Python Scripting in QuteCsound

Andrés Cabrera

mantaraya36@gmail.com
Python in QuteCsound

- Running a python file in python externally
- On the internal interpreter:
  - Through the interactive console
  - Evaluating code from any editor or the Python scratch pad.

Whole scripts can be evaluated quickly through the Scripts menu.

Sections can be defined with `##`
Currently a single instance, so all variables, imports, etc. are accessible everywhere.

Implemented through the PythonQt library:

- Adds dynamic exposing of C++ objects into python.
- Wraps all of Qt in Python (no need for pyQt or PySide)
Building support for PythonQt

- Download and build PythonQt (pythonqt.sourceforge.net)
- Build PythonQt make sure you match debug and release builds with QuteCsound
- Add CONFIG+=pythonqt to QuteCsound's qmake line
Elements of QuteCsound and Csound are exposed through a custom object which is available by default in the interpreter called "q".

This object wraps functionality from the QuteCsound interface, widgets, Code editors and running Csound instances.
QuteCsound interface

play(int index = -1, realtime = true)
pause(int index = -1)
stop(int index = -1)
stopAll()

Giving an index of -1 will affect the currently selected tab.
QuteCsound tabs

setDocument(index)
getDocument(name = "")

To set the active document or get a document index.
QuteCsound editors (getters)

getSelectedText(index = -1, int section = -1)
getCsd(index = -1)
getFullText(index = -1)
getOrc(index = -1)
getSco(index = -1)
getWidgetsText(index = -1)
getSelectedWidgetsText(int index = -1)
getPresetsText(index = -1)
getOptionsText(index = -1)
An index of -1 will affect the currently active tab.
QuteCsound editors

insertText(text, index = -1, section = -1)
setCsd(text, index = -1);
setFullText(text, index = -1)
setOrc(text, index = -1)
setSco(text, index = -1)

setOptionsText(text, index = -1) # unfinished
setWidgetsText(text, index = -1) # unfinished
setPresetsText(text, index = -1) # unfinished
QuteCsoud editors

Since all the text passes through the standard interfaces, all changes to the text editors are put in the undo/redo stack.
getFileName(index = -1)
getFilePath(index = -1)
setChannelValue(channel, value, index = -1)
getChannelValue(channel, index = -1)
setChannelString(channel, stringvalue, int index = -1)
getChannelString(channel, index = -1)
setWidgetProperty(channel, property, value, index = -1)
getWidgetProperty(channel, property, index = -1)
Widgets - interaction

Note that interaction is with widgets, not Csound, so be sure to connect Csound to widgets for this to work!
Widgets - creation

createNewLabel(int x, int y, int index = -1)
createNewDisplay(int x, int y, int index = -1)
createNewScrollNumber(int x, int y, int index = -1)
createNewLineEdit(int x, int y, int index = -1)
createNewSpinBox(int x, int y, int index = -1)
createNewSlider(int x, int y, int index = -1)
createNewButton(int x, int y, int index = -1)
createNewKnob(int x, int y, int index = -1)
createNewCheckBox(int x, int y, int index = -1)
createNewMenu(int x, int y, int index = -1)
createNewMeter(int x, int y, int index = -1)
createNewConsole(int x, int y, int index = -1)
createNewGraph(int x, int y, int index = -1)
createNewScope(int x, int y, int index = -1)
All widget creation functions return the widgets internal unique id (a string).

The Uuid string can be used instead of a channel name in the widget setter functions.
Csound instance

g版本() # QuteCsound API version
gSampleRate(int index)
gKsmps(int index)
gNumChannels(int index)

Last three return -1 if Csound is not running
Opcode queries

opcodeExists(QString opcodeName)

Not drawn from Csound but from the documentation. Should be in sync.

Designed for code generation/syntax highlighting.
Score Events

sendEvent(int index, QString events)

sendEvent(QString events)

Can be used to send score events to any running instance/tab.
registerProcessCallback(string func, int skipPeriods = 0);

Through the QuteCsound API, a Python function can be registered as a callback. It will be called after processing each ksmps block (with optional skipping of a number of periods)
The hard stuff

... i.e. What is not done yet
Interaction with f-tables

`getTablePointer(fn, index = -1)`

`copyTableToList(fn, index = -1, offset = 0, number = -1)`

`copyListToTable(list, fn, index = -1, offset = 0)`
Interaction with f-tables

You can't access f-tables while Csound is processing its buffer!

These functions must work through a messaging system that sends requests to the performance thread.

The functions will wait for the realtime thread, to read the data and return.

This is what SuperCollider does and some of this has already implemented in the csPerfThread C++ class.
Interaction with Live Event Sheets

QuteSheet* getSheet(int index = -1, int sheetIndex = -1)
QuteSheet* getSheet(int index, QString sheetName)

It would be great to have a data structure representing Live Event Sheets.

That way Sheets could be used as visual matrices to store information.
Some ideas to use it for

- Sequencer and notation applications
- Interactive pieces (e.g. Koenig realization)
- Automatic code generation/visualization:
  - Graphical F-table editor
  - Visual patching interfaces
- Design of custom control GUIs and widgets
- Remote control on mobile devices
  - Send the widgets and do the connections automatically
That's it

What would YOU do with it?

Questions/Suggestions?