

DESIGN AND IMPLEMENTATION OF CNMAT'S PEDAGOGICAL SOFTWARE

Michael Zbyszynski

Matthew Wright

Edmund Campion

Center for New Music and Audio Technologies (CNMAT)
Department of Music, University of California Berkeley,
1750 Arch Street, Berkeley, CA 94709 USA
{mzed,matt,campion}@cnmat.berkeley.edu

ABSTRACT

This paper outlines recent developments in pedagogical software resources at the CNMAT. We describe the Max/MSP/Jitter Depot: an organized system where software can be stored and shared. The Depot offers a wide range of support and includes basic programming tips, modular programming units for copy and paste, interactive tutorials on all aspects of computer music, and functioning musical works with commentary and criticism.

1. INTRODUCTION

In recent years, and especially during the 2006-7 academic year, CNMAT has devoted increased attention and resources to Computer Music Pedagogy. The typical learning process of a student composing computer-based music involves encountering many of the same programming problems and inventing the same solutions as their predecessors. While solving basic problems in programming, signal processing, or music has a definite pedagogical value, much of this activity is counterproductive and often impedes serious musical or aesthetic investigation. Rather than subject students to new rounds of wheel-invention, we are developing CNMAT's Max/MSP/Jitter ("MMJ") Depot, an organized system where Max patches can be stored and shared, offering users a range of support from basic programming tips to complete, functioning pieces from the repertoire, along with commentary and criticism. This Depot will be available for public download at: <http://cnmat.berkeley.edu/mmjdepot>.

Other individuals and institutions around the world are also attempting to improve this situation. Notable among them are:

- IRCAM's *Jimmies and Bennies*¹
- SAT's *nSlam*²
- *The UBC Max/MSP/Jitter Toolbox*³
- *Llopp* (collaboratively authored)⁴
- Chamagne and Ninh's *Max Objects Database*⁵
- Place and Lossius' *Jamoma*[2]
- Poletti's *MP.Tools*⁶
- Momeni's *aLib*⁷
- Vade's *v001*⁸
- Clarke, et al's *Sybil* [1]

¹ <http://forumnet.ircam.fr/351.html?&L=1>

² http://tot.sat.qc.ca/logiciels_nslam.html

³ <http://debussy.music.ubc.ca/muset/toolbox.html>

⁴ <http://llopp.klingt.org>

⁵ <http://www.maxobjects.com>

⁶ <http://forumnet.ircam.fr/uploads/media/octobre.pdf>

⁷ <http://www.alimomeni.net/share/max>

⁸ <http://001.vade.info>

The CNMAT MMJ Depot acknowledges these efforts with the development of a Music Information Center that links to resources outside of CNMAT.⁹ It is the authors' hope that the details of our organizational process will inspire others to create their own repositories.

The MMJ Depot is open to users with all levels of musical and technological expertise. Its intended audience includes composers, new media artists, computer scientists, and engineers. Notably, the Depot presents ready-to-use software in an aesthetic and pedagogical context. Those who wish to develop their programming skills and knowledge of electronic and computer music topics can explore the depot, following links to extended explanations when desired. Advanced users seeking robust and efficient programming are encouraged to plunder the depot for useful bits and utilities, including fully functioning applications. The MMJ Depot is implemented using CNMAT's Subversion Repository and Drupal-based website.[5]

2. GLOSSARY

The foundation of our organizational process has been to define a shared set of terms, which are listed below (see Figure 1) and described in more detail throughout the paper.

Application: A generally useful *patch* or *package*, with a GUI, that can be used without doing any Max programming. *Applications* are suggested when a composer or researcher desires to share an activity that has become habitual to the point where a fixed environment can exist. In other words, an *application* is a turnkey solution to some problem, not a reusable software component.

Baseline: The painstakingly selected *package* that we expect all users to have in their Max path upon which all other MMJ Depot software may depend (see 3.1, below).

Demo: A *patch* or *package* that shows one (impressive) idea, not necessarily with enough flexibility or generality to be useful for extended work.

Dependency: When any *patch* requires a specific form of *support*.

External: An object written in C, Java, or Javascript.

Help patch: Shows how one *object* (patch or external) works. Help patches are named *<object>.help* so they will be found by Max's built-in help mechanism.

Module: Generalized *package*, to be used in an *application*, *demo*, *tutor*, or *tutorial*. Modules have specific design guidelines (see 3.2, below).

⁹ Furthermore, it is possible to create custom web searches that encompass multiple websites of interest. e.g., <http://www.emsearch.net>

Object: General term meaning either a *patch* or an external written in C, Java, or Javascript.

Package: A specific, organized set of *objects* that serves some purpose.

Patch (aka “abstraction”): Any unit of software made in Max.

Repertoire: An *application* specific to a particular aesthetic expression, such as a musical composition or installation. (Repertoire is the only category which can have dependencies outside of the Depot.)

Support: An *object* that somehow supports your work, or a directory of same.¹

Tutor: A *patch* reference document that serves as a single collection point for the expanding communal wisdom on a particular *object* or programming topic.

Tutorial: A *package* explaining and contextualizing a topic for teaching and learning purposes.

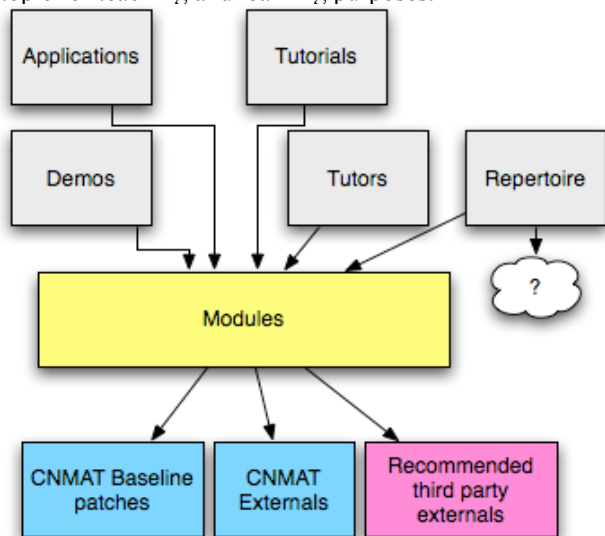


Figure 1 MMJ Depot Structure (arrows indicate possible dependencies)

3. CNMAT’s Max/MSP /Jitter Depot

3.1. Baseline Patches

This carefully selected set of patches is deemed so useful and general that we require all our users to have it in their Max search path; any of CNMAT’s pedagogical software may depend on these patches. The baseline is designed to include everything that we find essential to general programming, while remaining small, understandable, and supported. All of our baseline patches come with help patches. The baseline patches do not embody research breakthroughs; they are simply useful. Here are a few examples:

- *change-sym* is like *change* but works for symbols.²
- *int+frac* takes a float input, and outputs the integer part and the fractional part.³
- *multibuf* reads an arbitrary number of sound files into individual *buffer~* objects.⁴

¹ This meaning of the term “support” is not to be confused with the legal meaning, as in the sentence “Regents has no obligation to provide maintenance, support, updates, enhancements, or modifications” from the license that covers this software.

² *change-sym*: <http://cnmat.berkeley.edu/patch/2709>

³ *int+frac*: <http://cnmat.berkeley.edu/patch/2707>

- *nearly==* checks if two numbers are equal within a given tolerance.⁵
- *pipe-any* is like *pipe* but allows arbitrary messages and lists.⁶

3.2. Modules

Modules are generalized packages that are the building blocks of the CNMAT MMJ Depot. They are designed to be useful by Max programmers at all levels of experience. Any user can place a module into a larger patch and use it as designed. For beginners this allows musical work without immediately tackling all of the difficulties of the programming environment. More advanced users can look inside a module to find and copy solutions to typical programming problems that suggest good practice for future implementations, while expert users can revise modules then check in and release [5] new versions.

Modular systems such as Sybil [1] and Jamoma [2] provide a high degree of consistency in the end result by requiring that module developers adhere to conventions about visual layout, signal output, standardized input messages, etc., whereas collections of patches such as aLib and the Bennies are simply individual programmers’ collections of useful patches, without necessarily having any formal programming conventions. CNMAT has always valued technical and aesthetic openness and flexibility over conformance to predefined institutional structures.⁷ The only specific programming structures for our modules are that they take control input in the form of OSC messages [4], are able to be instantiated multiple times, and come with help patches. Where applicable, they can report their current settings by outputting the OSC messages that would get them to that state. The price of admission is therefore low for the programmer wishing to contribute a module. Many of the patches in the Depot have been curated from many years of CNMAT Max work that predates it; implementations of new features and best programming practices respect the intentions and style of the original programmer.

Each module’s OSC namespace is designed for use as a branch of an application’s namespace. Each instance of a module can be named using an argument, making it easy to address plural instances on one “OSC” channel. All of the messages that a module understands are outlined in a basic help patch, another common element of a module. Finally, many modules will have an interface patch that is intended for use in a *bpatcher*. The interface patch presents the user with a common set of controls in a compact, visual format. Because the interface patch is only an interface, it is easy for users to decide whether or not to use it.

3.2.1. Groovewrap⁸

Groovewrap is a quintessential module; it is basically a wrapper around the MSP object *groove~*, which plays

⁴ *multibuf*: <http://cnmat.berkeley.edu/patch/2813>

⁵ *nearly==*: <http://cnmat.berkeley.edu/patch/2702>

⁶ *pipe-any*: <http://cnmat.berkeley.edu/patch/2529>

⁷ Evidence of this value system can also be seen in the design of the OSC and SDIF protocols.

⁸ *Groovewrap*: <http://cnmat.berkeley.edu/patch/2858>

samples back at a variable rate from a buffer. *Groovewrap~* allows the user to specify transposition of sounds in half-steps (calculating the proper playback rate) and prevents clicks by creating a gain ramp at the beginning of playback. It also provides increased control of starting and stopping times, and *groove~*'s looping features. The *groovewrap* module includes an interface patch, *gwinterface*, which is used extensively in the *samplemixer* application (see section 3.3.1).

3.2.2. *Panhandler*¹

Panhandler is a package of patches that manage spatialization with Ville Pulkki's VBAP object [3], allowing users to specify panning azimuth independently of loudspeaker positions. *Panhandler~* works with one signal input, setting up the specified number of speaker locations in a circle, and configuring a *matrix~* object to handle gain scaling. It also provides users with messages to define time-based behaviors, either circling at a fixed rate or going to a specific azimuth in a number of milliseconds. It keeps track of the current azimuth at all times, so a sound's location will not seem to jump suddenly unless the user specifically calls for it. *Multipan~* does the same for multiple input signals.

3.3. Applications

3.3.1. *Samplemixer*²

Samplemixer is an *application* built out of modules, and functions as a compositional assistant. Four *groovewrap~*'s are mixed together to make a musical phrase with four samples. The users can compose a phrase by adjusting the timing, gain, and frequency parameters of each sample. The OSC messages that configure a phrase can be captured, and entire phrases can be recorded into a new sound file.

3.3.2. *Music Calculator*³

Music Calculator is an application that presents common calculations in the frequency and time domains, such as converting beats per minute to milliseconds per beat, or MIDI note to frequency. This application can be accessed via Max's "extras" menu, allowing users quick access to the workspace.

3.4. Tutors and Tutorials

Tutors and *tutorials* are designed primarily for learning purposes, but are also available to be plundered. The tutorials explain specific topics in depth; they cover programming details as necessary but tend to focus on core theoretical and conceptual issues in computer music, synthesis, signal processing, etc.

3.4.1. *Tutorial: CNMAT Spectral Tutorials*⁴

The *CNMAT Spectral Tutorials* grew out of a mini-course on CNMAT Technologies. CNMAT has developed a large collection of Max/MSP external objects that deal with aspects of spectral (additive and

resonant) synthesis. While each object is well documented with its own help patch, what was lacking was an overview that explained how the objects were intended to work together and what artistic problems they were developed to solve. Also, the help patches do not explain any of the underlying principles of spectral synthesis. The *Spectral Tutorials* were designed to address both of those concerns. Furthermore, although some of the tutorials are quite specific to implementation in Max, many offer general information that would be useful to anybody interested in spectral synthesis.

The *Spectral Tutorials* offer instruction to users with little or no experience with Max/MSP or computer music while giving concise guidance to more experienced users. In either case, it is important that the navigation and use of the actual patches be as transparent as possible. To accomplish this, a tutorial architecture was written in Max that is now slated to become its own "Tutorial Making Tutorial." The architecture has the following features:

Navigation: A single *bpatcher* appears in every numbered tutorial patch. It leverages Max's *pcontrol* object to enable the user to move to the next or previous tutorial, as well as to jump to any tutorial in the table of contents. When the user requests a new patch, the current patch is closed.

Banner: The title and general information are unified in another *bpatcher* at the top of every page

Visual Cues: Whenever possible, the tutorial controls are standardized, especially the controls that turn audio on and off and control gain. In the early patches, these controls are highlighted by blinking "LED" objects. Important Max objects are yellow while objects for the user to interact with are blue.

Overviews: Many of the tutorial chapters have summaries at the end. This proves useful when presenting them in a classroom environment and helps users who are working alone keep track of the bigger picture.

The *Spectral Tutorials* also incorporate patches that work in other roles in the CNMAT MMJ Depot. For instance, *sin-synth~* is a module and *singing-voice~* and *singing-voice~.help* combine to form a demo. The *migrators-explorer* example comes directly from David Wessel's repertoire.

3.4.2. *Tutor: DSP Settings*⁵

In contrast to the dozens of patches that make up the CNMAT Spectral Tutorials, a tutor such as *DSP_Settings* is usually only one patch. Within that patch, however, there are multiple layers of information (from a quick overview to a more thorough explanation) that anticipate users with different needs. The top level contains message boxes that users can copy and paste into their own work. These messages are organized by goal: efficiency, solid timing, etc. (see figure 2). The subpatches in a cluster near the top of the main level are entry points into detailed explanations of the Max Scheduler Modes and the possibilities for DSP Settings.

¹ *Panhandler~*: <http://cnmat.berkeley.edu/patch/2861>

² *Samplemixer*: <http://cnmat.berkeley.edu/patch/2860>

³ *Music Calculator*: <http://cnmat.berkeley.edu/patch/2727>

⁴ *CNMAT Spectral Tutorials*: <http://cnmat.berkeley.edu/patch/2741>

⁵ *DSP Settings*: <http://cnmat.berkeley.edu/patch/2738>

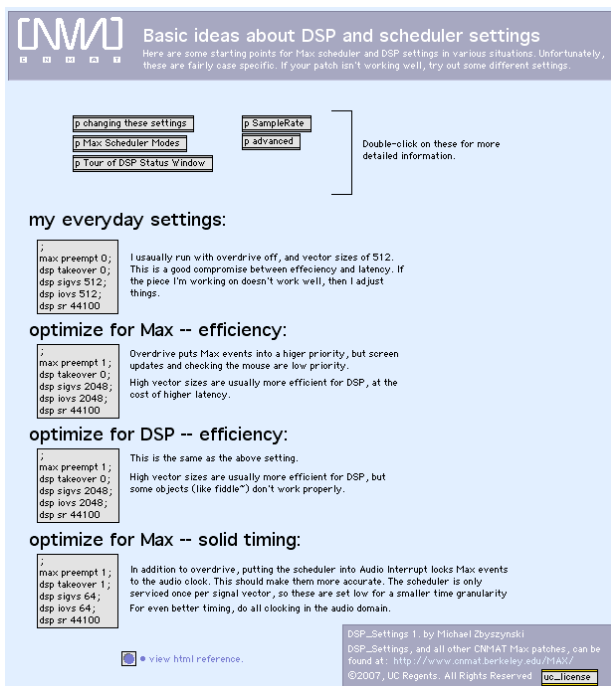


Figure 2 DSP Settings Tutor

3.4.3. Tutor: set-tutor¹

By convention, many of Max's built-in objects accept a "set" message that sends new input into an object without causing the object to output in response to the input. This tutor is CNMAT's central location for the theory and practice of using "set" messages. It includes simple examples demonstrating the use of "set" in low-level terms, as well as practical uses of "set" to keep multiple visual representations of the same data in sync. Whenever people in the CNMAT community discover a new use for this technique, they can add another example to this tutor; thus the tutor gradually collects more and more information and contributes to the institutional memory.

3.5. Demos and Repertoire

The Demo category is a collection of patches that are good at quickly exposing the possibilities of a patch or technique. The *singing-voice* patch is frequently used in this context. Unlike other categories, the demo category is not exclusive; many modules, tutorials, and pieces from the repertoire also function well as demos.

Repertoire includes finished pieces of music. It is not enough to ask "What can this software do?" By presenting technologies in an aesthetic context, the repertoire integrated into the Depot answers to the question: "What is this software good for?"

4. FUTURE WORK

For the first half-century of computer music, we have been mapping a frontier where, to some extent, each individual needed to be self-sufficient. CNMAT has annually experienced new groups of students trying to get started and falling into the same pitfalls as their

predecessors. We know there are solutions to these problems, but some students have not been able to access them. Having new students come in and solve the same programming problems again and again is productive for neither them nor CNMAT. Operating alone on the frontier will always be an important mode for some, but computer music in general is outgrowing the phase where each composer starts from nothing, and entering a phase where we improve our aesthetic results by learning from and using previous work.

One major technical advance would be automatic extraction of the list of all objects that a *patch* uses. We could then use this information to generate automatic cross-references that could answer questions like "what are all the patches in the MMJ Depot that rely on the *multibuf* patch?" Currently, this information is entered by hand.

Now that our basic infrastructure is in place and we have proven the value of this work with a small number of examples, the current task is to assimilate our existing pedagogically useful software into this framework, thereby making that software usable by students and the public outside of CNMAT. Future students and researchers will continue to add to the Depot; the dynamic nature of music technology requires that the depot is never finished, but is similarly dynamic. The ongoing process of contribution and refinement, where students learn by working with and adding to the MMJ Depot, is part of the pedagogical agenda at CNMAT. Through this process the whole Depot performs a mnemonic role in the community, representing the sum of many individuals' experiences.

5. ACKNOWLEDGEMENTS

Richard Andrews, Aaron Einbond, Adrian Freed, Jeremy Hunt, John MacCallum, Andy Schmeder, Jen Wang, David Wessel, and the UC Berkeley Music Department.

6. REFERENCES

- [1] Clarke, M., Watkins, A., Adkins, M. and Bokowiec, M., *Sybil: Synthesis by Interactive Learning*. Proc. ICMC, (Miami, FL, 2004), International Computer Music Association, 354-357.
- [2] Place, T. and Lossius, T., *JAMOMA: A Modular Standard for Structuring Patches in Max*. Proc. ICMC, (New Orleans, LA, 2006), International Computer Music Association, 143-146.
- [3] Pulkki, V., *Generic Panning Tools for MAX/MSP*. Proc. ICMC, (Berlin, Germany, 2000), International Computer Music Association, 304-307.
- [4] Wright, M., Freed, A., Lee, A., Madden, T. and Momeni, A., *Managing Complexity with Explicit Mapping of Gestures to Sound Control with OSC*. Proc. ICMC, (Habana, Cuba, 2001), International Computer Music Association, 314-317.
- [5] Freed, A., Schmeder, A., Wessel, D., and Wright, M. *CNMAT Information Infrastructure*. Proc. ICMC, (Copenhagen, Denmark, 2007), International Computer Music Association, present volume.

¹ set-tutor: <http://cnmat.berkeley.edu/2862>