blue
documentation
version 0.89.6 – 2002.9.3

steven yi

email: stevenyi@hellokitty.com
web: http://www.kunstmusik.com

about blue

blue is an object composition environment for use with Csound. blue interacts with csound by generating .CSD files, which it then feeds to csound for compilation. any version of csound that can be called by commandline is able to be used with blue.

one of blue's main features is the graphical timeline of soundObjects, of which a special soundObject—the polyObject—is able to group other soundObjects. polyObjects feature their own timeline, allowing for timelines within timelines. soundObjects can be anything that generates notes, be it a block of csound SCO, a script in python, or a graphical object. (at the time of this writing, there are no graphical soundObjects currently made).

soundObjects may further have noteProcessors added to them, which may do things like “add .3 to all p4 values” or “of these notes, only generate the first three”. noteProcessors are especially useful when used with instances from the soundObject Library.

the soundObject library allows for working with soundObjects by making *instances* of a soundObject. *instances* point to a soundObject, and when they generate notes, they use the soundObject they are pointing to to generate the notes for them. *instances* however can have their own properties which they will apply to the generated notes, so that they can be of different duration and have different start times, as well as have different noteProcessors used with them. the advante of *instances* versus manually copying a soundObject over and over again is that if you want to change the notes for all of these soundObjects, you'd have to manually go and change all of them, while with *instances*, you'd only have to change the one soundObject all of the instances are pointing to. (which also means you could make a song template file, all pointing to empty soundObjects, build the form of the piece, then change the soundObject's note material to “fill in the form”.)

other features include the orchestra manager, which allows for importing of instruments from .CSD files, a list to manage instruments, and the ability to selectively generate instruments when compiling .CSD files. instruments can be either a single csound instrument or may be a GUI instrument. (at the time of this writing, there are no graphical instruments currently made).

personal notes

blue is my tool for composition with csound. I made it originally for its ability to organize soundObjects in time, but also to organize and automate the more mundane tasks that I've found in working with Csound. it's here to make working with csound a more enjoyable and powerful experience.

as the program has developed over time i am finding more ways to make the user experience intuitive and faster to work with. and always more possibilities. the program's architecture was designed to be as open as possible, and i'm finding earlier decisions have paid off, as i can build new soundObjects or GUI instruments to explore ideas with relative ease. and for me especially, i can spend less time thinking about how i'm going to make music and just enjoy actually making music.

pleas feel free to email me with feedback, or join the blue mailing list. information is available at my website:

http://www.kunstmusik.com

and thanks to those people who have already given feedback. it's been great hearing from you!

Installing blue

1. Checking for a compatible Java JVM installed

blue requires a Java 1.3 or greater JVM (Java Virtual Machine). To test to see if you have a JVM installed and whate version, at a command prompt type "java –version". If you see something along the lines of "command not found" then you need to install a Java Virtual Machine.

A Java JVM for Linux, Windows, and Solaris can be found at:

http://java.sun.com/j2se/downloads.html

From here, select the version of Java you want to use (must be 1.3 or higher for blue), and then choose the download for your platform. There will be an option to download either the "JRE" or the "SDK". The JRE is just what is required to run java programs and is all that is required to run blue. If you're planning to do java development, then download the SDK, as it includes the JRE, but also all the development tools.

For Apple OSX, a Java 1.3.1 JVM should be installed with the system. If you don't have it, you can find it at:

http://developer.apple.com/java/

Unfortunately, there is no Java 1.3 compatible JVM for OS9.

2. Installing blue

After downloading the latest blue, please be sure to unzip with directory names intact. Some of the locations of files are important for blue to find files.

After unzipping, you should find a set of directories labeled "bin", "doc", "lib", "conf", and "work".

Included in the bin directory are a Windows batch file(blue.bat) as well as a Linux/OSX shell script(blue.sh). Click on the apropos file to start blue.

3. Known Platforms

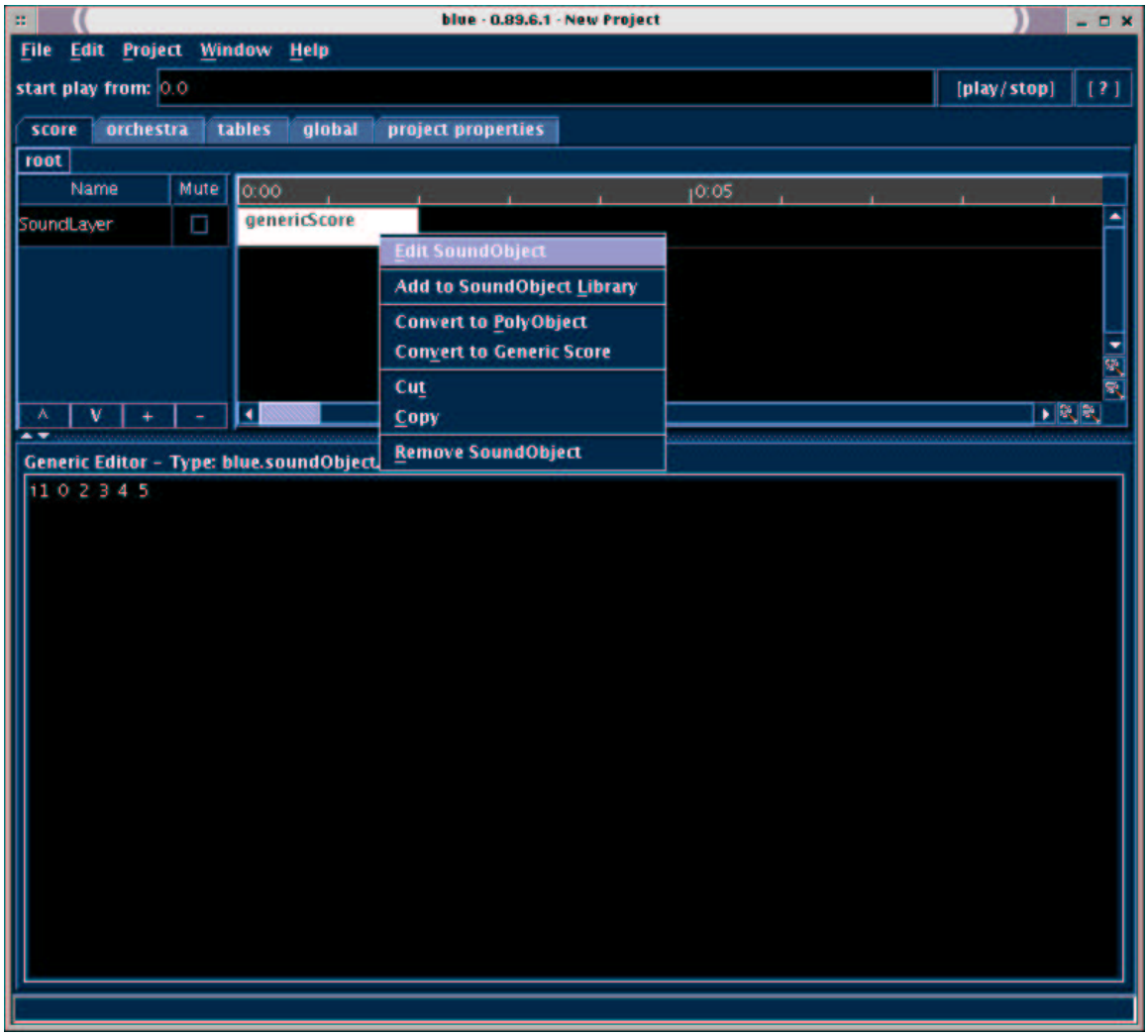
-Win98SE, Sun Java 1.3.1 JVM,
-Mandrake 8.0 GNU/Linux, IBM Java 1.3 JVM

-Mandrake 8.2 GNU/Linux, Sun Java 1.4.0 JVM
-Apple OSX, Apple 1.3.1 JVM

Sun's JVM 1.3.1 on Linux does NOT work well with blue, and I'd recommend against using it on Linux.

Using blue

the score edittor



PLAY BAR
The play bar at the top has a field for setting what time you want the score to play from, a play/stop button to start and stop playing, and a help button. When the play/stop button is used, blue will generate a .CSD file from the current work file, starting from the play time given in the field, and will use the command line given in the project properties tab to use to play the .CSD file.

TABS
the tabs below the play bar are used to switch to the different editors and managers in blue.

POLYOBJECT BAR

the poly object bar (shown above with only one polyObject, “root”) shows what polyObject you are currently editting. if you were to add a polyObject named “phrase 1” to the main timeline shown above, then double click that polyObject to edit it, the polyObject bar would have two buttons on it, one for ”root”, and one for “phrase 1”. you would then be editting “phrase 1”'s timeline. by clicking on the “root” button of the timeline, you would then return out of the polyObject's timeline and back in the root's timeline.

SOUNDLAYER EDITTOR

below the polyObject bar on the left, you will see the soundLayer edittor. here you can change the name of the soundLayer, as well as mute the layer (all soundObject's on muted layers will not be used to generate notes when creating .CSD files).

on the bottom of the soundLayer edittor are four buttons, “^”, “V“, “+”, and “-”. “^” and “V” will push up or push down soundLayers. (HINT: You can move multiple soundLayers by clicking on one soundLayer, then holding down shift and clicking on the last of the soundLayers you want to move, then using the “^” and “V” buttons.) the “+” will add a soundLayer after the currently selected soundLayer. if no soundLayers are selected, then it will add one to the end of the list. the “-” button will remove any selected soundLayers. it should ask for a confirmation before removing any layers.

THE TIMELINE

below the polyObject bar on the right is the main time line. it shows the time line for the currently editted polyObject. the magnifying glasses next to the scrollbars are used to zoom in on the time line, and those settings will be maintained between work sessions.

On the time line, rt-clicking on any soundLayer will show menu options for adding different types of soundObjects, or pasting a soundObject from the buffer if any soundObjects in the buffer are available. if you are using the soundObject library and use the “copy instance” button, it copies it to the soundObject buffer for use with pasting into the time line via this menu.

Once you have soundObjects on the time line, you can click on one to select it and see its editor below the time line, then click on the soundObject and drag the mouse to move the object around in time or to a different layer. Clicking and dragging near the right edge of the soundObject will allow you to change the duration of the soundObject.

You can also select multiple soundObjects by holding down shift and clicking multiple soundObjects. holding down shift and dragging one soundObject will move all of the selected soundObjects.

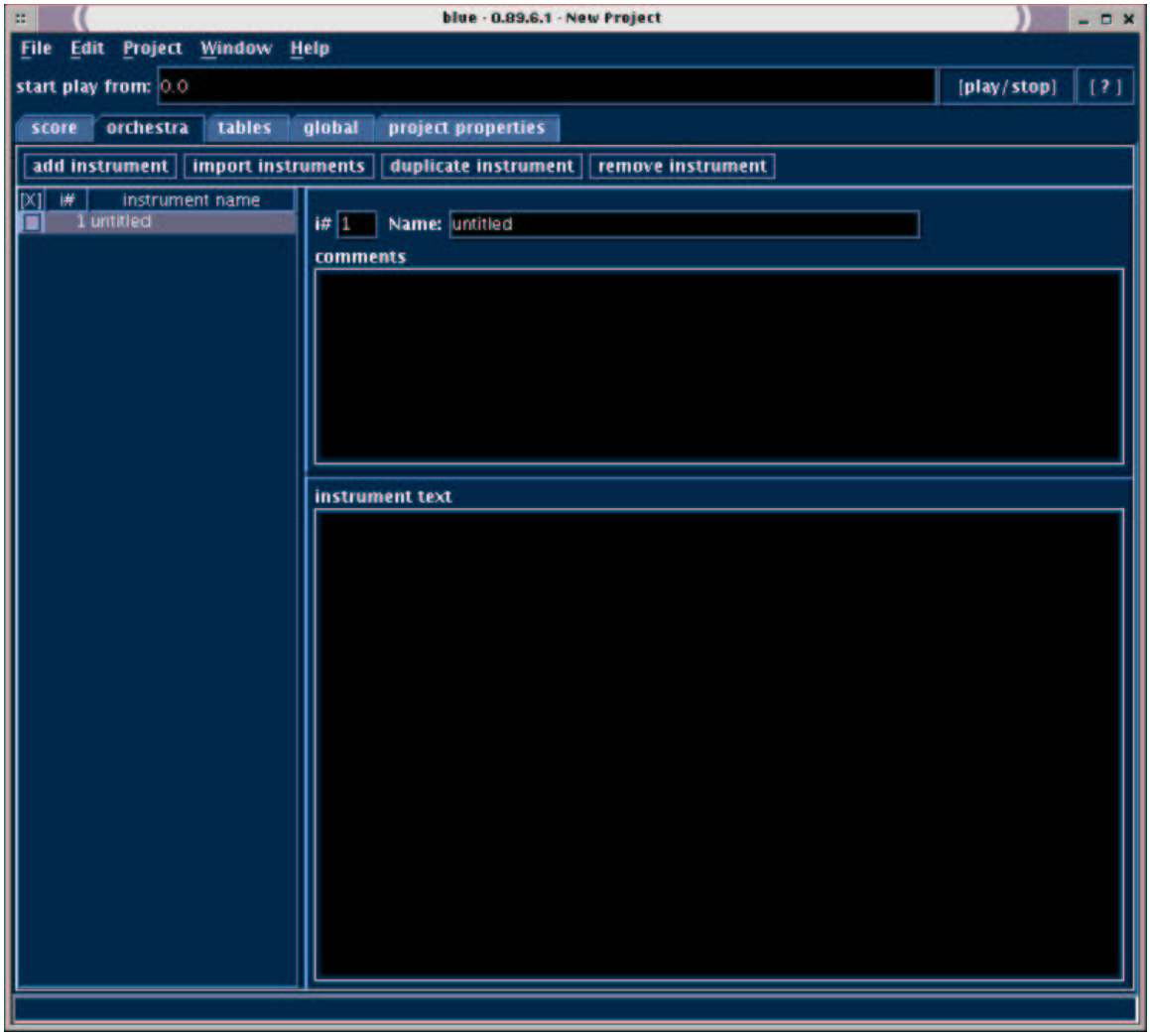
If you have a soundObject selected, you can edit the properties of the soundObject also by using the soundObject property dialog (which can be opened from the menu "Windows -> Sound Object Property Dialog" or by using the shortcut "F3". From this dialog you can change the name of the soundObject, it's start time and subjective duration, and also add and remove noteProcessors (if the soundObject supports it).

Rt-clicking on a soundObject pops open a menu (shown in the picture above) which will let you add a soundObject to the soundObject library(only the one clicked on), convert soundObjects into a polyObject(any selected soundObjects), convert a soundObject into a genericScore soundObject, or cut/paste/remove. Converting to a polyObject is useful if you're editting on a time line and decide you'd like to group the soundObjects together to handle them as a single soundObject. Converting to a genericScore is useful if you're working with a soundObject that isn't a genericScore and want to start editting it as a genericScore.

SHORTCUTS FOR THE TIMELINE

ctrl-c	- copy selected soundObject(s) (if multiple are selected, it will be pasted in as a polyObject)
ctrl-x	- cut soundObject(s) (if multiple are selected, it will be pasted in as a polyObject)
ctrl-click	- paste soundObject from buffer where clicked
shift-click	- when selecting soundObjects, this allows you to select multiple

orchestra manager



INSTRUMENT BUTTONS
The top row of buttons are *add instrument*, *import instrument*, *duplicate instrument*, and *remove instrument*.

Add Instrument will pop open a menu to select what kind of instrument to insert. Currently the only available intstrument is the "genericInstrument", though future GUI based instruments are planned to be added.

Import Instrument opens up a file dialog and allows you to choose a .csd or .orc file to import from. This will only import anything within "instr" and "endin", and will skip over any global variables.

Duplicate Instrument will duplicate whatever the current selected instrument is.

Remove Instrument will remove the currently selected instrument. Alternatively, you can also press the delete key.

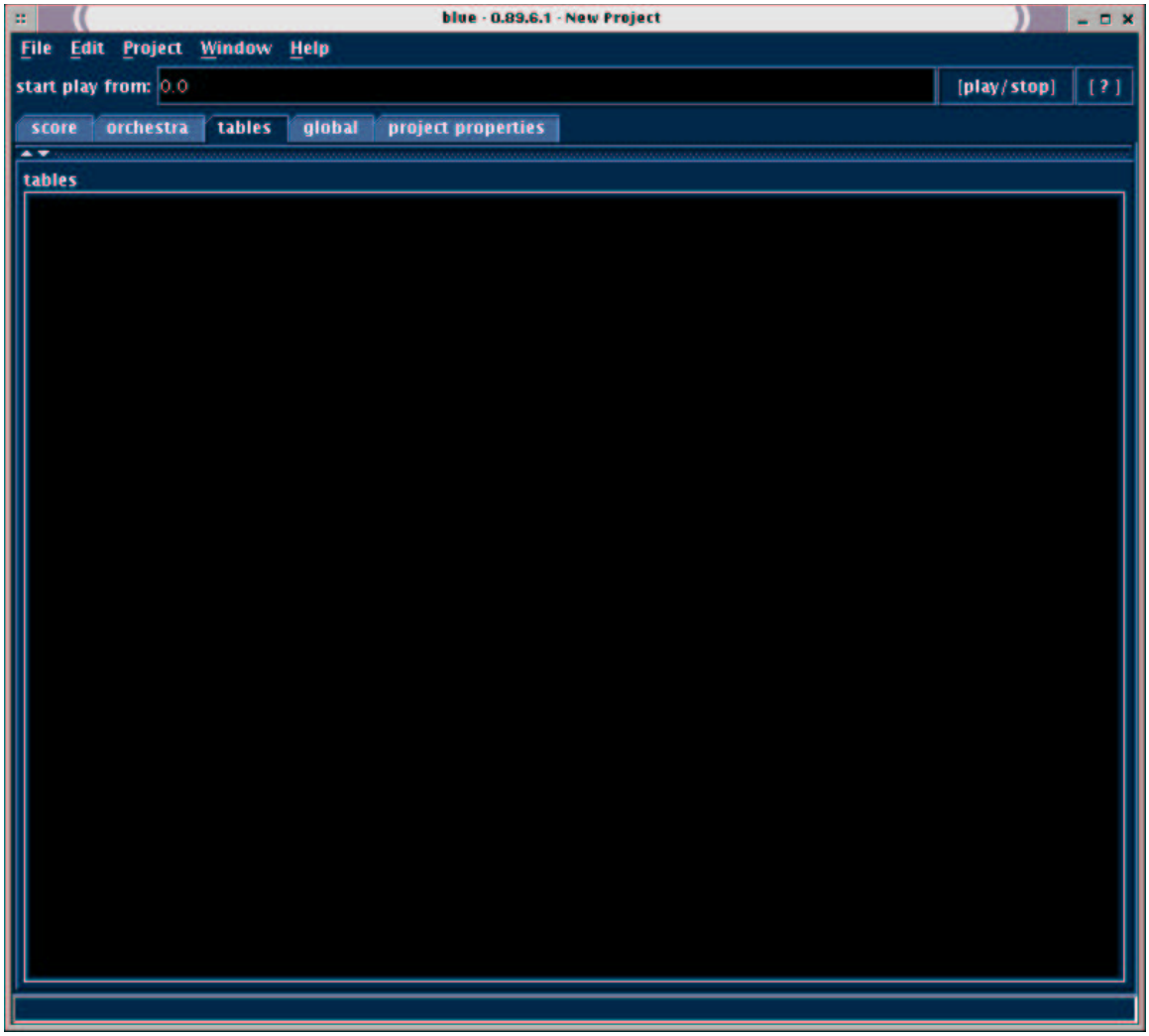
INSTRUMENT LIST
The list on the left side shows a list of instruments by instrument number and instrument name. The checkbox at the beginning of each row is to enable or disable that instrument when generating .CSD files. From my own experience, Csound will take a long time to parse a .CSD file if you have a lot of instruments. Selectively enabling and disabling generation of instruments will help, and will allow you to keep all of your instruments in one file and generate only the ones you need, much like a instrument librarian.

INSTRUMENT PROPERTIES
The top right are properties all instruments in blue have: and i#, name, and comments. the i# is the what is used in csound for defining an instrument (in csound orc syntax, it's what comes after "instr"). Name and comments are for your own use to help organize and document your instrument.

INSTRUMENT EDITTOR
In the bottom right is where the instrument edittor for the selected instrument is shown.

Currently, the only instrument type implemented is genericInstrument, which takes in standard csound orchestra instrument text. In the future, other types of instruments will be made with different interfaces (sliders, etc.) as well as patch saving possibilities.

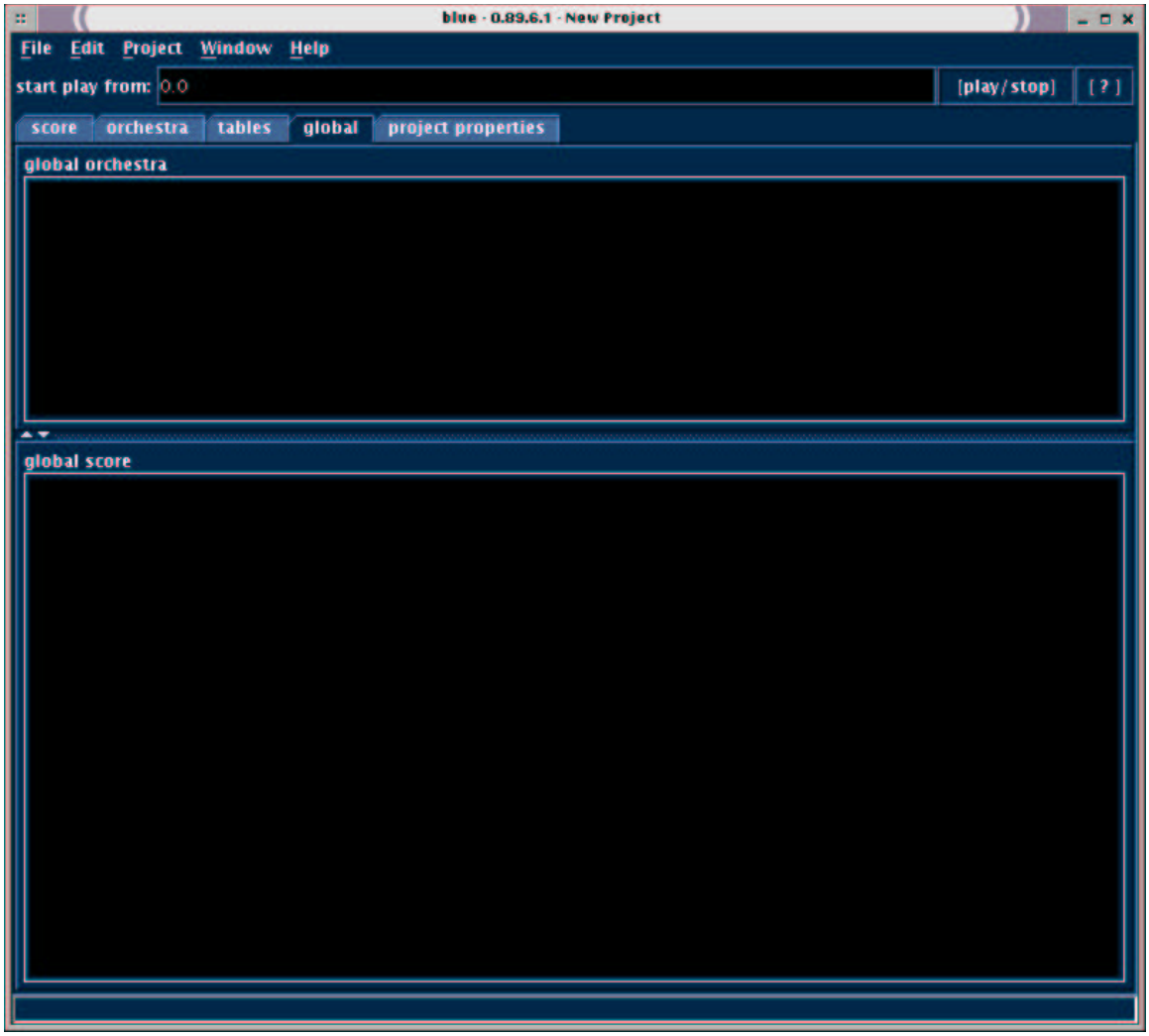
tables manager



The tables tab is pretty plain right now, only containing a text area for tables. Whatever is put in here will be immediately inserted above i-statemtents of the <CsScore> area of the generated CSD file.

Future plans are to have a similar interface to the orchestra with f#'s and names listed on the left, and visual representations of the ftables on the right.

global manager



GLOBAL ORCHESTRA – anything here will be inserted into the <CsOrchestra> section of the .CSD file before anything generated by the orchestra. things like global variables or gui instrumet definitions should go here.

GLOBAL SCORE – anything here will be inserted into <CsScore> section of the .CSD file before anything generated by the score timeline. the global score's processing is done outside of the score timelines, so any notes put here are not factored into values calculated from the timeline, like total duration, nor are any notes in the global score section translated like the timeline is when a different start time is used other than 0.0. for instance, if you use 2.0 as a start time, a note in the score timeline at 2.0 will be translated to start at 0.0, while a note with a start time of 2.0 in the global score section is not affected and will start at time 2.0.

there are also certain variables made available from blue (*blue variables)* that are useful for specific purposes. one of them is <TOTAL_DUR>. an example of it's use is:

i20 0 [<TOTAL_DUR> + 10] .95 1 1000

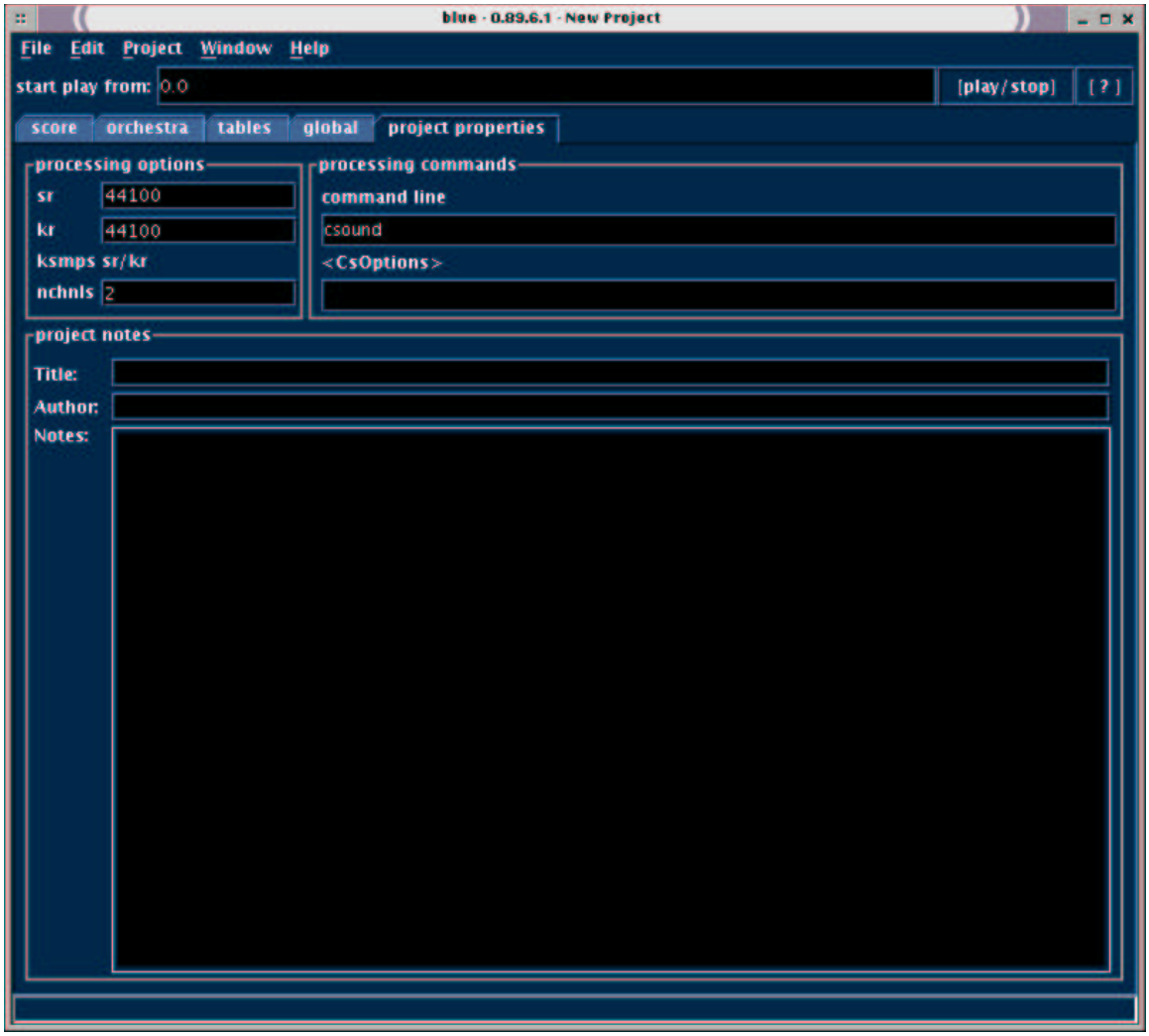
the note give above is for a global effect(a global reverb unit). this note will always start at time zero regardless of when i start playing the score, and will always last as long as the duration of the generated score from the timeline plus 10 seconds. because *blue variables* are a text swap, the above use of the bracket notation that csound uses was necessary. for a score with a 30second duration, the above note would have been generated as:

i20 0 [20 + 10] .95 1 1000

because of the *blue variable* used, i never have to change that note again, and i will always have the reverb unit on, regardless of when i start the score to play.

if i had not used the *blue variable* and put the note in the score timeline, i would have to constantly move the soundObject that had the reverb note to start at a time equal to or great than the play time i designated. (which is why i made the *blue variable* to use in the first place!)

project properties



PROCESSING OPTIONS – these are the standard options for csound, the control rate, sample rate, the ratio, and the number of channels. ksmps is automatically set to dividing the user-given sample rate divided by the user-given control rate.

PROCESSING COMMANDS – the string used in the command line is used for when pressing the play button, the string used in <*CsOptions*> is used when generating a .CSD file to disk. they differ because <*CsOptions*> may be useful when generating a .CSD file out to disk that you will later use to compile to a .wav file, while *command line* may be used for playing at the moment.

the command line to use is exactly the same as you would use on a console. the name of the temporary .CSD file that is generated before calling the command line is appended to the command line before execution.

for myself, when on linux i use:

    icsound-console –driver=portaudio

and on windows i use:

    c:\csound\csound -Wdo devaudio

as an example, if i pressed the play button and used the command line above, and the generated temporary .CSD file is named c:\temp\temp4523.csd, the command that would actually be executed would be:

    c:\csound\csound -Wdo devaudio c:\temp\temp4523.csd

PROJECT NOTES – a place to put title, author, and notes for the work file. for documentation purposes.

[tip] – i find that i often work on multiple platforms, my home computer with linux and my work computers with windows. i usually use multiple command lines with the same file, depending if i open it on windows or linux. usually, i keep all my command line strings in the *notes* area and copy and paste them into the command line depending where i'm at.

PLUGINS

soundObjects --

1. **genericScore**

accepts noteProcessors: yes

contains a block of csound score. the objective time within the genericScore starts at time 0, and notes within the genericScore are written relative to that. moving the genericScore around translates the starting time of the notes within in it, and stretching the block changes the subjective time of the score.

[try this] let's say you have a block of score starting at 0, i.e.:

```
i1 0 1 2 3 4 5
i1 1 1 2 3 4 5
i1 2 1 2 3 4 5
```

make three copies of on the main timeline. now move them so they start at different times, and then resize them. now generate a .csd file from selecting the "generate .csd file" option from the project menu. take a look at the generated .csd file and compare it to what is in blue.

2. **polyObject**

accepts noteProcessors: yes

a timeline. polyObjects can be seen as containers for other soundObjects. PolyObjects can also be embedded within each other.

[try this] on the root timeline, rt-click on a soundLayer and select "Add new PolyObject. you should have added a new polyObject to the timeline. now rt-click on the polyObject and select "edit soundObject". you should now be within the polyObjects timeline, and the button which says [root] should now have another button next to it that says [polyObject]. now try adding a few soundLayers and a few genericScore Objects. now click on [root]. you have now returned to the root timeline. here you should see the polyObject you've edited. now you can scale the object you've created and all of the objects held within will scale together as a group.

3. **sound soundObject –**

accepts noteProcessors: no

inline instruments. when experimenting with sounds, or if you're going to make a soundEffect and not a whole composition, it might be tedious to go and make an instrument and then have to edit score information. this object is here so that you can add instruments directly to the timeline. what is generated is a minimal i-statement that corresponds to the sound soundObject's bar on a timeline. this abstracts a concept of working with the sound itself instead of through the level of time of a note.

4. **pythonObject**

accepts noteProcessors: no

allows for the use of the python programming language to create score data. uses the Jython interpreter to interpret Python scripts. you may add your own python classes to the library for use with "import" statements in blue's "lib" directory, under "pythonLib". included with blue is Maurizio Umberto Puxeddu's pmask, which you will find in the pythonLib directory.

after writing your script to generate notes, you'll have to bring back into blue by assigning the variable 'score' the text string of the generated python score.

example

```
[…code for generating score]
…
score = str(ss)
```

(where ss is an object that will give you a score when made into a string)

5. **comment**

accepts noteProcessors: no

a comment soundObject; used to add comment text on the timeline and does not generate any notes.

6. **instance**

accepts noteProcessors: yes

a soundObject pointing to a soundObject in the soundObject library. the content of the soundObject is not editable except by editting the soundObject in the library to which the instance is pointing to. useful in conjunction with noteProcessors.

noteProcessors --

Note Processors are use in conjunction with soundObjects, and are used post-generation of the soundObject's noteList. They are used to modify values within the noteList.

NoteProcessors can be added via the soundObject property dialog. When a soundObject is selected on the timeline, and if the soundObject supports noteProcessors, you can add, remove, push up, or push down noteProcessors on the property dialog.

1. **AddProcessor**

parameters: value, pfield

The AddProcessor takes two parameters, one for pfield and one for value. When applied, it will add whatever value given to the given pfield for all notes.

For example, you have a soundObject with notes for an instrument that uses p4 as it's amplitude, which it is expecting as in decibels. So if your notes lied within the range of 78 and 82 db, and you used an addProcessor that was set for 4.2 to add to p4, your notes afterwards would be scaled to the range of 82.2 to 84.4.

The AddProcessors expects either a positive or negative float value for the value, and a postive integer for the pfield.

2. **MultiplyProcessor**

parameters: value, pfield

The multiplyProcessor works like the addProcessor, but multiplies the given pfield by the value.

3. **SubListProcessor**

parameters: start, end

The SubListProcessor will cut out notes from the soundObject's generated noteList. An example of it's use may be that you have a 12-tone row as soundObject in the soundObject library, and you're using instances of it as the basis of your work. You may only want to use notes 1-3 of the row, so you would use the SublistProcessor with a start of 1 and an end of 3.

The SubListProcessor will cut out notes, then translate them to start at the start of the soundObject, and scale them so that they take up the duration of the soundObject. If you had a five note soundObject with all notes have a duration of 1 second, all starting one after the other, with the soundObject starting at 0 seconds on the timeline, and if you used a SubListProcessor with start of 1 and end of 4, you'd end up with four notes being generated(the first four from the original soundObject), starting a 0 seconds on the timeline, with each notes duration lasting 1.25 seconds, each starting one right after the other.

4. **PchAddProcessor**

parameters: value, pfield

The PchAddProcessor works like the AddProcessor, but is assumed to be used with Pch notation(i.e. 8.00, 7.11, 9.03). So, if you have a soundObject where the notes inside have p5 is expecting a Pch value, and you the notes have values of 8.00, 8.04, and 8.07, then adding a PchAddProcessor with value -.07 and pfield 5 would add .07 to those notes, ending up with 7.05, 7.09, and 8.00.

If you had used a regular AddProcessor, you would have gotten 7.93, 7.97, and 8.00, and the first two notes would have really be equal to 14.11, 15.01, and 8.00 in Pch notation, and that wouldn't have been the desired affect.

5. **RetrogradeProcessor**

parameters: none

reverses all the generated notes

instruments --

managed by the orchestra manager, the instruments are generated into the <CsOrchestra> section of the .CSD file.

1. **GenericInstrument**

a generic edittor for csound instruments. insert your instrument text in the edittor, without “instr” or “endin”, as they're not necessary and will be generated by blue.

other notes

CODE POPUP – most of the textArea's used in blue are the BlueTextArea, which features a popup menu with shortcuts. rt-clicking on the text area will bring up two options, *opcodes* and *custom*. the *opcodes* shortcuts are read in from opcodes.xml, found in the *conf* directory. (as of this writing, the opcodes.xml file was last synchronized to csound 4.06, and needs some updating...). the *custom* shortcuts are read in from custom.xml, also found in the *conf* directory.

feel free to edit the custom.xml to add you're own submenus and shortcuts.

(i'll be adding a code library edittor that will let you edit the custom.xml file without having to edit xml in the near future.)

IMPORT FROM CSD
from the *File* menu, there is an option to import from a csd file. because blue can not interpret score data as being representative of any soundObject in particular, it takes all score data and places it in the global score section. tables will be put in the tables section, instruments get imported into the orchestra section, and project properties which can be extracted (kr, sr, etc.) will be placed in the project properties section.

importing from .CSD will replace the current working file, so it's best to use when starting from a .CSD and working from there.

usage ideas

GENERIC CSD EDITTOR
for whatever reason you might want, it is possible to not use the managers and edittors within blue, but rather just use the global edittor with the tables edittor. this would basically allow you to work on a csound file like you would in any text edittor.

SOUND DESIGN/PROTOTYPING
using the sound soundObject and working directly on the score timeline, it's nice to experiment with sounds on the timeline by directly using the orchestra language. "Let's see what it'll sound like if i use a sine wave here, together with an 4-op fm sound there..."

i often find myself initially working on the score timeline with the sound soundObjects, experimenting to create sounds and placing them in time as a way of sketching. later, i usually convert them to instruments managed by the orchestra manager and move on from the initial prototyping phase.

INSTRUMENT LIBRARIAN
with or without using blue's other features, the orchestra manager is useful as a instrument librarian. store all your instruments there, and if you want to, you can selective enable and disable them for generation. output a .CSD file and you now have a template .CSD to start working with outside of blue.