A TOOLKIT FOR MUSIC AND AUDIO ACTIVITIES ON THE XO COMPUTER

Victor Lazzarini
An Grúpa Theicneoláíocht Fuaim agus Ceoil Dhigitigh
National University of Ireland,
Maynooth, Ireland
Victor.Lazzarini@nuim.ie

ABSTRACT
This article describes the csndsugui toolkit. This software is designed for quick implementation of music and audio activities (applications) on the XO laptop, under the Sugar environment. The text introduces the platform, discussing its basic relevant attributes and the motivation for the toolkit. It follows on by discussing the basic components of the software, the Csound Python API and GTK/sugar. This is complemented by a discussion of the toolkit classes and their elements. The article concludes with some examples and an appreciation of further prospects for the system.

1. INTRODUCTION
The One Laptop Per Child (OLPC) initiative[3], founded on the principles of Constructionist Learning[7], aims to provide a platform for children’s education and development, the XO laptop (fig.1). One of the aspects of this platform is its support for media and, in particular, digital audio.

Some of the relevant XO laptop hardware specifications are[4]:
- CPU: x86-compatible (AMD Geode LX-700), with clock speed at 433 Mhz.
- 256MiB DRAM
- No magnetic (rotating) hard-drive. Instead, flash memory is used for mass storage (1024 MiB SLC NAND flash).
- Liquid-crystal display (7.5” dual-mode TFT)
- AC97-compatible audio, with internal stereo speakers and mono mic; mini jacks for input/output (Analog Devices AD1888 and Analog Devices SSM2211 for audio amplification)
- Video camera (colour, 640 by 480 pixels at 30 fps)
- Wireless networking, 802.11b/g interface. Includes support for 802.11s Mesh networking.
- Three USB-2.0 connectors.

In terms of CPU power and RAM, the laptop has quite a generous specification; its limitations are more in terms of mass storage space. As for software [5], the operating system is based on the Fedora Core distribution of Linux. A number of high-level programming environments are supported on the laptop: Python[8] (for application development), Javascript (browser scripting), Csound[1] (audio and music), Squeak (media authoring) and logo (education). The underlying graphics development libraries are based on the GIMP toolkit (GTK+, PyGTK).

The presence of Python, which forms the basis of the system’s graphical user interface (GUI) and application (“activity”) model and Csound provides a very powerful set of tools for development. The former is one of the most intuitive and flexible of the modern scripting languages and the latter arguably the most complete free-software audio processing system.

As part of the applications being developed for the XO, we find TamTam[6] a music exploration activity, developed by a group led by Jean Piche at the University of Montreal. It aims to be an easy-to-use tool for learning through music-making. TamTam uses Csound as its audio engine and provides a completely intuitive interface to it. Like all other activities, it is written in Python, using PyGTK as its GUI toolkit.

In this article, we will be presenting the csndsugui toolkit, which is another software component for music and audio on the XO laptop. This work has a different aim to TAM-TAM, which is to provide an easy-to-use and flexible support for the development of custom audio activities on the XO. The objective is to allow users to create their own small applications for dedicated purposes: for teachers to demonstrate certain principles, for older children to create their own computer instruments, etc..

The csndsugui toolkit has been developed to provide specialised support for the creation of Csound-based activities. Here, it is envisaged that the user would have some knowledge of both Csound and Python,
although not necessarily requiring fluency on both or any of these languages. The toolkit should provide a quick migration route from the FLTK widget opcodes, which integrate Csound5, as they will not be supported in the XO release of the language. Since many interesting Csound applications have been developed using these opcodes to implement their GUI, it will be quite useful to provide a route for porting these to the XO.

2. SYSTEM COMPONENTS

As discussed in the introduction, the toolkit is based on two components: Csound, accessed through its API, and Python (with GTK), for activity development. The basic GUI environment on the XO laptop is called Sugar (fig.2). Also written in Python, it provides the activity model for the system.

2.1. The Csound Python API

The Csound API is, in most part available for Python Scripting [2], with a few exceptions related to lower-level operations that are only suited to C/C++ programming. In general, the basic operations to control the Csound engine can be summarised as follows:

1. Csound instance creation:
   ```python
   from csnd import *
   cs = Csound()
   ```

2. Code compilation:
   ```python
   result = cs.Compile("mycsd.csd")
   ```

3. Performance:
   ```python
   perf = CsoundPerformanceThread(cs)
   perf.Play()
   ```

   This simple example demonstrates the spawning of a separate thread for performance, but it is possible to implement a single-threaded version, if needed. Once an instance of Csound is performing, events can be sent to it using, for instance,
   ```python
   perf.InputMessage(event_string)
   ```

   with which instrument instances can be started or stopped. In addition, communication between Csound and Python is enabled by the software bus, using
   ```python
   cs.SetChannel("channel_name", value)
   ```

   for sending control data in, and
   ```python
   value = cs.GetChannel("channel_name")
   ```

   These and a few other methods of the Csound class can be used to control Csound in a very flexible way. The csndsugui toolkit combines these facilities with the GUI elements provided by PyGTK and Sugar to allow for simple and transparent activity development.

2.2. GTK and Sugar

Sugar is the core of the XO laptop user interface. Applications running on it are called activities. From a programming point of view, they are based on subclases of the Activity class, which is itself derived from the gtk.Window class (as well as gtk.Container). So it provides access to GTK graphics and widgets, which can be then integrated into an application. In order to create such application, the user will provide a class of his/her own, derived from sugar.activity.Activity. Such user-defined classes are then instantiated by sugar when an Activity is launched. In addition, Activities will in general also require a toolbox to be instantiated, so that default controls, such as close, minimise, etc., can be easily accessed. The application itself is a bundle of several files, including graphics resources, inventories, etc., as well as the user supplied Python script(s).

![Figure 2. The Sugar desktop environment.](image)

The programming of Activity subclasses can be quite complex, as there is extensive support for a variety of components. The principle of the csndsugui toolkit is to allow all this complexity to be bypassed. The user will only need to derive a bare-bones Activity class and then use the toolkit to build the interface to his/her Csound code.

3. THE CSNDSUGUI TOOLKIT

The csndsugui toolkit is founded on a base class, BasicGUIL which implements all the graphics, widget support and the basic communication mechanism. From this is derived the CsndGUI class, which implements all the Csound engine control. The BasicGUIL class holds an externally-created Activity object that is its main window. It creates the standard Activity toolbox and any widgets that the user defines. Currently it supports the following GTK widgets:

- Sliders (and slider banks)
- On/off buttons (and button banks)
- Message buttons
• Spin buttons
• File choosers
• Frames
• Boxes
• Text labels

More widgets can be added, provided they are implemented in PyGTK.

BasicGUI user methods:

button(box, title): creates an ON/OFF button
mbutton(box, message, title): creates a message button
box(vert, parent, padding): creates a box (vert. or horiz.)
filechooser(box, title): filechooser button
slider(init, start, end, x, y, box, title, vert, linear): creates a linear or exponential slider (vert. or horiz.)
spin(init, start, step, page, box, accel, title): creates a spin button
text(name, box, colour): creates a text label
framebox(name, vert, parent, colour, padding): creates a frame box (vert. or horiz.)
vsliderbank(items, init, start, end, x, y, box): creates a vertical slider bank
hsliderbank(items, init, start, end, x, y, box): creates a horizontal slider bank
buttonbank(self, items, box): creates a button bank

__init__(act, colour, vert): creates a BasicGUI object based on Activity instance act

The key aspect of the BasicGUI class implementation is that it provides an automatic generation/assignment of software bus channels. These carry the name label given to the widget and can be easily referenced in Csound code. They can also be assigned default names, such as BN, for a button, where N is the button number (in order of creation). All the communication calls are hidden away from the user, who only needs to create the widgets and use the control channels in his/her Csound instruments.

The BasicGUI class does not know anything about Csound, so it requires that all of its functionality be overridden. This is done in the CsndGUI class, which implements the calls to the Csound API. A CsndGUI object will contain instances of Csound and CsoundPerformanceThread. It provides some basic methods for Csound operations, such as ‘play’, ‘pause’, etc..

CsndGUI user methods:

csd(name): sets the source CSD (unified Csound code file) and compiles it
play(): starts a Csound performance
pause(): pauses performance
reset(): resets Csound, ready for a new CSD
set_channel(chan, val): set a value on a control channel, generally not used directly.
set_filechannel(chan, name): sets a string on a file channel, generally not used directly
set_message(msg): sets a message (RT event), generally not used directly (accessed through the GUI widgets)

__init__(act, colour, vert): creates a CsndGUI (and therefore a BasicGUI) object based on Activity instance act

Typically, users will never create a BasicGUI object directly, but will instead instantiate a CsndGUI. Once this is done, a CSD can be passed to it and the performance can be initiated using the play() method. The next section demonstrates some typical uses of the toolkit.

4. ACTIVITY EXAMPLE

A cndsugui-based activity will be based on two user-written components: a python script, where the GUI is defined and an unified Csound code file (CSD), which is an XML file containing the Csound instruments and any relevant score events. Depending on the controls set up in the Python script, a number of channels will be available to Csound instruments. These can be used to control any parameters of the synthesis/processing operation. As mentioned above, the channels will be identified by their widget labels, so they can be easily referenced.

The simple example shown here creates 8 parallel oscillators with independent on/off and frequency controls. A button bank is used for the oscillator switches and sliders for frequency adjustment. The activity script is shown below. It defines a subclass of activity.Activity (from the sugar package), which, as mentioned above, is derived from gtk.Window. Only the __init__() method of that class is required. A m CsoundGUI object is created as a class member and passed the class reference. Then the CSD for the application is set, followed by the creation of all required widgets. The code is completed by starting the Csound engine, which will be automatically stopped on exit.

import cndsugui from sugar.activity import activity

# the activity (sub-)class
class Waves(activity.Activity):
def __init__(self, handle):
    activity.Activity.__init__(self, handle)

    # colours
    red = (0xFF00, 0, 0)
    bg = (0xFFFF, 0xF000, 0xFFFF)
    app = cndsugui.CsoundGUI(self, bg)
    app.csd("waves.csd")
    app.text("Making Waves")

    # bounding boxes for widgets
    sfbox = app.box(False)
    bfbox = app.box(False)

    # boxes for sliders and buttons
    sbox = app.framebox("frequencies", False, sfbox, red, 40)
    bbox = app.framebox("oscillators", False, sfbox, red, 40)

    # widgets
The Csound code, as defined in the CSD file will then define all the instruments, using the controls provided in the Python script. The channels defined in there have the default names “BN” for buttons (on/off, 0/1) and “SN” for sliders. Channels are exported to global variables for ease of use:

```csound
/* gkbN holds the (0 or 1) values of each button in the bank (B1-8) */
gkb1 chnexport "B1", 1
...
```

Two instruments are used: the first to trigger instances of the second, which contains the sound synthesis code. The triggering is done by checking the values of the button channels B1 to B8:

```csound
instr 1
kl init 1
.../* if gkb1 changes to 1 then the instance 1 of instr 11 is triggered */
if gkb1 == 1 then
if kl == 1 then
event "i", 11.1, 0, -1, 1
kl=0
endif
/* if gkb1 changes to 0 then the instance 1 of instr 11 is killed */
else
if kl == 0 then
event "i", -11.1, 0, -1,1
kl=1
endif
endif
/* and so on for the other 7 buttons */
...
endin
```

The synthesis instrument looks up the value of the S1 to S8 channels to get its frequency and the “main_volume” channel (spin widget) to get its amplitude:

```csound
instr 11
/* p4 is instance ID */
s1 sprintf "$%d", p4
kl chnget s1
k2 chnget "main_volume"
kv tonek k2/10, 10
ka linerr 2000*kv, 0.1, 0.1, 0.05
a1 oscili ka, k1, 1
outs a1,a1
endin
```

One instance of instr 1 runs for the duration of the program and up to 8 instances of instr 2 can be triggered at different times by using the button controls.

As mentioned before, Activities are bundles containing the Python scripts plus resource files. The CSD file is placed in the activity bundle (its top directory). Other important files are:

- **setup.py**: used to create the activity and contains a call to bundlebuilder.start() with the name of the Activity subclass.
- **activity/activity.info**: contains all the required information about the activity (name, icon file, class name, etc).

The process of putting together an activity bundle is quite simple, once all the required Python and Csound code are written. It is well documented by the OLPC development team.

### 5. FURTHER PROSPECTS

The csndsugui is in its version 1, providing a fully-functional software, with features that make it a substitute to the Csound FLTK widget set, plus some extras such as the file chooser control. In its envisaged that most work will now be concentrated in integrating it further with Sugar, in order to take advantage of its facilities for GUI development. Some extra work on documentation, mainly in the development of tutorials for target users, will be necessary. The latest versions of the toolkit, reference and example applications can be found in:

http://dev.laptop.org/git?p=activities/csndsugui

### 6. REFERENCES


