

# THE SIREN 7.5 MUSIC AND SOUND PACKAGE IN SMALLTALK

Stephen Travis Pope

CREATE, University of California, Santa Barbara

stp@create.ucsb.edu

## ABSTRACT

Siren is a programming framework for developing music/sound applications in the Smalltalk programming system. It has been under development for more than 20 years, and the newest version (7.5) has a collection of major updates and new subsystems. This paper briefly introduces Siren, and then concentrates on the significant new features, interfaces, and applications in Siren 7.5.

## 1. INTRODUCTION

The Siren system is an open-source general-purpose software framework for music and sound composition, processing, performance, and analysis; it is a collection of about 375 classes written in Smalltalk-80. Siren and its predecessors (MODE, HSTK, and DoubleTalk) have been thoroughly documented; see, e.g., the *Proceedings of the 1987, 1989, 1991, 1992, 1994, 1999, and 2003 ICMCs*, several book chapters, and articles in *Computer Music Journal* (16.3) and *Organised Sound* (1.1). The current version of Siren (7.5) runs on VisualWorks Smalltalk (available for free for non-commercial use) and supports cross-platform streaming I/O via OpenSoundControl (OSC), MIDI, and multi-channel audio ports.

Siren is a programming framework and tool kit; the intended audience is Smalltalk developers, or users willing to learn Smalltalk in order to write their own applications. The built-in applications are meant as demonstrations of the use of the libraries. Siren is not a MIDI sequencer, nor a score notation editor, though both of these applications would be easy to implement with the Siren framework.

There are several elements to Siren:

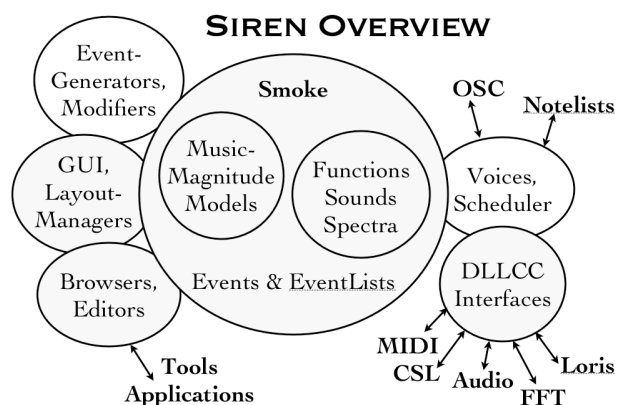
- the Smoke music representation language (music magnitudes, events, event lists, generators, functions, and sounds);
- voices, schedulers and I/O drivers (real-time and file-based voices, sound, score file, OSC, and MIDI I/O);
- user interface components for musical applications (UI framework, tools, and widgets);
- several built-in applications (editors and browsers for Smoke objects); and
- external library interfaces for streaming I/O and DSP math (sound/MIDI I/O, fast FFT, CSL & Loris sound analysis/resynthesis packages)

The Siren release is available via the web from the URL <http://FASTLabInc.com/Siren>. Note that you need a Smalltalk virtual machine and run-time to use Siren; you can down-load the free (non-commercial) system from Cincom at <http://smalltalk.cincom.com>.

The "executive summary" of the Siren music representation (Smoke) (from the 1992 ICMC paper) is

as follows. Music (i.e., a musical surface or structure), can be represented as a series of "events" (which generally last from tens of msec to tens of sec). Events are simply property lists or dictionaries that are defined for some duration; they can have named properties whose values are arbitrary. These properties may be music-specific objects (such as pitches or spatial positions), and models of many common musical magnitudes are provided.

Figure 1. Siren Framework Overview



Events are grouped into "event lists" (AKA composite events or event collections) by their relative start times. Event lists are events themselves and can therefore be nested into trees (i.e., an event list can have another event list as one of its events); they can also map their properties onto their component events. This means that an event can be "shared" by being in more than one event list at different relative start times and with different properties mapped onto it.

Events and event lists are "performed" by the action of a scheduler passing them to an interpretation object or voice. Voice objects and applications determine the interpretation of events' properties, and may assume the existence of "standard" property names such as pitch, loudness, voice, duration, or position. Voices map application-independent event properties onto the specific parameters of I/O devices or formatted files. A scheduler expands and/or maps event lists and sends their events to their voices in real time. Sampled sounds can also be described as objects, by means of synthesis "patches," or signal processing scripts involving a vocabulary of sound manipulation messages.

## 2. Models and Design Patterns

A wide range of the canonical object-oriented design patterns and architecture models are found in Siren:

**Composite** -- events/event lists, display items/display lists -- class structure for hierarchies of objects

**Adaptor** -- voices, ports, graphics -- interface objects translate between message protocol "languages"

**Singleton** -- ports, scheduler, external interfaces -- a class that has a single well-known instance.

**Decorator** -- event modifiers, voices, layout -- one object "wraps" another and forwards messages sent to it on to its "subject" or "component."

**Observer** -- MVC, MIDI, processing -- an object registers itself as an "observer" of some aspect of another object, wanting to get update messages when the observed object changes.

**Strategy** -- layout managers, event generators, voices -- a set of classes provide a family of algorithms that encapsulate their client objects.

**Proxy** -- voices, ports, Smoke, external interfaces -- one object serves as a representative of another in some context, possibly adapting or cacheing.

**Chain of Responsibility** -- MVC, voices, input -- object-oriented recursion iterates through tree structures using composed command objects.

**Visitor** -- Smoke, voices, graphics -- active objects traverse data structures operating on them.

**Double-dispatching** -- Smoke -- polymorphic support for mixed-mode operations among families of classes and species using multiple inheritance.

**Multi-threading** -- scheduler, voices -- several threads share some state to coordinate with one another via mutexes, critical sections, and semaphores

### 3. WHAT'S NEW?

Many parts of Siren were upgraded, enhanced, or rewritten for the 7.5 release. In addition to new features in the **Smoke** representation language and **EventGenerator** models, all of the **external interfaces** (e.g., for sound I/O, MIDI, FFTW, and other external libraries) have been rewritten to be more portable and robust. The entire **EventScheduler** has been updated, and there are several new packages based on **SWIG-generated external interfaces**, such as the **CSL**, **Loris**, **Aubio**, and **LPC** application interfaces. Francois Pachet's **pitch class framework** has been ported from the MusEs system (<http://www-poleia.-lip6.fr/~fdp/MusES-papers.html>). For details, see the on-line Siren workbook page at [http://fastlabinc.com/Siren/Workbook/Pitch\\_Classes\\_and\\_Scales.html](http://fastlabinc.com/Siren/Workbook/Pitch_Classes_and_Scales.html).

The support for **OpenSoundControl** has been updated and improved, so that Siren can serve as an OSC-generating client for servers written in several languages, such as CSL and SuperCollider. A new file format called **s7 files** supports multiple versions of related control and content files and their metadata for use with score versioning systems and analysis/resynthesis tools. Two new tools - the **SirenUtility** and the **SirenTransport** - have been added to assist users with general environment maintenance and with session state, and data persistency has been incorporated into the **SirenSession** class.

Lastly, the documentation, including the web pages, the **on-line Siren workbook**, and the complete **reference manual**, have been greatly enhanced in Siren 7.5. The sections below present the highlights of the new features of Siren 7.5.

## 4. EVENTGENERATORS

Most composers who use Siren for any length of time gradually adapt a generator-centric view, defining their compositions as families of event generator classes with specialized instance creation (constructor) methods for the appropriate mode of description. Though the core EventGenerator abstractions of *Cluster*, *Cloud*, and *Ostinato* are rarely extended is common for users to extend the framework with new creation and filtering/processing methods and scripts in EventGenerator class methods that create event lists and manipulate them. An example of a recent extension is the *ExtDynamic-SelectionCloud* class, which is created with an underlying event list of [(time -> chord) (time -> chord) ...] and then generates its event list (up-front or at run-time) by selecting from among the lists according to a specified cross-fade function.

## 5. SIREN GUI APPLICATIONS

Siren graphical applications are based on the simple display list graphics framework in the class categories *MusicUI-DisplayLists* and *MusicUI-DisplayListViews*. This package includes display items such as lines, polygons, curves, text items, and images, hierarchical display lists, and display list views, editors, and controllers. The display list view/controller/editor are MVC components for viewing and manipulating display lists. Simple examples of the display list framework are shown in the Figures below.

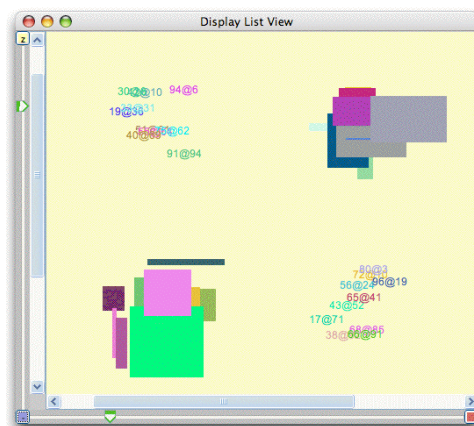


Figure 2. Hierarchical Display List View with grouped scroll and zoom widgets

There are several layouts for the zoom/scroll bars; in the default layout, the bars are grouped on the left and bottom of the window (Figure above, MS-Windows look-and-feel showing a hierarchical display list with four distinct groups). The zoom bars are gray sliders on the outside, and the scroll bars are the usual color and look, and are set inside of the zoom bars. The Figure below illustrates a different layout and look-and-feel, showing selection within a very large display list.

The Siren version of "Navigator MVC" framework is based on layout manager objects that can generate display lists from structured objects. *LayoutManagers* take data structures and generate display lists based on their layout policies. For example, to see a class inheritance hierarchy as an indented list, use an

IndentedListLayoutManager; another LayoutManager might display the hierarchy as a left-rooted tree, as in Figure 4 below.

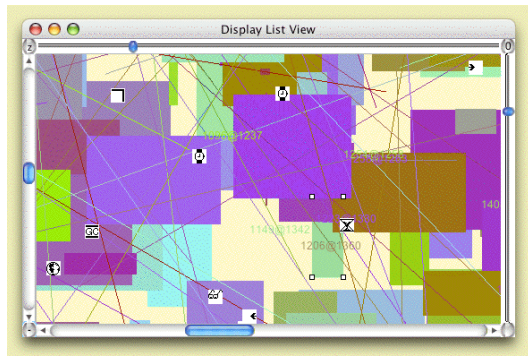


Figure 3. Different Scroll/Zoom widget layout with a large display list and a selection

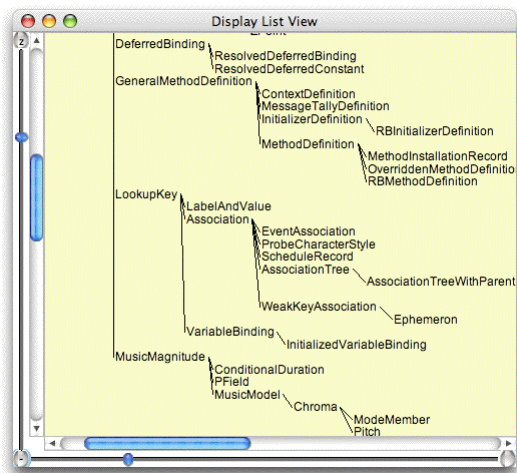


Figure 4. TreeLayoutManager Example displaying a class hierarchy as a left-rooted tree

This framework obviously enables a variety of musical notations, with LayoutManagers that use pitch and time to compute note position and other note properties to determine the associated graphical glyphs, as shown in the simple notation on the next page.

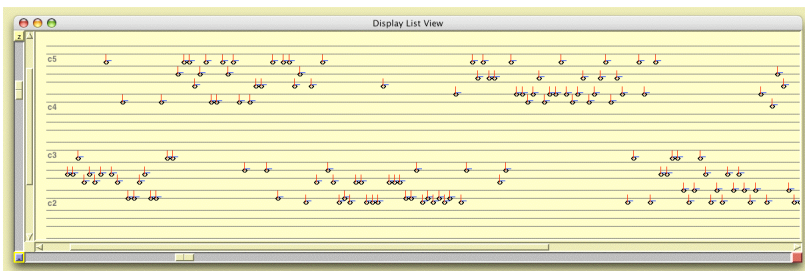


Figure 5. PitchTimeLayoutManager music notation

## 6. USING SIREN SCHEDULERS

Siren supports a flexible score language (Smoke), and separate output performance using a real-time scheduler and “voice” objects that map an event’s abstract properties to the concrete parameters of a specific synthesis technique (e.g., MIDI, Csound note list files, or a given OSC address space). This facility has allowed us to use Siren together with programmable OSC- or

MIDI-controlled synthesis servers written (e.g.,) in CSL or SuperCollider.

The Siren EventScheduler class singleton instance (Schedule) holds onto two SEventQueue lists, one for normal clients (typically event lists), and one just for timers. The SEventQueue is a simply doubly-linked list of ScheduleRecord objects sorted by start time (with a class pool of ScheduleRecords and an efficient insert method). All times are in microseconds (meaning long integers). The main scheduler loop is in the method *run* forks a process running the loop

```
[running] whileTrue: [self callNextAppointment]
```

It’s the *callNextAppointment* method that looks for a client or timer that’s ready and schedules it. If there’s nothing to do, the loop sleeps a bit; the actual amount of the inner delay (i.e., the scheduler’s grain size) is a class variable, so that users can change it at run-time. It is typically in the range of 500 usec. If a client is found to run, it is then asked for its “next time” (or nil), so repeating events (as in an event list or repeating timer) get re-scheduled at eagerly or lazily computed time intervals.

## 7. SIREN’S EXTERNAL INTERFACES

Siren includes several interfaces between Smalltalk and support libraries written in C. These use the VisualWorks dynamic linked library and C connect (DLLCC) package to generate Smalltalk classes whose methods are references (proxies) to C functions in a dynamic library. DLLCC has an interface compiler that works by taking the declarations found in C/C++ header files and generating the wrapper code for a set of Smalltalk classes, along with a pointer to the dynamic library file for use at run-time. The four main DLLCC external interface classes for Siren are:

- LibSndFile - sound file IO in many formats
- PortMIDI - cross-platform MIDI API
- PortAudio - cross-platform audio I/O API
- FFTW - Fast Fourier Transform

For each of these packages, there is a Smalltalk class (a subclass of *SirenExternalInterface*) with the proxy methods, and small module of “glue” code (compiled from 200-400 lines of C), which links together with the actual package’s shared dynamic library.

The methods in the external interface classes are typically called from Siren model classes, such as sound I/O methods in the SoundPort class (which interface to PortAudio), or FFTW methods that call FFTW functions from within the spectrum function class. These wrapper classes often hold an interface object and adapt its methods to the appropriate model API.

## 8. SIREN AND SWIG

Like DLLCC, SWIG (Simplified Wrapper and Interface Generator) is a C/C++ interface generator; it targets scripting languages such as Perl, Python, Ruby, and Tcl.

There is a Smalltalk back-end to the SWIG interface generator ported by Ian Upright; with this, one has external interface objects (as above) whose methods call the functions created by SWIG, which mirror the object

methods of the source package that was fed into SWIG. This mechanism allows us to use SWIG APIs to access the CSL and Loris sound analysis/resynthesis packages from within Siren (See Figure 6 on the next page). These facilities are the subject of another paper by the same author elsewhere in these *Proceedings*.

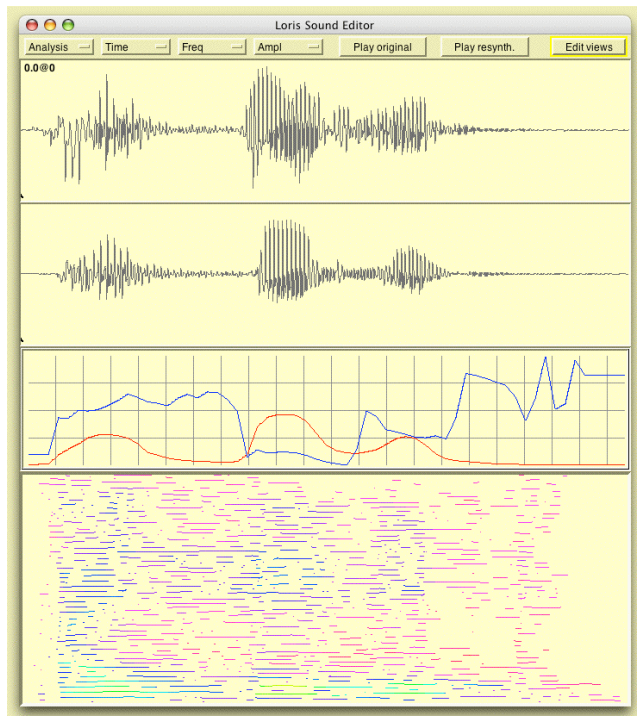


Figure 6. Loris Analysis/Morphing Tool: original sound at top, then resynthesized, then extracted features, then Loris partial spectrum editor; many operations available in the menu bar

## 9. FILE-BASED INTERFACES

There are several new systems in Siren 7.5 that are based on objects whose data is read in from formatted files generated by another application. These are:

- LPC - read/write/edit Csound linear prediction analysis files; and
- Aubio - read output files from the aubio library (<http://aubio.piem.org>) for audio feature extraction, labelling, and segmentation.

The LPC tools support (e.g.,) editing and smoothing speech pitch estimates; the aubio-based tools show up as methods in class *Sound* that allow users to automatically add note onset cues to a sound, or to segment it, or to generate a score from a sound (audio-to-MIDI), which works well for simple, monophonic sound files.

## 10. OTHER NEW FACILITIES

There are many new features in the kernel Siren *Function* and *Sound* classes, including function arithmetic and new sound I/O formats.

Siren 7.5 also defines a new file format, called *s7* files, which can bundle a sound and its associated multiple versions of analysis/resynthesis data (spectral derived envelopes, feature extraction records, etc.) together in a single package. We have reinstated support for instrument spectrum databases such as the Sandell

Harmonic Archive (SHARC), and can now use them to load wavetable samples (as in the next Figure), or to control sample banks of CSL sum-of-sine additive synthesis/mixer/spatialization server configured over OSC.

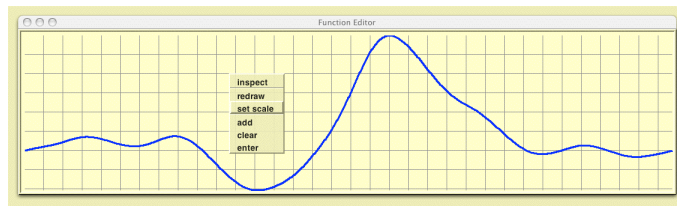


Figure 7. Tuba wavetable generated from the SHARC sample spectrum database

The new configuration and session management tools mentioned above provide GUIs for system set-up and testing (the first pane below), and a multi-threaded cache manager, scheduler transport bar and clock counter (second pane). In the lower view, the buttons on the left are hierarchical menus to load items from the content database, and the list on the right is the loaded sounds, scores, and playing scheduler threads (with its pop-up menu).

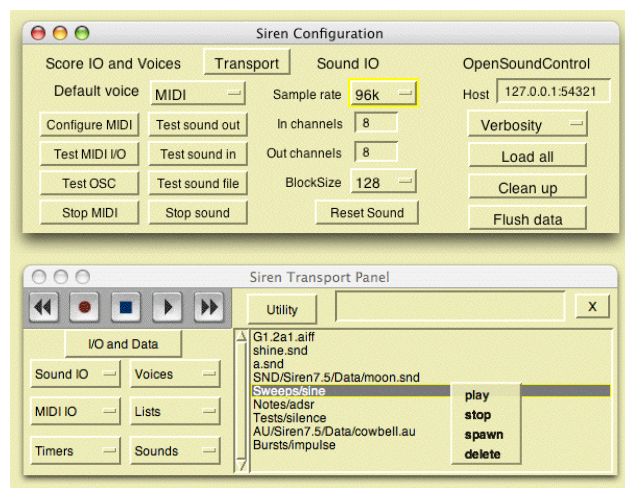


Figure 8. SirenUtility and Siren Transport: MIDI, OSC, and audio IO set-up, session data mgmnt., multi-threaded scheduler control, workspace timelines, and universal transport/clock

Lastly, the CREATE Real-time Application Manager (CRAM) allows a Siren application to function as a grid manager for server-farm-based distributed applications such as large-scale sound synthesis and spatialization.

## 11. EXTENDING SIREN

The Siren 7.5 workbook is composed of active document workbook pages for each of Siren's packages, with extensive descriptions and demonstration code in executable blocks. The Workbook Programmer's Notes enable users to start extending the Siren frameworks, for example with new music magnitude models, new function descriptions and mappings, added event list behaviors, refined layout managers and display methods, adding to the built-in Siren editor and

controller classes for new application features, or even making new external interfaces.

## 12. SOUND EXAMPLES

Since I've used Siren and its predecessors for 23 years now, most of my music from that period used these Smalltalk tools for one phase of composition and realization. See the recent triple-disc (two CDs and one DVD) release *Ritual and Memory* (from EMF) for examples and a video Sampler/Tour of music software tools I've developed (<http://HeavenEverywhere.com/RitualAndMemory/Tour>).

## 13. EVALUATION

While I will claim that Siren is a comprehensive, integrated and sophisticated framework for music/sound description and processing that runs across most popular delivery platforms, I will also acknowledge that it is large, complex, mixes several models and metaphors, is based on a rather obscure programming language and development environment, and has minor bugs and incomplete APIs. Smalltalk is still the simplest and most flexible programming language around (as evidenced by its many imitators, e.g., Objective-C, FScript and Ruby); the Smalltalk class library is the most comprehensive, compact, easy to learn, internally consistent, and best integrated on the planet; and the Smalltalk programming environment (coding and GUI tools) is most powerful development tool set available for any language (and in fact, it has been used to make cross-compiling IDEs for everything from assembler to C++).

To evaluate an application-specific software framework like Siren, the metric of productivity can easily be quantified in terms of the ease of extension of the base models (new event and function methods), the ease of extension of views, and controllers, and the level of effort required for integration of composite widgets into new end-user tools. If the framework works, it should be reflected in a high degree of reuse of software components, with many new features being enabled by a small amount of code (LOC, lines-of-code) spread around in methods in specialized subclasses of various magnitudes, collections, and streams.

We see in the examples of the recent Siren subsystems—the LorisSound model class back-end, and the Loris Analysis/Morphing editor and controller functions (together 2 KLOC), the LPC pitch editor (800 LOC), or the CSL model, external API and scripting tools (500 LOC)—that a diverse set of applications can be written that each represent just a few hundred lines of actual code (and some GUI configuration files). The full Siren application GUI survey ([http://fastlabinc.com/Siren/Doc/Siren.GUI\\_2007.html](http://fastlabinc.com/Siren/Doc/Siren.GUI_2007.html)) bears further witness to the adaptability of the model and widget sets; the entire Siren system, with all models, tools, and GUIs, amounts to about 55 LKOC of Smalltalk.

All this being said, Smalltalk-80 is lamentably far from being mainstream, and students are tending to leave C/C++/Java in favor of contemporary scripting languages, many of which share language features (e.g., message-passing) or delivery system (virtual machines) with Smalltalk, but lack its scalability, base libraries, shared code-base and integrated development environment.

Over the years, Smalltalk's class library has evolved to incorporate code packages for virtually all domains of software, and Siren uses or has used Smalltalk libraries for networking, encryption, data streaming, XML, RDBMS, CORBA, OpenGL, SOAP, HTTP, and many other areas. Smalltalk's development environment is known for its coding browsing and debugging tools, and for its integrated code database and versioning/configuration tools; modern systems use one of several distributed/shared code management systems, such as Envy, Store, or SqueakMaps.

There are also several important subsystems of Siren (esp. the sound mixers, the 8S speech database system, and the harmonic analysis expert system) that are no longer supported, since I never ported their GUIs back after the move to and from the Squeak dialect of Smalltalk (1996-2001). While Smalltalk implementations are quite well (ANSI) standardized in terms of the programming language and core class libraries, the GUI frameworks differ between the various commercial and open-source versions, so that complex applications are not always portable, e.g. between Squeak, VisualAge, VisualWorks, and ObjectStudio. The stale GUIs lamentably include the layout manager and editor for common-practice Western music notation (CMN), and the track/loop DAW-style sound/stream/sequence editor. Examples of the atrophied GUI components are shown in the Figures below.

The central advantages of Siren relative to other languages and tools with which it might be compared are:

- comprehensive support for all phases of score and sound creation, processing, and performance;
- extensible/adaptable framework with malleable code and special “refactoring” tools and source code control; and
- stable (> 20 years) language and code base, cross-platform IO and GUIs, multi-API class libraries.

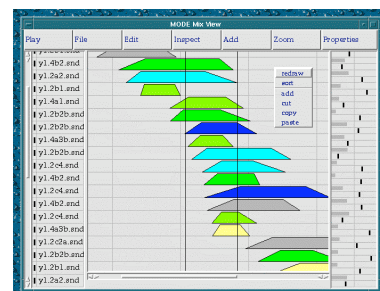
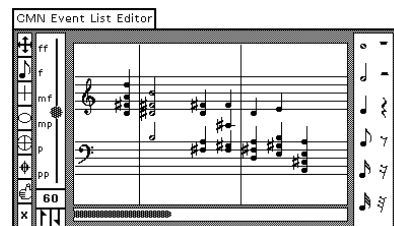


Figure 9. Deprecated layouts: common music notation and MixView

## 14. CONCLUSIONS

The Siren framework is the result of over 20 years of development within the same programming language and code development environment. Over these years, I have used it to build a wide range of tools, from low-level signal tweaking tools (LPC and spectrum editors) to very high-level composition systems (TrTrees and DB-based composers tools). Many of Siren's features and tools would be quite difficult to reproduce on other systems, be they score tools, sound models and APIs, or customizable digital audio workstations.

## 15. REFERENCES AND LINKS

- Pachet, F. 1994. The MusES system: an environment for experimenting with knowledge representation techniques in tonal harmony. *First Brazilian Symposium on Computer Music, SBC&M '94*
- Pope, S. T. 1987. "A Smalltalk-80-based Music Toolkit." *Proceedings of the 1987 International Computer Music Conference.*
- Pope, S. T. 1993. "The Interim DynaPiano: An Integrated Tool and Instrument for Composers." *Computer Music Journal* 16:3.
- Pope, S. T. 2002. "Music and Sound Processing in Squeak Using Siren." Invited Chapter in M. Guzdial and K.m Rose, eds. *Squeak: Open Personal Computing and Multimedia.* Prentice-Hall.
- Pope, S. T. 2007. "Scripting and Tools for Analysis/Resynthesis of Audio" submitted to *Proceedings of the 2007 International Computer Music Conference.* also at <http://FASTLabInc.com/Siren>
- Pope, S. T., and C. Ramakrishnan. 2003. "Recent Developments in Siren: Modeling, Control, and Interaction for Large-scale Distributed Music Software." *Proceedings of the 2003 International Computer Music Conference.*

Aubio: <http://aubio.piem.org>

CREATE Signal Library: <http://FASTLabInc.com/CSL>

CRAM: CREATE Real-time Applications Manager  
<http://create.ucsb.edu/CRAM>

FFTW: <http://www.fftw.org>

LibSndFile: <http://www.mega-nerd.com/libsndfile>

Loris: <http://sourceforge.net/projects/loris>

OSC: <http://www.cnmat.berkeley.edu/OpenSound-Control>

PortAudio: <http://www.portaudio.com>

PortMidi: <http://www.cs.cmu.edu/~music/portmusic>

SHARC: <http://www.timbre.ws/sharc>

SuperCollider: <http://supercollider.sourceforge.net/>

SWIG: <http://www.swig.org>