Cabbage, a new GUI framework for Csound

Rory Walsh

Contents

Introduction

• Cabbage is a new framework for the development of cross-platform standalone Csound-driven software.

• The framework provides Csound users who have no low-level programming skills with a simple albeit powerful toolkit for the development high-end 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.

• It is foreseen 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.

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.

- API's provide developers with a means of harnessing an existing applications functionality within a host application.

- The Csound API can be used to control an instance of Csound through a series of different calling functions. In short, the Csound API makes it possible to harness all the power of Csound in your own applications.

- 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, Lisp, C++, etc.

Using the Csound API

- In order to use the API you will need to download the Csound source code which can be found here: http://csound.sourceforge.net/

- To create a minimal Csound host users must

  1) Create an instance of Csound using csoundCreate()
  2) Initialise the Csound library using csoundInitialize()
  3) Reset and prepare an instance of Csound for compilation using csoundPreCompile()
  4) Compile the Csound input files (such as an orchestra and score) as directed by the supplied command-line arguments using csoundCompile()
  5) Perform the score using csoundPerformKsmps()
  6) Destroy the instance of Csound using csoundDestroy()

A simple Csound application

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

int main(int argc, char* argv[])
{
int result=0;

/*Create an instance of Csound*/
CSOUND*csound=csoundCreate(0);

/*Initialise the library*/
csoundInitialize(&argc, &argv, 0);

/*precompile*/
csoundPreCompile(csound);

/*Compile Csound, i.e., run Csound with
the parameters passed via the command line.
Return false is successful*/
result=csoundCompile(csound,argc,argv);

/*check to see that Csound compiled Ok, then
start the performance loop. csoundPerformKsmps()
returns false if successful*/
if(!result)
{
while(csoundPerformKsmps(csound)==0);
}

/*Finally destroy Csound*/
csoundDestroy(csound);

return result;
}
```

• Cabbage was written using the Csound C++ API interface which provides dedicated Csound and performance thread classes. Here is a minimal example:

```
#include "csound.hpp"
#include "csPerfThread.hpp"
#include <conio.h>

int main(int argc, char *argv[])
{
/*create a Csound object*/
Csound* csound = new Csound;
/*pre-compile instance of csound*/
csound->PreCompile();
/*compile instance of csound*/
csound->Compile(csound,argc,argv));

/* create a CsoundPerfThread object */
CsoundPerformanceThread* perf = new
CsoundPerformanceThread(csound);

/* start csound thread */
perf->Play();

/* pause program, it won't go stop performing until the
user presses a key */
getch();

/* stop thread */
perf->Stop();
/*delete instance of csound*/
delete csound;
}
```

- The real power of the host API is the way in which two-way communication can be set up between the host application and Csound through the use of the named channel software bus.

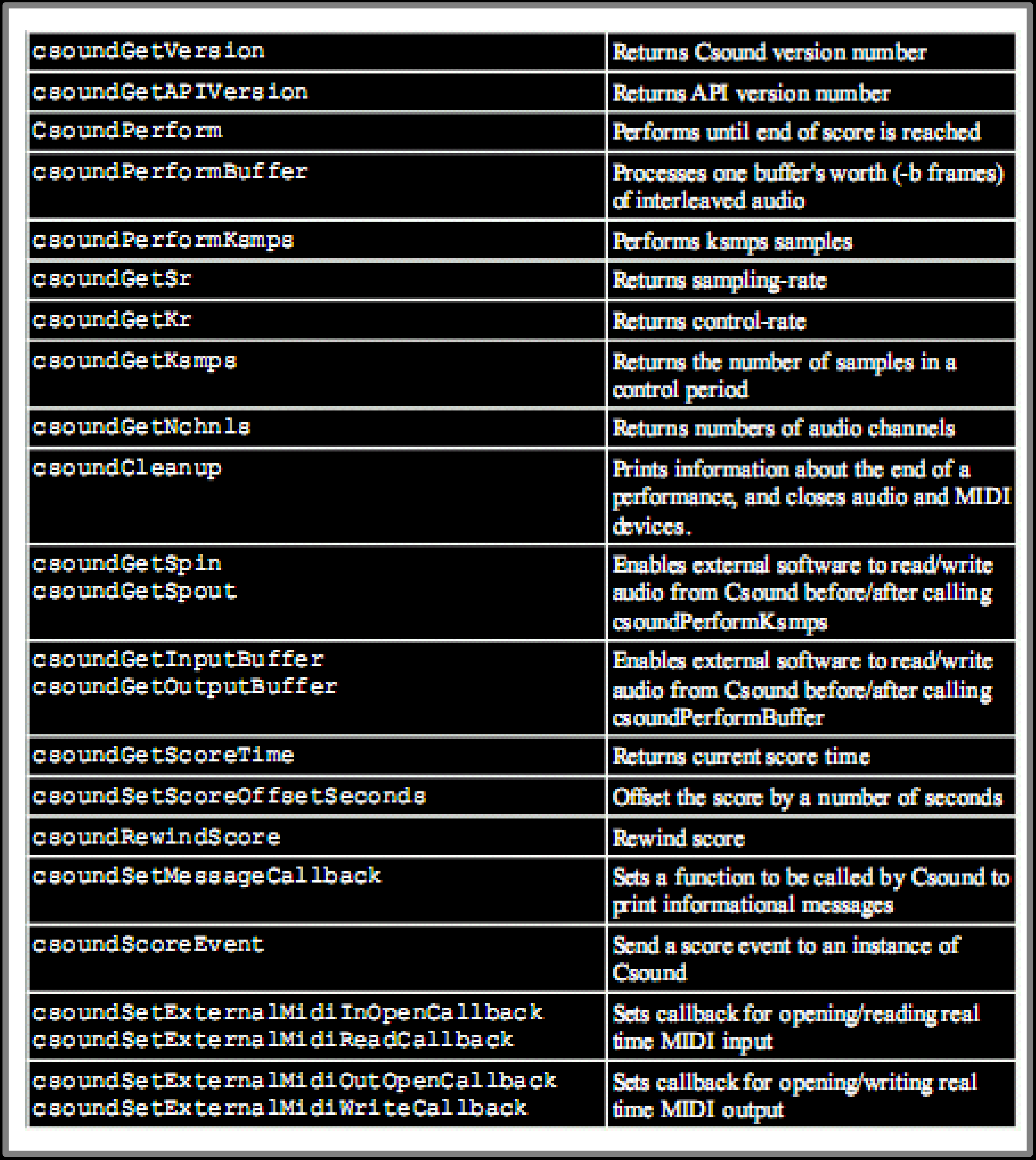
- Using one of the *'channel'* opcodes in conjunction with one of the channel API functions provides users with a powerful interface for communication between Csound and a host applications.

Example demonstrating the use of the named channel bus

```
#include "csound.hpp"
#include "csPerfThread.hpp"
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
int hold=1;
/*create a Csound object*/
Csound* csound = new Csound;
/*pre-compile instance of csound*/
csound->PreCompile();
/*compile instance of csound*/
csound->Compile("test.csd");
/* create a CsoundPerfThread object */
CsoundPerformanceThread* perf =
new CsoundPerformanceThread(csound->GetCsound());

/* start csound thread */
perf->Play();
/* pause program, it won't stop performing until the user presses 0 */
while(hold){
cin >> hold;
csound->SetChannel("freq", (MYFLT)hold);
}

/* stop thread */
perf->Stop();
/*delete instance of csound*/
delete csound;
}
```

```
<CsoundSynthesizer>
<CsInstruments>
sr = 44100
kr = 64
nchnls = 1

instr 1
kfreq chnget "freq"
a1 oscil 10000, kfreq, 1
out a1
endin

</CsInstruments>
<CsScore>
i1 0 100
</CsScore>
</CsoundSynthesizer>
```

Useful API functions



`csoundGetVersion`	Returns Csound version number
`csoundGetAPIVersion`	Returns API version number
`CsoundPerform`	Performs until end of score is reached
`csoundPerformBuffer`	Processes one buffer's worth (-b frames) of interleaved audio
`csoundPerformKsmps`	Performs ksmps samples
`csoundGetSr`	Returns sampling-rate
`csoundGetKr`	Returns control-rate
`csoundGetKsmps`	Returns the number of samples in a control period
`csoundGetNchnls`	Returns numbers of audio channels
`csoundCleanup`	Prints information about the end of a performance, and closes audio and MIDI devices.
`csoundGetSpin` `csoundGetSpout`	Enables external software to read/write audio from Csound before/after calling csoundPerformKsmps
`csoundGetInputBuffer` `csoundGetOutputBuffer`	Enables external software to read/write audio from Csound before/after calling csoundPerformBuffer
`csoundGetScoreTime`	Returns current score time
`csoundSetScoreOffsetSeconds`	Offset the score by a number of seconds
`csoundRewindScore`	Rewind score
`csoundSetMessageCallback`	Sets a function to be called by Csound to print informational messages
`csoundScoreEvent`	Send a score event to an instance of Csound
`csoundSetExternalMidiInOpenCallback` `csoundSetExternalMidiReadCallback`	Sets callback for opening/reading real time MIDI input
`csoundSetExternalMidiOutOpenCallback` `csoundSetExternalMidiWriteCallback`	Sets callback for opening/writing real time MIDI output

wxWidgets

- wxWidgets is a cross-platform library used for the development of Graphical User Interface (GUI) applications.

- wxWidgets was started in 1992 by Julian Smart at the University of Edinburgh

- wxWidgets gives you a single, easy-to-use API for writing GUI applications on multiple platforms that still utilise the native platform's controls and utilities

wxWidgets basics

• Every wxWidgets program defines an application class derived from the wxApp class. This derived class handles the running of the application.

• While the application iteseelf 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.

A 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;
}
```

Csound and Graphical Interfaces

- Providing Csound users with tools to develop GUI instruments is not a unique concept. Since Csound 4.2 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

- Cabbage is being developed as an alternative rather than a replacement for FLTK GUI opcodes.

- Cabbage is not dependant on opcodes so the instrument code can easilt be ported to other systems such as csLADSPA for example.

The Cabbage GUI framework

• Cabbage can be split into two parts, the application framework and the application builder.

• 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 application builder 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.

Cabbage Syntax

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

• Each and every Cabbage control has 4 common parameters; their position on screen, top, left and their size, width height. All other parameters are optional and if left out the default values will be assigned. Parameters can appear in any order.

Cabbage controls

- Apart from `form` and `groupbox` controls every other GUI control communicates with Csound on a named channel specified using the channel() identifier. For example:

```
scrollbar channel("freq_1"), pos(85, 23),
max(500), value(200)
```

- In the example above a scrollbar is created which will communicate to Csound on a channel named “freq_1”. Our Csound instrument code could look like this:

```
instr 1
kfreq chnget “freq_1”
a1 oscil 20000, kfreq, 1
out a1
endin
```

GUI controls currently available in Cabbage:

- Form
- Application Menus
- Scrollbar
- Button
- Combobox
- Checkbox
- Groupbox
- TextCtrl
- PVS Viewer / VU Level Meter (under construction!)

All GUI controls except for Forms and Groupboxes are full duplex. Messages can be sent and received by any of them.

Building cabbage applications

- In order to build the final application users must run the cabbage builder application from the command line as follows:

```
cabbage  csound_file.csd   output_app
```

- Running the the cabbage application builder simple appends the given csd file to a copy of the cabbage binary data file. When the new binary file is opened it opens the csd file, reads the cabbage syntax all dynamically creates the appropriate controls.

A basic cabbage application

```
<Cabbage>
form caption("Freq example") size(280, 50)
menu TopItem("File"), RunCsound("Start", "Stop"), StdOut("View Console"), Exit("Close")
scrollbar channel("scrollbar_1"), pos(15, 23), min(200) max(500), value(200)
textctrl channel("panel_1"), pos(17, 184), value(200)
</Cabbage>
<CsoundSynthesizer>
<CsOptions>
csound  -odevaudio -b64 -B4096 temp.orc temp.sco
</CsOptions>
<CsInstruments>
; Initialize the global variables.
sr = 44100
ksmps = 64
nchnls = 2

gkscrollbar_1 chnexport "scrollbar_1", 1
gkscrollbar_1 chnexport "panel_1", 2

instr 1
gkscrollbar_1 init 200
a1 oscil 10000, gkscrollbar_1, 1
out a1, a1
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1
i1 0 300
</CsScore>
</CsoundSynthesizer>
```

Future development

• The system has been tested on Linux and Windows and works well. There is however plenty of work to be carried out before the first public release; first and foremost the code has to be tidied up; a template for writing new components must be added; more examples need to be distributed with the system.

Currently under investigation:

- The idea of allowing different cabbage applications to plug into each other in order to form a larger modular system, this can easily be achieved using audio rate channels
- Adding an integrated editor component so that users can edit their instruments and GUIs during run-time
- A GUI designer akin to those found in IDEs such as Glade and CodeBlocks.

Acknowledgements

• I would like to thank all the developers on the Csound mailing lists for their support and advice. In particular Victor Lazzarini, Matt Ingalls, John ffitch, Michael Goggins, and Istvan Varga for their work on the Csound Host API

• I also wish to thank all the developers on the wxWidgets mailing lists, in particular Vadim Zeitlin.

• Finally I wish to thank Dundalk Institute of Technology, Ireland, for their continued support of my research.

More info

- For more information please visit http://sourceforge.net/projects/cabbage/