifier instead. All sliders come with a number box which displays the current value of the slider. By default there is no caption but if users add one Cabbage will automatically place the slider within a captioned groupbox. This is useful for giving labels to sliders.

```
button chan(“chanName”)  pos(x,y),
size(width,height),
items(“OnCaption”,“OffCaption”)
```

Button creates a on-screen button that sends an alternating value of 0 or 1 when pressed. The “channel” string identifies the channel on which the host will communicate with Csound. “OnCaption” and “OffCaption” determine the strings that will appear on the button as users toggle between two states, i.e., 0 and 1. By default these captions are set to “On” and “Off” but users can specify any strings they wish. If users wish they can provide the same string to both the 'on' and 'off' caption. A trigger button for example won't need to have its captions changed when pressed.

```
checkbox chan(“chanName”), pos(x,y),
size(width, height), value(val),
caption(“Caption”), colour(“Colour”)
```

Checkboxes function like buttons. The main difference being that the associated caption will not change when the user checks it. As with all controls capable of sending data to an instance of Csound the “chanName” string is the channel on which the control will communicate with Csound. The value attribute defaults to 0.

```
combobox chan(“chanName”),
caption(“caption”), pos(x,y), size(width, height),
value(val), items(“item1”, “item2”, ...)
```

Combobox creates a drop-down list of items which users can choose from. Once the user selects an item, the index of their selection will be sent to Csound on a channel named by the string “chanName”. The default value is 1 and three items named “item1”, “item2” and “item3” fill the list by default.

```
groupbox caption("Caption"), pos(x,y),
size(width, height), colour("Colour")
```

Groupbox creates a container for other GUI controls. It does not communicate with Csound but is useful for organising the layout of widgets.

```
image pos(x, y), size(width, height), file("file
name"), shape("type"), colour("colour"),
outline("colour"), line(thickness)
```

Image draws a shape or picture. The file name passed to `file()` should be a valid pixmap. If you don't use the `file()` identifier image will draw a shape. Three type of shapes are supported:

- rounded: a rectangle rounded corners (default)
- sharp: a rectangle with sharp corners
- ellipse: an elliptical shape.

```
keyboard pos(x,y), size(width, height)
```

Keyboard creates a virtual MIDI keyboard widget that can be used to test MIDI driven instruments. This is useful for quickly developing and prototyping MIDI based instruments. In order to use the keyboard component to drive Csound instruments you must use the MIDI interop command line flags to pipe the MIDI data to Csound.

3.1.2 MIDI control

In order to control your Cabbage instruments with MIDI CC messages you can use the `midictrl(chan, ctrl)` identifier. `midictrl()` accepts two integer values, a controller channel and a controller number. As is the case with the MIDI keyboard widget mentioned above Cabbage handles all it's own MIDI IO. The following code will attach a MIDI hardware slider to a Cabbage slider widget:

```
slider chan("oscFreq"), bounds(10, 10, 100, 50),
range(0, 1000, 0), midictrl(1, 1)
```

By turning on MIDI debugging in the Cabbage host users can see the channel and controller numbers for the corresponding MIDI hardware sliders. Using `midictrl()` means that you can have full MIDI control over your Cabbage instruments while running in the standalone host. This feature is not included with Cabbage plugins as the host is expected to take control over the plugin parameters itself.

3.1.3 Native Plugin Parameters

Most plugin hosts implement a native interface for displaying plugin parameters. This usually consists of a number of native sliders that corresponds to the number of plugin parameters as can been seen in the following screen-shot.

Fig 3. A Cabbage plugin loaded with Renoise

While slider widgets can be mapped directly to the plugin host GUI, other widgets must be mapped differently. Toggling buttons for example will cause a native slider to jump between maximum and minimum position. In the case of widgets such as comboboxes native slider ranges will be split into several segments to reflect the number of choices available to users. If for example a user creates a combobox with 5 elements, the corresponding native slider will jump a fifth each time the user increments the current selection.

Figure 4. Host automation in Renoise

The upshot of this is that each native slider can be quickly and easily linked with MIDI hardware using the now ubiquitous 'MIDI-learn' function that ships with almost all of today's top DAWs. Because care has being taken to map each Cabbage control with the corresponding native slider, users can quickly set up Cabbage plugins to be controlled with MIDI hardware or through host automation as in fig.4.

4 Cabbage plants

Cabbage plants are GUI abstractions that contain one or more widgets. A simple plant might look like this:

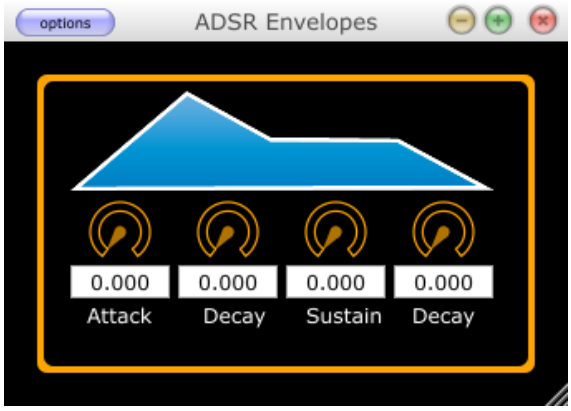

Figure 5. A basic ADSR abstraction.

An ADSR is a component that you may want to use over and over again. If so you can group all the child components together to form an abstraction. These abstractions, or plants, are used as anchors to the child widgets contained within. All widgets contained within a plant have top and left positions which are relative the the top left position of the parent.

While all widgets can be children of an abstraction, only groupboxes and images can be used as plants. Adding the identifier `plant("plantName")` to an image or groupbox widget definition will cause them to act as plants. Here is the code for a simple LFO example:

```
image plant("OSC1"), bounds(10, 10, 100, 120),
colour("black"), outline("orange"), line(4)
{
rslider channel("Sigfreq1"), bounds(10, 5, 80,
80), caption("OSC 1") colour("white")
combobox channel("Sigwave1"), bounds(10, 90,
80, 20), items("Sin", "Tri", "Sqr Bi"),
colour("black"), textcolour("white")
}
```

Fig 6. The code above represents the LFO on the far left.

The `plant()` identifier takes a string that denotes the name of the plant. This is important because all the widgets that are contained between the pair of curly brackets are now bound to the plant in terms of their position. The big advantage to building abstractions is that you can easily move them around without needing to move all the child components too. Once a plant has been created any widget can link to it by overloading the `pos()` identifier so that it takes a third parameter, the name of the plant as in `pos(0, 0, "LFO")`.

Apart from moving plants around you can also resize them, which in turn automatically resizes its children. To resize a plant we use the `scale(newWidth, newHeight)` identifier. It takes new width and height values that overwrite the previous ones causing the plant and all its children to resize. Plants are designed to be reused across instruments so you don't have to keep rebuilding them from scratch. They can also be used to give your applications a unique look and feel. As they can so easily be moved and resized they can be placed into almost any instrument.

Figure 5. An example using several plants together.

5 Examples

The easiest way to start developing Cabbage instruments and plugins is with WinXound.

WinXound is an open-source editor for Csound and is available on all major platforms. Communication between Cabbage and WinXound is made possible through interprocess communication. Once a named pipe has been established users can use WinXound to take complete control of the Cabbage host meaning they can update and export plugins from the Cabbage host without having to leave the WinXound editor.

When writing Cabbage plugin users need to add -n and -d to the CsOptions section of their .csd file. -n causes Csound to bypass writing of sound to disk. Writing to disk is solely the responsibility of the host application(including the Cabbage native host). If the user wishes to create an instrument plugin in the form of a MIDI synthesiser they should use the MIDI-interop command line flags to pipe MIDI data from the host to the Csound instrument. Note that all Cabbage plugins are stereo. Therefore one must ensure to set nchnls to 2 in the header section of the csd file. Failure to do so will results in extraneous noise being added to the output signal.

The first plugin presented below is a simple effect plugin. It makes use of the PVS family of 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 an incoming audio stream. The plugin averages the amp/freq time functions of each analysis channel for a specified time. The output is then spatialised using a jitter-spline generator.

```
<Cabbage>
form caption("PVS Blur") size(450, 80)
hslider pos(1, 1), size(430, 50) \ channel("blur"), min(0),
max(1), \ caption("Blur time")
</Cabbage>
<CsoundSynthesizer>
<CsOptions>
-d -n -+rtmidi=null -M0 -b1024
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 32
nchnls = 2

instr 1

kblurtime chnget "blur"
asig inch 1
fsig  pvsanal   asig, 1024, 256, 1024, 1
ftps  pvsblur   fsig, kblurtime, 2
atps  pvsynth   ftps
apan jspline 1, 1, 3
outs atps*apan, atps*(1-apan)
endin
</CsInstruments>
<CsScore>
f1 0 1024 10 1
i1 0 3600
</CsScore>
</CsoundSynthesizer>
```

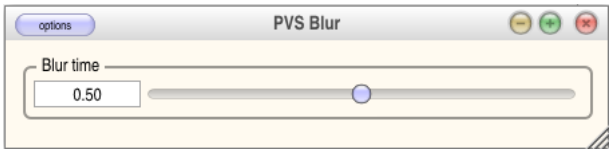

Figure 6. A simple spectral blurring audio effect

The second plugin is a MIDI-driven plugin instrument. You will see how this instrument uses the MIDI-interop command line parameters in CsOptions to pipe MIDI data from the host into Csound. This plugin also makes use of the virtual MIDI keyboard. The virtual MIDI keyboard is an invaluable tool when it comes to prototyping instruments as it sends MIDI data to the plugin just as a regular host would.

```
<Cabbage>
form caption("Subtractive Synth") size(474, \ 270),
colour("black")
groupbox caption(""), pos(10, 1), size(430, \ 130)
rslider pos(30, 20), size(90, 90) \ channel("cf"), min(0),
max(20000), \ caption("Centre Frequency"), \
colour("white")
rslider pos(130, 20), size(90, 90) \ channel("res"),
size(350, 50), min(0), max(1),\ caption("Resonance"),
colour("white")
rslider pos(230, 20), size(90, 90) \ channel("lfo_rate"),
size(350, 50), min(0), \ max(10), caption("LFO Rate"),
colour("white")
rslider pos(330, 20), size(90, 90) \ channel("lfo_depth"),
size(350, 50), min(0), \ max(10000), caption("LFO
Depth"), \ colour("white")
keyboard pos(1, 140), size(450, 100)
</Cabbage>
<CsoundSynthesizer>
<CsOptions>
-d -n -+rtmidi=null -M0 -b1024 \
--midi-key-cps=4 --midi-velocity-amp=5
;-+rtaudio=alsa -odac
</CsOptions>
<CsInstruments>
; Initialize the global variables.
sr = 44100
ksmps = 32
nchnls = 2

massign 0, 1

instr 1
kcf chnget "cf"
kres chnget "res"
klforate chnget "lfo_rate"
klfodepth chnget "lfo_depth"

aenv linenr 1, 0.1, 1, 0.01
asig vco p5, p4, 1
klfo lfo klfodepth, klforate, 5
aflt moogladder asig, kcf+klfo, kres
outs aflt*aenv, aflt*aenv
endin
```

```
</CsInstruments>
<CsScore>
f1 0 1024 10 1
f0 3600
</CsScore>
</CsoundSynthesizer>
```

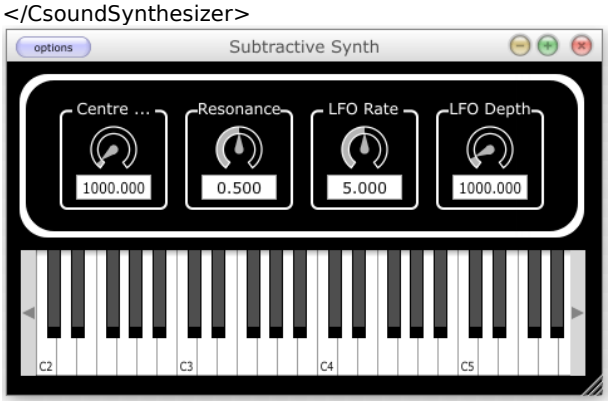

Figure 7. A simple plugin instrument.

6 Conclusion

The system has been shown to work quite well in a vast number of hosts across all platforms. It is currently being tested on undergraduate and postgraduate music technology modules in the Dundalk Institute of Technology and the feedback among users has been very positive. The latest alpha version of Cabbage, including a version of WinXound with support for Cabbage can be found at http://code.google.com/p/cabbage/. A full beta version is expected to be released very soon.

7 Acknowledgements

I'd like to express my sincere thanks to everyone on the Csound, Renoise and Juce mailing lists. Without their help and assistance this project would not have been possible. I'd also like to thank the author of WinXound, Stefano Bonetti for his kind help and assistance over the past few months.

References

[1] Linux VST Homepage http://www.linux-vst.com/

[2] Steinberg Website http://ygrabit.steinberg.de/~ygrabit/public_html/index.html

[3] Apple Audio units Developer Homepage http://developer.apple.com/audio/audiounits.html

[4] WinXound Homepage http://winxound.codeplex.com/

[5] ffitch, J. "On The Design of Csound5." Proceeedings of Linux Audio Developers Conferencence. ZKM, Karlsruhe, Germany. 2004

[6] Cycling 74 Homepage http://cycling74.com/products/maxmspjitter/

[7] PdVst Homepage http://crca.ucsd.edu/~jsarlo/pdvst/

[8] CsoundVST http://michael-gogins.com

[9] Lazzarini, Walsh "Developing LADSPA plugins with Csound" Proceedings of the Linux Audio Developers Conference TU Berlin, Germany. 2007

[10] Walsh, Lazzarni, Brogan. "Csound-based cross-platform plugin toolkits'. Proceedings of the Internation Computer Music Conference, Belfast, NI. 2008

[11] Walsh, R. "Cabbage, a new GUI framework for Csound". Proceedings of the Linux Audio Conference KHM Cologne, Germany. 2008.

[12] WxWidgets Homepage www.wxwidgets.org

[13] Juce website http://www.rawmaterialsoftware.com/juce.php