Lilt2/Esci Elengí (*Subclassing is No Good*)

Ioannis Zannos

18 September 2012

About

This library contains SuperCollider tools and examples coded and collected since 2003 by Ioannis Zannos, organized using the RepQuark by Martin Carlé. It is organized in Quarks, and includes a utility for installing those Quarks from a menu, which is installed by putting an alias of the folder named "PutMyAliasInExtensions" into the user's SuperCollider/Extensions folder.

New - Suggested

See: iz.core folder, Snippets

- AppModel, AppModel2: Interconnect GUI elements between themselves and to NodeProxy objects, without having to save GUI elements to variables. Also add MIDI and OSC funcs easily, and enable / disable input from these in groups when a window containing those widgets comes to the foreground.
- Code, ProxyCode: Navigate, select and run code snippets between comments. Evaluate them in ProxySpace, and add basic control of the resulting NodeProxies from the keyboard.
- ProxyCodeEditor, ProxyCodeMixer, ProxyCodeMixer3: Edit code of NodeProxies created from ProxyCode, navigate the history of all source code snippets for each proxy, automatically parse, create menus and map control parameters for each proxy to MIDI-mappable sliders and knobs. Auto-insert variable declaration code for using loaded buffers in NodeProxy source functions.
- BufferListGui: GUI for managing lists of buffers, auto-saved as sctxar. Includes preview with play and with user custom code. Loaded buffers will automatically re-load when the server re-boots. Loaded buffers are available via Library.at('Buffers') for ProxyCodeEditor to insert in NodeProxy source code.

Download

Download from: https://github.com/iani/SC or:

```
git clone git://github.com/iani/SC.git
```

Credits

- Code, Resource and Chain by Ioannis Zannos, March-May 2011
- Widget, ProxyCode, ProxySourceEditor by Ioannis Zannos, May-August 2012
- Quarks modularisation scheme by Martin Carlé, September-October 2011
- Server GUI by Sergio Luque
- beastmulch quark containing BEASTmulch UGens by Scott Wilson and the BEAST research team at the University of Birmingham.
  See: http://www.birmingham.ac.uk/facilities/BEAST/research/mulch.aspx
- collins quarks ported from Nick Collins' work at: http://www.sussex.ac.uk/Users/nc81/code.html
- SC3PlugIns by the SuperCollider community, re-organized here as separate (Rep-)Quarks.
- Collection of older code and notes by Ioannis Zannos.

Copying (License)

Installation

As of 2011-06-08 The library is being reorganized to a modular plugin-form using quarks (thanks MC). To make these quarks available via a menu on MacOS X, make an alias of the folder "PutMyAliasInExtensions" and put the alias in:

```
~/Library/Application Support/SuperCollider/Extensions/
```

Then recompile SuperCollider.
Choose iz.local from the Quarks menu. Recommended Quarks to try out are:

- LiltBase (Provide utility classes for other Quark modules)
- Snippets (Place document windows conveniently and provide navigation and code execution utilities)
- ServerPrep (Handle loading of Buffers, SynthDefs and automate booting of Server before starting Synths)
- Resource (Bind Synths and other objects to symbols and make them available through the symbols)

- ServerGui (Alternative, more compact GUI for the Servers, by Sergio Luque)
- Chains (Alternative score-writing class for scheduling execution of functions with patterns)

New Application Framework: AppModel

AppModel, AppModelVersion2, Value, Widget

These are currently in quark iz.core/Snippets.

Simplify the task of creating guis, connecting views to values by name and adding actions and update messages for each view. Also simplify the tasks of adding MIDI and OSC to each object in the application, of re-opening closed windows (see AppStickyWindow, AppModel:stickyWindow), and of automatically switching MIDI control to the foremost window.

A first prototype is tested with classes AppModel, Adapter, and the subclasses of AppNamelessView. Example applications are built in BufferListGui, ProxyCodeEditor, ProxyCodeMixer, ProxyCodeMixer3.

A second version of the above is given in classes AppModel2, Value, Widget, NumberAdapter, SpecAdapter2, ListAdapter2. This will gradually become the application framework for all applications in Lilt2, as it substantially simplifies coding. See first examples in folder AppModelVersion2/Examples.

NotificationCenter additions: Flexible messaging with cleanup ”objectClosed”

Simplify the connection of objects for sending messages to each other via NotificationCenter. Automate the creation of mutual NotificationCenter registrations to messages, and their removal when an object receives the message objectClosed. This makes it easier to establish messaging between objects in the manner of the Observer pattern exemplified by classes Model and SimpleController, while shotening and clarifying the code required to use NotificationCenter.

The advantage gained is that it is no longer needed to check whether an object stored in a variable is nil in order to decide whether to send it a message. One can create messaging interconnections between objects without storing one in a variable of the other, and one can safely send a message to an object before it is created or after it is no longer a valid receiver of that message. Notification connections can be removed by method objectClosed, which can be called when a view or other dependent object closes.

Class Code

Enable the selection of parts of a SuperCollider document separated by comments followed by :, the movement between such parts, and the execution of those parts through keyboard shortcuts. Additionally, wrap these code parts in a routine so that number.wait messages can be written straight in the code, without wrapping them in { }.fork or Routine({ }).

Also ensure that the code will run after the default server is booted and the Buffers and SynthDefs defined as Udefs in a Session have been loaded.

Shortcuts provided are:

- Command-shift-x: Evaluate the code in an AppClock routine. Booting the default server if needed
- Command-shift-alt-x: Evaluate the code in a SystemClock routine Boot default server if needed
- Command-shift-v: Evaluate and post the results of the code, without routine or server booting
- Command-shift-j: Select the next code part
- Command-shift-k: Select the previous code part
- Command-shift-}: open a list of the code segments of the current Document
- Command-alt-shift-}: open a widow with buttons for running the code segments of the current Document
- Command-alt-control-shift-}: Create OSCresponders for running the code segments of the current Document

Class CodeProxy

Evaluate code snippets in a Document using Code-keyboard shortcuts in a ProxySpace, and create NodeProxies from the comments at the beginning of each Snippet. Parse additional argument specifications from the comments. Provide essential play-stop and volume increase-decrease commands as keyboard shortcuts.

Class ProxySourceEditor

Edit the source code of a NodeProxy created from a snippet, provide controls for its arguments automatically, and browse the history of source code snippets for this proxy. Also provide MIDI bindings for each proxy parameter control gui item.

Class Panes

Arrange Document windows on the screen conveniently for maximum view area on the screen. Provide 2 layouts: single pane and 2 panes side by side, with keyboard shortcuts for switching between them. Provide an auto-updating document list palette for selecting documents by mouse or by string search. Provide a way for switching between a dark colored document theme and the default document theme via keyboard shortcuts, with automatic updating of the coloring of all relevant documents.

Class Dock

Provide some useful shortcuts for common tasks: browseUserClasses : Open a list of all classes defined in the user's Application Support directory. Typing return on a selected item opens the code file with the definition of this class.

insertClassHelpTemplate : Insert a template for documenting a class named after the name of the document. Inserts listings of superclasses, class and instance variables and methods.

openCreateHelpFile : Open a help file for a selected user class. Automatic creation of the file is reserved to code residing outside the distribution files of this library.

showDocListWindow : An auto-updating window listing all open Documents, with selection by mouse click or by text search.

closeDocListWindow : Close the document list window

Class Chain, EventStream, Function:sched and Function:stream

Simplify the creation and access of Streams from Patterns and their use with Routines and Functions scheduled for repeated execution.

Example: Simplify the above code even further, while enabling control of dtime (and any other parameters) via patterns:

```
(
{   // Symbol:stream creates and / or accesses the stream as appropriate:
   \default.mplay([\freq, \freq.prand((25..50), inf).midicps])
      .dur(0.1, exprand(0.01, 1.0));
   // play 20 events only
   \duration.stream(Prand([0.1, 0.2], 20));
}.stream;
)
```

Note: symbol.stream(Prand(…)) is equivalent to symbol.prand(…)

Also chain timed sequential execution of functions, with sound or not, in a manner more direct than Pbind.

```
(
//:3 different synth functions sharing patterns.
Chain(Pseq([
     { \default.play([\amp, 0.05, \freq, ~freq.next]).dur(~dur2.next, ~fade.next); },
     { { Resonz.ar(WhiteNoise.ar(2.5), \freq.n.dup, 0.01) }.play.dur(\dur2.n, \fade.n); },
     { { SinOsc.ar(\freq.n.dup / 2, 0, 0.07) }.play.dur(\dur2.n, \fade.n); },
], 20),
() make: {    // store shared patterns in the global environment of the Chain:
        \dur2.pseq([0.1, 0.2], inf);
        \fade.pseq([0.1, 0.2, 1], inf);
```

```
        \freq.pseq([80, 85, 87, 90, 92].midicps, inf)
});
//: ---
)
```

Other example:

```
(
//:Example combining a single synth and a chain of synths.
Chain(Prand([ // choose from the following at random:
        {       // Play a series of events
           \default.mplay([\freq, (50..80).choose.midicps]).dur(0.03, exprand(0.01, 0.3));
           // The number and timing of the events is defined through arguments to the chain me
        }.chain({ Prand([0.06, 0.07, 0.14], 10 rrand: 20) }),
        {       // Play a single synth.
                { | freq = 400 | SinOsc.ar(freq * [1, 1.2], 0, 0.02) }
                        .play(args: [\freq,  \freq.pseries(4).next * 100])
                         .dur(0.1 rrand: 1, 0.5 rrand: 2.5)
        }
], 30
));
//: ---
)
```

Class ServerPrep

- Obviate the need to boot the server manually before starting synths.
- Ensure that Buffers and SynthDefs are allocated / sent to the server before starting synths, efficiently.
- Provide a safe way for registering synth and routine processes to start automatically when the server boots or when the tree is inited, ensuring that SynthDefs and Buffers will be loaded first.

Classes involved:

- ServerPrep
- ServerActionLoader
- SynthLoader
- DefLoader
- BufLoader

- RoutineLoader
- UniqueBuffer
- Udef

Class SynthResource

Simplify the creation and control of Synths by storing them in a dictionary for later access, and by providing utility methods for controlling the duration and release time, for synchronizing the execution and life time of routines pertaining to a synth, and for attaching other objects that react to the start and end of a synth.

Example of how SynthResource can simplify the code required:

Without Symbol:mplay

```
(
{
   loop {
      {    var synth;
         synth = Synth(\default, [\freq, (25..50).choose.midicps]);
         0.1.wait;
         synth.release(exprand(0.01, 1.0));
      }.fork;
      [0.1, 0.2].choose.wait;
   };
}.fork;
)
```

Using Symbol:mplay

```
(
{
   loop {
      \default.mplay([\freq, (25..50).choose.midicps])
         .dur(0.1, exprand(0.01, 1.0));
      [0.1, 0.2].choose.wait;
   };
}.fork;
)
```

Class Spectrograph

An example application showing some of the features of this library. Creates a window showing a live running spectrogram of one of the audio channels. The fft polling process for the spectrogram

is persistent, that is, it starts as soon as the server boots and re-starts if the server's processes are killed by Command-. It (optionally) stops when the Spectrograph window is closed.

This class was inspired by the Spectrogram Quark by Thor Magnusson and Dan Stowell, and is a rewrite to show how the code can be made clearer (and the behavior safer and more consistent regarding boot/quit of the server and open/close of the spectrogram window).

Note: The Spectrograph may occasionally crash SuperCollider if it is running on a MacBook with battery power. I have not been able to trace the source of the problem so far but suspect this is due to fast Image updates causing problems with the Graphics Card.