Cabbage, a new GUI framework for Csound

Rory Walsh
Dundalk Institute of Technology
Dundalk, Co.Louth
Ireland.
rorywalsh@ear.ie

Abstract

Csound is one of the most extensive and powerful audio programming languages available to electroacoustic composers today. With the release of the Csound Host API more and more developers are harnessing the power of Csound in their own applications. This has lead to a welcome increase in the number of Csound front-ends and customised host applications. This paper will describe Cabbage, a new framework for the development of cross-platform standalone Csound software. This text will explore the implementation of said framework and conclude with examples of the framework in use.

Keywords

Computer Music, Musical Signal Processing, Audio programming languages, Interface Design

1 Introduction

Cabbage is a new framework for the development of standalone Csound[1] applications. The framework provides 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 this project is to provide composers and musicians with a means of easily building and distributing high-end audio applications. The toolkit was developed in C++[2] using both the Csound API[3] and the wxWidgets GUI library[4].

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 a control program. Essentially APIs provide developers with a means of harnessing an existing applications functionality within a host application.

The Csound API 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. Although written in C there are a number of different interfaces available to programmers who prefer to use other languages such as Java, Python, Tcl, etc.

Building applications that harness the power of Csound is a relatively simple process. In order to implement the most basic Csound API application users must:

- Create an instance of Csound
- Initialise the instance of Csound
- Compile Csound
- Perform the score
- Destroy Csound

A minimal example in C is presented below[1]:

```
#include <stdio.h>
#include "csound.h"

int main(int argc, char* argv[])
{
/*Create an instance of Csound*/
CSOUND*csound=csoundCreate(0);
/*Initialise the library*/
csoundInitialize(&argc, &argv, 0);
/*Compile Csound*/
int result=
csoundCompile(csound,argc,argv);

if(!result)
{
while(csoundPerformKsmps(csound)==0);
}

/*Finally destroy Csound*/
```

[1]In order to build this code users will need the csound header files and the csound library.

```
csoundDestroy(csound);

return result;
}
```

The real power of the host API is in the way an instance of Csound can communicate with the host application through the use of the channel software bus. Using one of the *'software bus'* opcodes in conjuntion with one of the channel API functions provides users with a powerful interface for communication between Csound and a host applications.

1.2 wxWidgets

wxWidgets is a cross-platform library used for the development of Graphical User Interface (GUI) applications. Apart from being a GUI toolkit wxWidgets also provides great tools for I/O streams, drag and drop, multithreading, image loading and saving, HTML viewing and printing, and much more. A full overview of wxWidgets is beyond the scope of this paper but it is still worth while looking at the basic steps involved in building a simple wxWidgets application.

Every wxWidgets program defines an application class derived from wxApp. This derived class handles the running of the application. Each wxWidgets application will also need an OnInit() function that's called when wxWidgets is ready to start running an application. This OnInit() function is similar to the main() function in C. While the main application is derived from the wxApp class, the main GUI window is created by deriving a class from wxFrame. wxFrame is the main GUI window that users see when their application launches. It usually contains a menu bar as well as other GUI widgets which users interact with. Below is an example of a very minimal wxWidgets application.

```
#include "wx/wx.h"

class myFrame : public wxFrame
{
public:
myFrame(const wxString& title, const
wxPoint& pos, const wxSize& size,
long style = wxDEFAULT_FRAME_STYLE);
};

myFrame::myFrame(const wxString& title,
const wxPoint& pos,
const wxSize& size, long style)       :
wxFrame(NULL, -1, title, pos, size,
style)
{
}

class myApp : public wxApp
{
public:
   virtual bool OnInit();
};

IMPLEMENT_APP(myApp)

bool xxxApp::OnInit()
{
myFrame *frame = new myFrame("Hello
World", wxPoint(50, 50), wxSize(450,
340));

frame->Show(TRUE);
return TRUE;
}
```

1.3 GUIs and Csound

Providing Csound users with tools to develop GUI instruments is not a unique concept. Since Csound 4.23 it has been possible to develop GUI instruments using Csound FLTK opcodes. While the FLTK opcodes do provide users with a means of developing graphical user interfaces their implementation in Csound is not without it's problems, in particular when it comes to using them in multi-threaded applications[5].

2 The Cabbage GUI framework

Cabbage is being developed to help users realise their own cross-platform standalone audio software which combines the processing power of Csound with the GUI possibilities of wxWidgets. It is forseen that users will employ the framework to create task specific applications such as computer music instruments for works of electroacoustic music, research tools for audio programming and pedagogical tools for teaching computer music and digital signal processing techniques.

2.1 Technical aspects

As previously mentioned the toolkit was developed in C++. It use both the Csound and CsoundPerformanceThread classes which are part of the 'interfaces' library; an auxiliary library for Csound which provides interfaces for several different programming languages.

The toolkit can be split into two parts, the application framework and the pseudo-compiler. The application framework is a generic binary file that dynamically creates GUI forms, controls and menus depending on the specific instructions provided in the associated Csound file. The pseudo-compiler which runs from the command

line bundles everything together by making a copy of the generic binary file and appending the contents of a Csound file to the end of said binary. The advantage of embedding the Csound code into the binary executable is that users need only distribute the executable.

3 Cabbage GUI Syntax

The syntax used to create GUI controls is quite straightforward and should be provided within special Cabbage tags, i.e., <Cabbage> and </Cabbage> at the top of a unified Csound file. Each line of Cabbage specific code should relate to one GUI control only and the syntax is non case-sensitive.

3.1 GUI Controls

Each and every Cabbage control has 4 common parameters; their position on screen and their size. Apart from position and channel all other parameters are optional and if left out the default values will be assigned. Parameters can appear in any order. Below is a list of the different GUI controls currently available in Cabbage.

```
form caption("title"), position(Top,
Left), size(Width, Height)
```

Form creates the main application window. Top, Left, Width and Height are all integer values. The default values for size are 400x600. Forms do not communicate with an instance of Csound, only child widgets and menus contained within a form can communicate with an instance of Csound, therefore no channel identifier is needed.

```
scrollbar channel("chanName"),
position(Top, Left), size(Width, Height),
min(float), max(float), value(float),
kind("horizontal"/"vertical")
```

Scrollbar creates a scrollbar/slider that can be used to send data to Csound on the channel specified through the "chanName" string. Min and Max will determine the slider range while value initialises the slider to a particular value. "kind" specifies whether the slider will appear horizontally or vertically. "kind" is set to horizontal by default.

```
button channel("chanName")
position(Top, Left), size(Width,
Height),OnOffCaptio("OnCaption","OffCapti
on")
```

Button creates a button on screen that can be used to turn instruments on or off. It can also be used to turn parts of certain instruments on and off. The "chanel" string identifies the channel on which the host will communicate with an instance of Csound. "OnCaption" and "OffCaption" determine the strings that will appear on the button as users toggles between two states, i.e., 0 or 1. By default these captions are set to "On" and "Off" but the user can specifiy any strings they wish.

```
checkbox channel("chanName"),
position(Top, Left), size(Width, Height),
value(val), caption("Caption")
```

Checkbox creates a checkbox which functions like a button only 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 while the caption is set by defalt to the same name as the channel.

```
combobox channel("chanName"),
position(Top, Left), size(Width, Height),
value(val), items("item1", "item2", ...)
```

Combobox creates a drop-down list of items which end-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 0 and three items named "item1", "item2" and "item3" fill the list by default.

```
textctrl caption("Caption"),
position(Top, Left), size(Width, Height),
beveltype("bevelType"), colour("colour"),
fontColour("fontColour")
```

TextCtrl creates a static text control. This control does not communicate with Csound, hence one need not provide a channel name. "bevelType" can be set to "lower", "raised" or "none" while back and font colour specify the colour for the background and foreground text[2].

```
groupbox caption("Caption"),
position(Top, Left), size(Width, Height),
colour("Colour")
```

Groupbox creates a container for other GUI controls. It does not communicate with Csound but

[2]Font and background colours must be given as wxWidgets colours which are defined in the wxColourDatabase class. More information can be found in the wxWidgets documentation.

is useful when it comes to organising different widgets.

3.2 Cabbage Menu controls

Cabbage also provides users with a means of constructing their own user-defined menus which can be used to send information to an instance of Csound. Cabbage menu controls can also be used to do the following:

- Start and stop a Csound performance.
- Open the Csound console so users can see the messages being output by Csound
- Set CsOptions before an application starts processing
- Exit an application

The basic syntax for every menu is as follows:

```
menu channel("chanName"), value(int), (...)
```

Following this users specify the commands they wish to appear on the menu. The value that is sent on the associated channel will be the index of the item that the user selects. The commands are as follows:

```
RunCsound("StartCaption","StopCaption")
```

Adding this to a menu will create a file command that will start or stop an instance of Csound when it is pressed.

```
StdOut("Caption")
```

StdOut will add a new menu command which when clicked will cause the Csound console to appear. This is invaluable when debugging applications

```
Exit("Caption")
```

Exit will create a menu command that will stop a performance, destroy an instance of Csound and close the GUI application.

```
CsSetup("Caption")
```

CsSetup will provide end-users with a menu command that displays a text box where they can edit their instruments CsOptions before a performance. This is generally used to select certain audio devices and can also be used to pass strings to Csound.

```
TopItem("Caption")
```

TopItem places a user defined command at the top of the menu bar.

```
Item("Caption")
```

The Item command places a command on the menu bar. The following code shows an example which creates two menus. The first one creates a system-type menu that will start/stop an instance of Csound, let the user view the Csound output console and let the user exit the application. The second menu can be used to send data to Csound, in this case frequencies.

```
<Cabbage>
form caption("Example"), \
position(303, 137), size(10, 10)
menu channel("menu_0"), value(0), \
TopItem("File"), \
RunCsound("Start_Csound","Stop_Csound"),\
StdOut("View_Console"), Exit("Close"), \
DropDownItem("tester)

menu channel("menu_1"), value(4), \
TopItem("Freq"), \
DropDownItem("100Hz"), \
DropDownItem("200Hz"), \
DropDownItem("300Hz"), \
DropDownItem("400Hz"), \
DropDownItem("500Hz")
</Cabbage>
```

The menu can be used to interact with Csound using the `chnget` opcode. For example:

```
(...)
kfreqChoice chnget "menu_1"
if(kfreq==0) then
aout oscil 10000, 100, 1
elseif(kfreq==1) then
aout oscil 10000, 200, 1
(...)
```

4 Putting it all together

In order for different Cabbage widgets to communicate with an instance of Csound one must make use of the previously mentioned channel opcodes. While the `chnget` and `chnset` opcodes are the easiest to implement `chnexport` allows users to set up global channels for both input and output. The syntax for chnexport is defined in the Csound manual as:

```
gkval chnexport Sname, imode[, itype, idflt, imin, imax]
```

Sname is the channel name. This should match the "chanName" of the Cabbage control the user wishes to communicate with. `imode` specifies

whether the user wishes to set up a channel for output or input, or both. idflt specifies the default value for the channel. itype specifies the channel subtype. This defaults to 0 in which case idflt, imin, and imax are ignored. It can also be set to integer values only, linear or exponential. `chnexport` generally appears outside instrument definitions. Below is a simple example that uses a slider to change the frequency of an oscillator.


```
<Cabbage>
form captopn("Freq App"), \
position(10, 10), size(219, 90)
menu channel("menu_0"), value(0), \
TopItem("File"), \
RunCsound("Start","Stop"), \
StdOut("View_Console"), Exit("Close")
scrollbar channel("freq"), \
position(8, 12), value(0), max(1000), \
min(0)
</Cabbage>
<CsoundSynthesizer>
<CsOptions>
-odevaudio -b10 -idevaudio
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
/* retrieve data from channel "freq".
Mode is set to 1 to indicate that the
channel is used for input */
gkscrollbar_0 chnexport "freq", 1

instr 1
/* use the data from channel "freq" to
change frequency of oscillator */
a1 oscil 10000, gkscrollbar_0, 1
out a1
endin

</CsInstruments>
<CsScore>
f1 0 1024 10 1
i1 0 100
</CsScore>
</CsoundSynthesizer>
```

In order to build the standalone application users must run the Cabbage compiler from the command line as follows:

```
>cabbage freqApp.csd SampleApp
```

To illustrate how data can be sent from an instance of Csound to a host application automation can be added to the instrument presented above. We can instruct our Csound instrument to send data to the scrollbar on the channel named "freq" as in the example below.

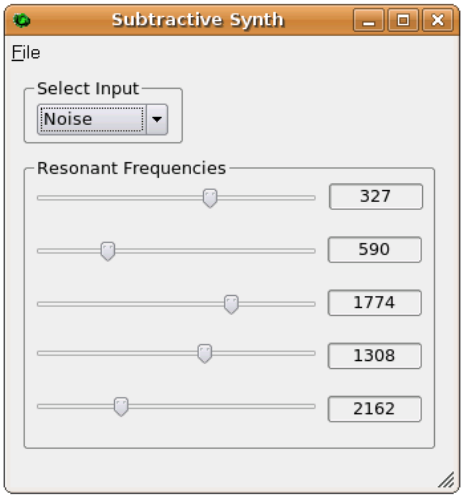
```
<CsInstruments>
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

/* Mode is set to 3 to indicate that the
channel is used for both input and output
*/
gkscrollbar_0 chnexport "freq", 3

instr 1
kauto linseg 0, 2, 1000, 2, 100
gkscrollbar_0 = kauto
a1 oscil 10000, gkscrollbar_0, 1
out a1
endin

</CsInstruments>
```

Sending data from an instance of Csound to the host application is useful when users want their sliders to mimic real faders such as those found on a MIDI controller. The following instrument will cause our on-screen sliders to move in sympathy with a MIDI fader.

```
(...)
<CsInstruments>
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

gkscrollbar_0 chnexport "freq", 3

instr 1
kMsldr ctrl7 7, 1, 0, 1000
gkscrollbar_0 = kMsldr
a1 oscil 10000, gkscrollbar_0, 1
out a1
endin

</CsInstruments>
```

5 Examples

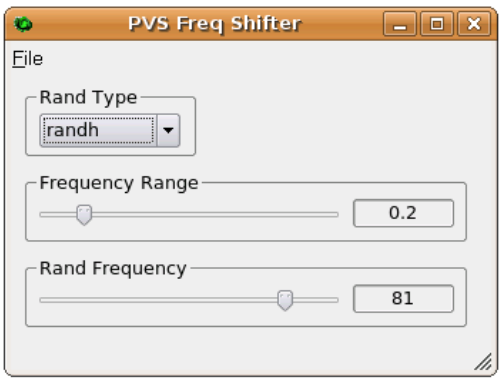
Presented below are two complete examples. The first one is a simple subtractive synthesiser. The user can choose between an additive set of harmonically related cosine partials or band-limited noise as the input source. The sliders are used to filter frequencies from particular frequency bands.




```
<Cabbage>
form caption("Subtractive Synth") \
position(10, 10) size(250, 287)
menu channel("menu_0") value(0), \
TopItem("File"),  \
RunCsound("Start", "Csound"), \
StdOut("View_Console"), \
Exit("Close")
groupbox caption("Resonant Frequencies")\
, position(62, 12), size(220, 164), \
scrollbar channel("scrollbar_1"), \
position(85, 23), size(160, 17), min(0),\
max(500), value(200)
textctrl channel("panel_1"), \
position(87, 184), size(40, 16) \
value("0")
scrollbar channel("scrollbar_2"), \
position(110, 23), size(160, 17), \
min(500), max(1000), value(800)
textctrl channel("panel_2"), \
position(110, 184), size(40, 16), \
value("0")
scrollbar channel("scrollbar_3"), \
position(137, 23), size(160, 17), \
min(1000), max(1500), value(1300)
textctrl channel("panel_3"), \
position(137, 183), size(44, 16), \
value("0")
scrollbar channel("scrollbar_4"),
position(165, 23), size(160, 17), \
min(1500), max(2000), value(1800)
textctrl channel("panel_4"),
position(166, 184), size(44, 16), \
value("0")
scrollbar channel("scrollbar_5"), \
position(193, 23), size(160, 17), \
min(2000), max(2500), value(2000)
textctrl channel("panel_5)", \
position(194, 184), size(44, 16), \
value("0")
combobox channel("combobox_12"), \
position(24, 36), size(60, 21), \
value(0), items("Buzz", "Noise")
groupbox caption("Select Input"), \
position(8, 12), size(116, 48)
</Cabbage>
<CsOptions>
-odevaudio -b10
</CsOptions>
<CsoundSynthesizer>
<CsInstruments>
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

gkscrollbar_1 chnexport "scrollbar_1", 1
gkscrollbar_1 chnexport "panel_1", 2
gkscrollbar_2 chnexport "scrollbar_2", 1
gkscrollbar_2 chnexport "panel_2", 2
gkscrollbar_3 chnexport "scrollbar_3", 1
gkscrollbar_3 chnexport "panel_3", 2
gkscrollbar_4 chnexport "scrollbar_4", 1
gkscrollbar_4 chnexport "panel_4", 2
gkscrollbar_5 chnexport "scrollbar_5", 1
gkscrollbar_5 chnexport "panel_5", 2
gkSrc chnexport "combobox_12", 1

instr 1
if(gkSrc==0) then
abuz  buzz  10000, 0, 1000, 1;
elseif(gkSrc==1) then
abuz randi 10000, 10000;
endif

iq1 = 300
iq2 = 300

; filterbank 1
af11   reson  abuz,  gkscrollbar_1,
gkscrollbar_1/iq1
af12   reson  af11,  gkscrollbar_2,
gkscrollbar_2/iq1
af13   reson  af12, gkscrollbar_3,
gkscrollbar_3/iq1
af14   reson  af13, gkscrollbar_4,
gkscrollbar_4/iq1
af15   reson  af14, gkscrollbar_5,
gkscrollbar_5/iq1

; filterbank 2
af21   reson  abuz, 1700, 1700/iq2
af22   reson  af21, 2000, 2000/iq2
af23   reson  af22, 2800, 2800/iq2
af24   reson  af23, 4000, 4000/iq2
af25   reson  af24, 5900, 5000/iq2

amix =  af11+af12+af13+af14+af15
aout balance amix, abuz
out aout
endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1
i1 0 300
</CsScore>
</CsoundSynthesizer>
```

The second exmple creates a real-time frequency shifter which makes use of several phase vocoder streaming (PVS) opcodes[6]. The frequency is set randomly and users can specify the frequency range and the rate at which random numbers are generated. Users can also select the type of

random number generation they wish to use in order to control frequencies.




```
<Cabbage>
form caption("PVS Freq Shifter"),
position(10, 10), size(284, 219)
menu channel("menu_0"), value(0), \
TopItem("File"), \
RunCsound("Start|Stop"), \
CsSetup("Settings"), \
StdOut("View_Console"), Exit("Close")
groupbox caption("Rand Type"),
position(8, 12),\ size(80, 52)
combobox channel("randType"), \
position(28, 24), size(60, 21), \
value(1), items("randh", "randi")
groupbox caption("Frequency Range"), \
position(60, 12), size(252, 44)
scrollbar channel("freqRange"), \
position(78, 18), size(172, 17), \
min(0), max(300)
textctrl channel("panel_4"), \
position(78, 196), size(60, 16), \
value("0"), beveltype("lowered"),
groupbox caption("Rand Frequency"), \
position(252, 48), size(106, 12)
scrollbar channel("randFreq"),
position(126, 19), size(172, 17), \
min(0), max(100)
textctrl channel("panel_7"), \
position(126, 196), size(60, 16), \
value("0"), beveltype("lowered")
</Cabbage>
<CsoundSynthesizer>
<CsOptions>
-odevaudio -b10 -idevaudio
</CsOptions>
<CsInstruments>
gkRandType chnexport "randType", 1
gkFreqRange chnexport "freqRange", 1
gkFreqRange chnexport "panel_4", 2
gkRandFreq chnexport "randFreq", 1
gkRandFreq chnexport "panel_7", 2

sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

instr 1
asig1 inch 1;
gkRange = gkFreqRange/100

if (gkRandType==1) then
krand randh gkFreqRange/100, gkRandFreq
elseif(gkRandFreq==2) then
krand randi gkFreqRange/100, gkRandFreq
endif

fim   pvsanal  asig1,1024,256,1024,0
pvoc analysis
fsig  pvscale fim, krand, 2
apvs  pvsynth fsig
pvoc synthesis
acomb1 comb apvs, 1, 0.2
acomb2 comb apvs, 1, 0.15

outs acomb1, acomb2

endin

</CsInstruments>
<CsScore>
f1 0 1024 10 1
i1 1 200
</CsScore>
</CsoundSynthesizer>
```

6 Conclusion

The current version of Cabbage works quite well but there is still plenty of development to be carried out before the first public release. Work has begun on developing an interface whereby users can easily add new components to the toolkit. A component for viewing spectral data is also currently being developed.

Another idea being investigated is providing users with a simple mechanism for routing audio between Cabbage applications so that each application can become part of a bigger modular system. Improvements also need to be made to the build system. Currently Cabbage can be built using a GNU makefile but other build systems are being investigated. A simple 'drag and drop' interface for designing Cabbage applications is also being considered.

7 Acknowledgements

I would like to thank all the developers on the Csound mailing lists for their support and advice. In particular I would like to thank Victor Lazzarini, Matt Ingalls, Michael Goggins, and Istvan Varga for their work on the Csound Host API, without which this project would not have been possible. Finally I wish to thank Dundalk Institute of Technology, Ireland, for their continued support of my research.

References

[1] Barry Vercoe et Al. 2005. The Csound Reference Manual. http://www.csounds.com/manual/html/index.html

[2] Bjarne Stroustrop. 1991. The C++ Programming Language, second edition. Addison-Wesley, New York.

[3] John ffitch. 2004. On The Design of Csound5. Proceedings of the 3rd Linux Audio Developers Conference. ZKM, Karlsruhe, Germany.

[4] wxWidgets homepage: www.wxWidgets.org

[5] Csound mailing list archive, May 02, 2006: http://www.nabble.com/Standalone-FLTK-exe%27s-from-Cabbage-tf1537330.html#a4189937

[6] 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.