MIUP Portable User Interface for Music

Example of jo_tracker - a tracker interface for Csound

Johann PHILIPPE[⋆]

No Institute Given

Abstract. This article presents graphical tools designed to work with Csound. First section introduces the context in which those tools were built. Then, second part presents MIUP, an open source graphical library designed to build audio softwares. Finally, last part describes jo_tracker, a tracker software for Csound built with MIUP.

Keywords: MIUP, IUP, jo_tracker, Csound, User Interface, Lua, C++

1 Introduction

In it's relatively recent relashionship with softwares and computing tools, contemporary music has been experiencing many difficulties inherent to preservation of tools. This is particularly true for mixt and electroacoustic music, which, at the same time, benefits a lot of those technologies. However, it is harder today to play a music from the last thirty years than to play some complex musics from the early 20th century. Most of the time, this difficulty appears when the electronic part of a music is based on an old program that is no longer working, or an old Max MSP patcher. In this kind of cases, it is almost an archeological work to find how was supposed to work this old software, and it becomes longer to update this program than it was to first create. As it is an important preservation problem, it must be a concern for composers who uses this technologies. Csound likely stands as the best alternative to this preoccupations, for many reasons. First, Csound is open-source, so a program could be reconstructed from scratch. Also, Csound developpers take care of retro-compatibility, which allows an old program to be played on a recent distribution. Moreover, Csound has an important community sharing knowledge about sound and music computing. It is why tools presented in this article are mostly designed to work with Csound, yet it could work with other audio programming languages.

2 MIUP - a C++ user interface library for music

MIUP stands as portable user interface for music. This is a cross platform toolkit designed to easily write musical softwares using Csound. It uses IUP [1], a cross platform C library working with system native interface elements. It

⋆ Thanks to François Roux.

is fully distributed as source code under MIT license (except a modified version of CsoundThreaded). MIUP has a few dependencies : all of them are licensed under MIT. Some of them are included as source code inside the project repository. Though, a few ones need to be linked to the program in order to work : the IUP library (all of it, including canvas draw and im), sqlite3, sndfile. Of course, Csound needs to be linked to the program if it is used as the application's audio engine. MIUP project contains a set of C++ class which describes useful tools for musical softwares : sliders, levelmeters, spinboxes (...), but also more complex class like a curve editor, a waveform visualization plot, a code editor for Csound... MIUP provides a connection mechanism that can be used to fire callbacks between several objects. It also provides a callback system to retrieve Csound output channels and display their values inside the interface. MIUP allows user to take full benefit from Csound with a flexible user interface. It allows user to write its own widgets, following a simple design diagram. This library was first written in Lua as a toolkit to write a particular software : the jo_tracker. It has been fully rewritten in C++ to extend its potential to other softwares.

2.1 MIUP basic functionnalities

IUP C library provides one data type that is used for every interface element :

```
Ihandle *
```

The interface element can be initialized like this :

```
Ihandle *IupButton(const char *action)
```

It also provides a set of functions that can be used to modify attributes of interface elements.

```
void IupSetAttribute(Ihandle *element, const char *name, const char *value)
const char * IupGetAttribute(Ihandle *element, const char *name)
```

Since every IUP element is of type *Ihandle **, users can easily construct some complex interface architecture, imbricating boxes (layouts) inside other boxes. MIUP base class `MiupObj` is a simple C++ wrapper to this C mechanism. It is recommanded that every MIUP interface element inherits from this class. The class contains a private `Ihandle *` that holds a reference to the interface element, and implements some public methods to modify attributes of the object :

```
void setAttr(const char *name, const char *value)
const char *getAttr(const char *name)
Ihandle *getObj()
```

Every MIUP widget inherits from this base class called MiupObj, and constructs the interface element with a different IUP initializer function. The `getObj()`

method returns the `Ihandle *` pointer. It is a necessary method for every interface element, and it allows a compatibility with the standard IUP syntax. The `Ihandle *` element returned by the `getObj()` method will be the one displayed in the interface.

2.2 Main Features

Here is a quick list of the available widgets and classes :
-Widgets : button, toggle, sliders, levelmeter, gainmeter, spinbox, matrix
-Plots : curve editor, waveform visualizer (realtime and soundfile)
-Containers : boxes (vertical, horizontal, scrollable), radio...
-Audio : Threaded callbacks, AudioFileReader, CsoundThreaded (slightly modified)
-Text : Text widget (one line, or multiline), Scintilla editor
-Utilities : filesystem, string conversion, templated print facilities, some data types, signaled value...
It also contains a set of features, including JSON [5], Signal and Slot [4] used to connect objects, and a thread safe callback mechanism for Csound control channels.

2.3 Code Examples

Any MIUP program must contain at least a call to `Miup::Init()` at the beginning, and `Miup::MainLoop()`, `Miup::Close()` at the end. Creating a widget can be as simple as :

```
Slider<double> sl(-90,-90,6,0.01,"HORIZONTAL");
LevelMeter<double> lv(-90,-90,6);
Button but("PLAY"); // creates a button displaying "PLAY"
```

Widgets can be pushed inside containers like this `Vbox vbx(&sl,&lv,&but);`.Then, the final container must be pushed in a new dialog.

```
Dialog dlg(&vbx);
dlg.show();
```

When their internal callback is triggered (like button click), Widgets emits signals that can be connected to any function or method with the same signature. For example, we could do :

```
sl.valueChangedSig.connect(&lv,&LevelMeter<double>::setLevelMeter);
```

This would update level meter value to slider value. If the signature is different, the connect method can also be called with a lambda as argument. It allows to connect one signal to multiple actions in one statement.

```
CsoundThreaded cs;
sl.valueChangedSig.connect([&cs](double val){cs.setChannel("gain",val);});
```

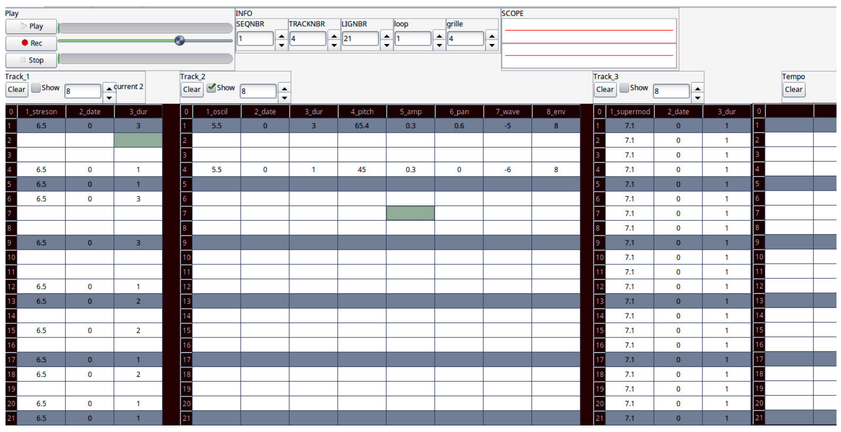
This would send the slider value to Csound as a control channel when it changes. Interface can also be refreshed with Csound control output control channels. This functionnality works by passing a std::function as argument to CsoundThreaded pushMethodCallback method. It can be done using lambdas, or std::bind.

```
cs.pushMethodCallback("level",[&lv](double val){lv.setLevelMeter(val);});
```

Internally, Csound will look the "level" channel value at k-rate, and push the lambda and the value in a queue. The queue is then processed in the `Miup::MainLoop()` function. This part of the work benefits from the great work of Michael Gogins on *CsoundThreaded* [2], which has been slightly modified to perform those callbacks.
Example MIUP Simple program playing a sine

3 J_tracker - a tracker interface for Csound

Tracker softwares, also called soundtracker, are musical sequencers which tracks are based on a grid. Users can write values in the grid, corresponding to synthesizers instructions. Those are often MIDI instructions (note, velocity), that can be added to some basic controls on the note (pan,delay...). Sequencer starts at the top of the grid, and iterates over each line, before reaching the last one. Soundtrackers can be thought as step sequencers with an improved writing precision. This kind of software was very popular in the 1990's. Today, only a few of them are still under active development, including Renoise and OpenMPT.



INFO: SEQNBR 1, TRACKNBR 4, LIGNBR 21, loop 1, grille 4

Track_1: Clear, Show ☐, 8, current 2

0	1_streson	2_date	3_dur
1	6.5	0	3
2			
3			
4	6.5	0	1
5	6.5	0	1
6	6.5	0	3
7			
8			
9	6.5	0	3
10			
11			
12	6.5	0	1
13	6.5	0	2
14			
15	6.5	0	2
16			
17	6.5	0	1
18	6.5	0	2
19			
20	6.5	0	1
21	6.5	0	1

Track_2: Clear, Show ✓, 8

0	1_oscil	2_date	3_dur	4_pitch	5_amp	6_pan	7_wave	8_env
1	5.5	0	3	65.4	0.3	0.6	-5	8
2								
3								
4	5.5	0	1	45	0.3	0	-6	8
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								

Track_3: Clear, Show ☐, 8

0	1_supermod	2_date	3_dur
1	7.1	0	1
2	7.1	0	1
3	7.1	0	1
4	7.1	0	1
5	7.1	0	1
6	7.1	0	1
7	7.1	0	1
8	7.1	0	1
9	7.1	0	1
10	7.1	0	1
11	7.1	0	1
12	7.1	0	1
13	7.1	0	1
14	7.1	0	1
15	7.1	0	1
16	7.1	0	1
17	7.1	0	1
18	7.1	0	1
19	7.1	0	1
20	7.1	0	1
21	7.1	0	1

Fig. 1. jo_tracker version 2 (JPG).

Jo_tracker is a tracker interface for Csound. Inspired by Renoise, it's intention is to mix editing ease of tracker softwares with synthesis precision of Csound.

It can be used to generate some precise sound sequences and to write electronic music. Its first version was a Max MSP patcher. Though, the software span quickly required to be thought as an independant software, so it needed to be based on a real programming language. The second version was written with Lua (using IUP for user interface and Terra as a low level programming language for C libraries). This version is far more efficient, really quicker than the first one. Though, the codebase wasn't easy to read, and so, was hard to maintain. In order to distribute a clean version of jo_tracker and MIUP, third version of jo_tracker and second version of MIUP are both fully rewriten in C++, with improvements and some new features. It is a necessary work for further developments.

3.1 Base and principle

First requirement of jo_tracker was to provide a track system, with tracks able to manage an infinite number of parameters. With this feature, one line on a track is equivalent to one csound score "i" statement. This allows to combine or choose between very descriptive scores and algorithmic orchestras. Each track is shown as a spread sheet (See Fig. 1), where each column index corresponds to its Csound equivalent p-field. Eeach track can contain a different number of columns according to the needs of the instrument. Tracks number of lines and columns can vary between two sequences. Though, the number of lines in a sequence is the same for every track. In order to allow users to write some sequences, the tracker implements the notion of sequence : a sequence is equivalent to a time section in Csound. It's an abstract concept that can be thought as a time item containing score data. The main tab also contains one particular track : the tempo track. It can be used to do some tempo interpolation, or to instantaneously jump to another tempo. Obviously, jo_tracker provides some facilities such as copying any sequence's data to another sequence, cleare data, save a project (in a human readable text file), export a CSD file.

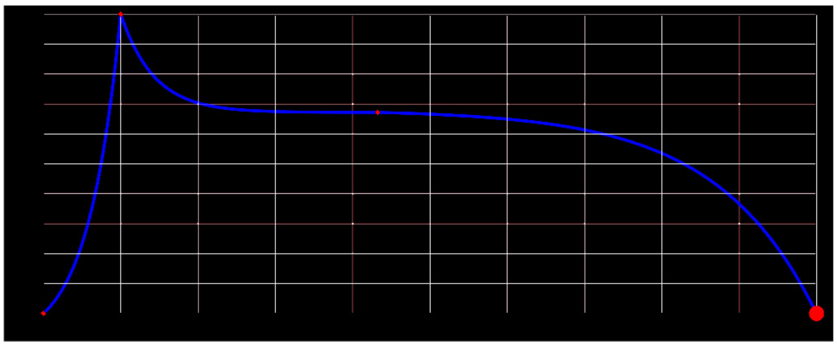


Fig. 2. Curve editor (JPG.

Second tab (See Fig. 2) is dedicated to GEN routines editing. It contains three major elements :
- A curve editor, allowing user to draw curves that will be translated in the GEN16 syntax.
- A waveform plot, mostly used to display waveforms from samples used as GEN01 data.
- Another spread sheet which function is to describe some other GEN routines data (simple arrays in GEN02, synthesis waveforms in GEN10).

There are three modes for starting Csound inside jo_tracker. Each mode generates a csound score, composed with one t statement (tempo track), multiple (many) i statements, and some f statements corresponding to GEN editors data. The main and first mode is activated by clicking the "Play" button. It triggers a call to Csound C API which starts a performance in realtime mode. The second mode (Record button) triggers a realtime recording of the current project. It is mostly useful if the orchestra is used with input channels or any realtime external device. It records the project into a stereo WAV file. The last mode calling Csound API acts as a non-realtime renderer, which also creates a stereo WAV file. It can be activated by clicking the "Render Stereo" item in "Files" menu.

3.2 New features, and upcoming improvements

Jo_tracker's third version brings a set of new features, allowing for a more flexible use. Though, since third version is still under development, some of these features are not ready yet.
-An embedded code editor for Csound orchestras. It allows user to write orchestas directly in the software. It provides syntax highlighting, autocompletion. It also calls the Csound API function `EvalCode` to check whether current instrument uses valid Csound syntax. On success, it registers the instrument in a database.
-Curve editor now supports spline mode (GEN08) and bezier quadratic curves (GENquadbezier)
-A new editor allowing user to manage sequences order and number of loops. As such, it can be considered as a meta editor allowing to manage the general shape of a composition.
-Record and render modes are now available for multichannel audio files
-Tracks are connected to a macro system, based on Csound powerful macro system.
-An audio device list is already implemented and will be added to the parameters menu

As future improvements, both MIUP and jo_tracker could benefit from Eric Wing's work [3] to port IUP on new platforms : MacOSX, Android, iOS, and Web browser. Since a lot of musicians use MacOSX as their composing environment, this target is considered as the major one.

4 The References Section

References

1. IUP Tecgraf Puc site, `https://www.tecgraf.puc-rio.br/iup/`
2. Csound Github site, `http://csound.github.io`
3. Eric Wing Github site, `https://github.com/ewmailing?tab=repositories`
4. cpp11nullptr Github site, `https://github.com/cpp11nullptr/lsignal`
5. Niels Lohmann Json project Github site, `https://github.com/nlohmann/json`