


- Research reports
- Musical works
- Software

# PatchWork

## Introduction

*Second Edition, April 1996*

IRCAM  Centre Georges Pompidou

Copyright © 1993, 1996, Ircam. All rights reserved.

This manual may not be copied, in whole or in part,  
without written consent of Ircam.

This manual was written by Mikhail Malt,  
in collaboration with Curtis Roads,  
translated and revised by Joshua Fineberg, and  
produced under the editorial responsibility of  
Marc Battier - Marketing Office, Ircam.

The software was conceived and programmed by  
Mikael Laurson, Camilo Rueda, and Jacques Duthen.

Version 3.0 of the documentation, April 1996.  
This documentation corresponds to version 2.5.1 of the software.

Apple Macintosh is a trademark of Apple Computer, Inc.  
PatchWork is a trademark of Ircam.

Ircam  
1, place Igor-Stravinsky  
F-75004 Paris  
Tel. (33) (1) 44 78 49 62  
Fax (33) (1) 42 77 29 47  
E-mail [ircam-doc@ircam.fr](mailto:ircam-doc@ircam.fr)

---

# Ircam Users Groups

The use of this program and its documentation is strictly reserved for members of the IrcamSoftware Users Groups. For further information, contact:

Marketing Office  
IRCAM  
1, Place Stravinsky  
F-75004 Paris  
France

Telephone (1) 44 78 49 62  
Fax (1) 42 77 29 47  
Electronic mail: [bousac@ircam.fr](mailto:bousac@ircam.fr)

Please send suggestions and comments on this documentation to

Electronic mail: [ircam-doc@ircam.fr](mailto:ircam-doc@ircam.fr)



To see the table of contents of this manual, click on the Bookmark Button located in the Viewing section of the Adobe Acrobat Reader toolbar.

# Contents

Résumé .....	6
PW and Computer Assisted Composition .....	7
History of the PatchWork Project .....	7
Basic Elements of PatchWork .....	9
Modules .....	9
Patches .....	9
Editors .....	10
Lisp Programming .....	13
Prerequisites .....	13
The PatchWork Documentation Set .....	14
Getting Started with PatchWork .....	15
Opening a PatchWork Image .....	15
A Walk Through Nth Overtones .....	16
PatchWork Windows and Menus .....	28
Quick Tour of the PatchWork Menu Bar .....	30
The PWoper Menu Detailed presentation .....	36
The Kernel and Music Menus .....	48
The Kernel Menu .....	48
The Music Menu .....	61
PatchWork Modules .....	65
What is a Module? .....	65
Structure and Behavior of PatchWork Modules .....	68
Operations on Modules .....	74
Lisp Data Types and PatchWork Interconnections .....	80
On-line Documentation .....	81
Additional Keyboard Commands .....	83
PatchWork Editors and Musical Objects .....	88
The multi-bpf Editor Module .....	88
Displaying Data with the multi-bpf Editor .....	91
PatchWork's Musical Structure .....	98
The Chord Editor Module - measure-line .....	99
The chordseq Module .....	111
The multiseq Module : Editing Polyphonic Sequences .....	115
The rtm Rhythm Editor Module .....	118
Bibliography .....	139
Index .....	140

# Résumé

Dans ce manuel, vous trouverez des présentations approfondies sur les différents aspects de la programmation avec PatchWork (PW). La lecture de ce document sera utile pour faciliter la prise en main du logiciel. Il est complété par un manuel tutoriel (*PatchWork Tutorial*) qui, lui, contient 40 exemples commentés d'applications musicales.

Après une brève introduction à l'histoire de PatchWork (« PatchWork and Computer Assisted Composition »), le manuel présente les éléments de base (« Basic Elements of PatchWork ») : modules, patches, éditeurs, programmation en Lisp, documentation.

Vous pourrez utiliser le logiciel grâce à la présentation d'un exemple concret (chapitre « Getting Started with PatchWork »).

Les fenêtres et les menus de PatchWork sont commentés dans le chapitre « PatchWork Windows and Menus ».

Le principe et l'emploi des modules (ou « boîtes ») sont présentés dans le chapitre « PatchWork Modules ».

Enfin, la présentation des éditeurs et des objets musicaux fait l'objet du dernier chapitre: « PatchWork Editors and Musical Objects ».

Après la bibliographie, un index clôt le manuel.

# 1 PW and Computer Assisted Composition

PW (PatchWork) is an interactive environment for computer assisted composition. It consists of a set of tools that help the composer generate and manipulate material in the "precompositional" stages. Such an environment can be thought of more as an intelligent sketchpad or a powerful musical calculator, and less as a machine for automated composition.

An environment for computer-assisted composition must provide tools for the creation of musical material. These tools must be capable of representing, playing and transforming the created material. The composer must be able to reuse previous steps of work, to define structural hierarchies in which low-level material can be elaborated.

PatchWork's lets users choose the level of access to the musical objects (for example: a note, a beat, a measure, a section, a whole piece); thus a composer can work on musical material and musical form in a continuous manner.

Another requirement of an environment for computer-assisted composition is communication with other programs for editing, live performance, music printing, and sound synthesis.

Finally, an environment for computer-assisted composition must be extensible. It must provide tools not just for the creation of musical objects, but also for the creation of new concepts, formalizations, and transformations.

To summarize, the PatchWork environment is an interactive environment for music composition and control of sound synthesis. It helps composers do the following:

- Generate and manipulate musical objects (chords, sequences and rhythms organized in hierarchical structures)
- Exchange material with other environments such as Max, MIDI sequencers, musical notation programs (Coda Finale), or any application that can make use of data in a text format (ASCII), Enigma files and MIDI files.
- Control sound synthesis (MIDI synthesizers, Csound, CHANT, Mosaïc)
- And generally, create and execute Lisp programs through the intermediary of graphic patches

The current version, written by Mikael Laurson, Camilo Rueda and Jacques Duthen, was developed in Apple Macintosh Allegro Common Lisp. PatchWork's paradigm is based on functional programming, through the use of graphical *patches*. As well as graphical patches, users can access the text-based programming environments Common Lisp and CLOS (Common Lisp Object System).

## History of the PatchWork Project

Computer-assisted composition and tools for synthesis control have been permanent interests at Ircam, as witness these predecessors of PatchWork:

FORMES (1982) — developed for the control of synthesis and the organization of hierarchical tasks, especially for CHANT, the phase vocoder, and the Ircam 4X synthesizer.

CRIME (1985-1986) — an environment for computer-assisted composition in Lisp; the first at Ircam to generate output in contemporary music notation.

PreFORM (1986) — an object-oriented Lisp platform, for Apple Macintosh. This program offered the possibility of graphical interaction; it also controlled MIDI functions.

Esquisse (1988) — an environment for computer assisted composition that incorporates aspects of spectral music

The original concept for PatchWork was conceived by Mikael Laurson in Finland. It was then integrated into Ircam as a project of the Department of Musical Research. The initial program was rewritten and enhanced by Mikail Laurson, Jacques Duthen and Camilo Rueda at Ircam.

Musical applications and documentation were written by Xavier Chabot, David Waxman, Francisco Iovino, Joshua Fineberg, and Mikhail Malt. Jean-Baptiste Barrière, Tristan Murail and Andrew Gerzso supervised the development work.

Pierre-François Baisnée, Yves Potard, Magnus Lindberg, Marc-André Dalbavie, Kaija Saariaho, Antoine Bonnet, François Nicolas, and Frédéric Durieux participated in the project at different stages of the development. The MIDI driver and task manager are by Lee Boyton, revised by Jacques Duthen.

The design of version 2.5.1 of PatchWork is the result of collaboration between Camilo Rueda, Mikhail Malt, Tristan Murail, Xavier Chabot, Gérard Assayag, and Antoine Bonnet. The actual version 2.5.1 of PW is the result of collaboration between Gérard Assayag, Carlos Augusto Agon and Mikhail Malt.

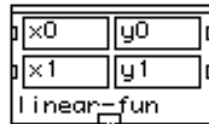


# 2 Basic Elements of PatchWork

PatchWork can be divided into four basic elements: modules, patches, editors, and Lisp programming. Chapters 4, 5, 6, and 7 examine these elements in detail. Here we present a brief overview of each one.

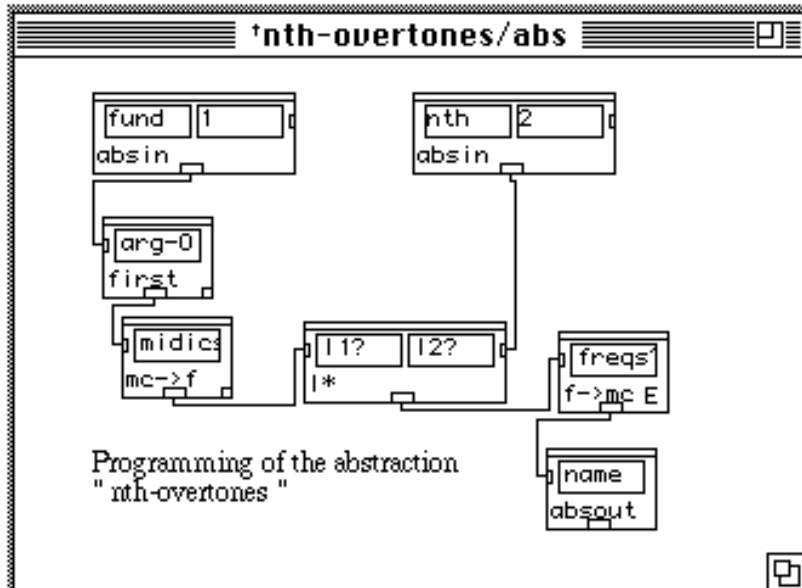
## Modules

A module is a usually a graphic representation of a Lisp function. These representations are presented in the form of rectangles (boxes) that contain inputs of specific types and an output. Modules can be evaluated by placing the cursor on the output rectangle (see design below) and Option-clicking with the mouse.



## Patches

A patch is a construction of PatchWork modules in a window. It is, in reality, a graphic program, as the following example shows.



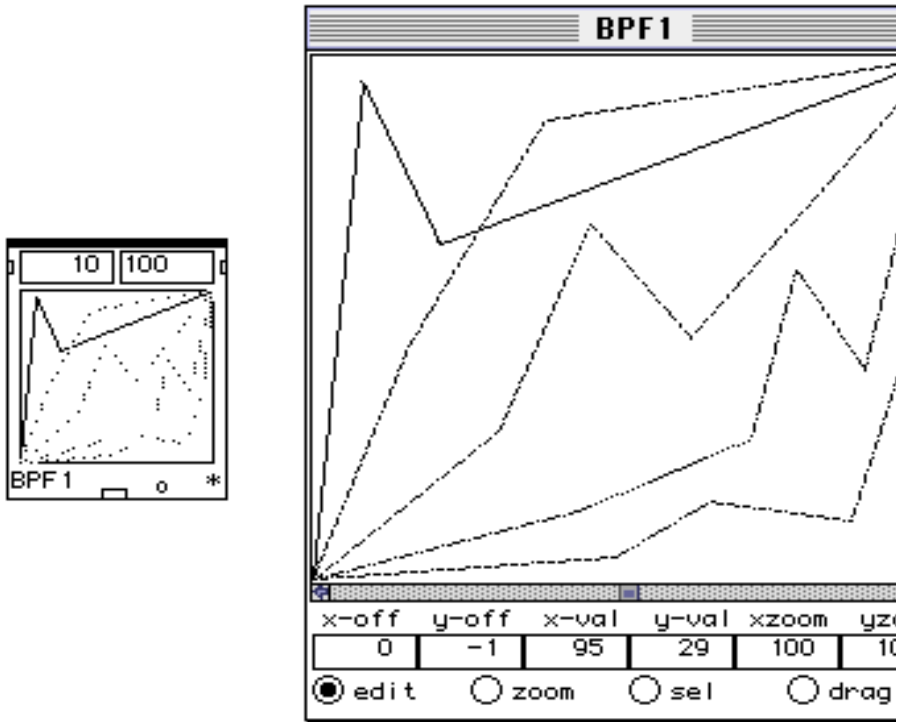
# Editors

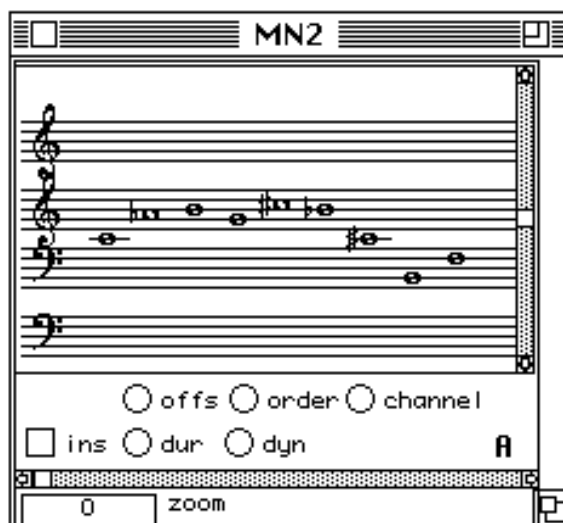
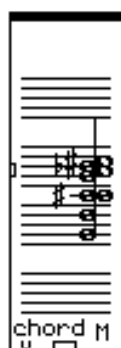
An editor is a specialized PatchWork module that permits not only the calculation (or rather, evaluation) of parameters, but also graphical editing of the entered parameters. PatchWork offers these editors:

1. multi-bpf, for simultaneous breakpoint functions (line segments)
2. chord, for music notation of a chord with MIDI playback
3. rtm (rhythm editor), for music notation of pitches and arbitrarily complicated rhythms with MIDI playback
4. poly-rtm, for multiple simultaneous rhythms sequences (from the rhythm editor) with MIDI playback
5. chordseq, for a sequence of chords with MIDI playback
6. multiseq, for multiple sequences of chords with MIDI playback

The next pages show screen images of the first three editors. All the editors will be explained later on.

Breakpoint function editor





(4 4)	((4
(6000	60
RTM2 <input type="checkbox"/> x E*	

S

$\frac{4}{4}$   
 $\frac{4}{4}$

= 60

1

1

1

1

100

100

Play

SPlay

Stop

measure

staff

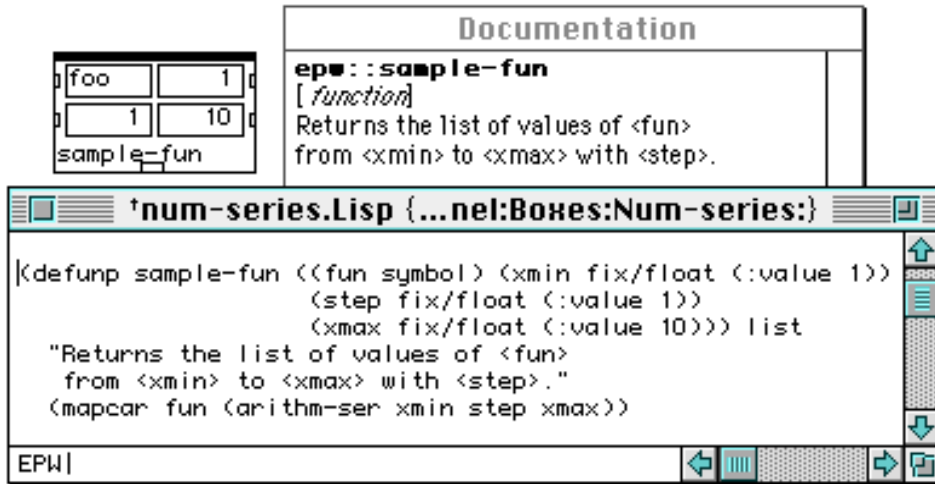
stcnt

scale

speed

# Lisp Programming

PatchWork is embedded in Common Lisp. This means that users can take advantage of the rich Lisp programming environment (interpreting, browsing, debugging, etc.). The functions of PatchWork's modules can be edited and used as a point of departure to create new modules. Every PatchWork module has associated on-line documentation, accessed by typing `d` with the module selected.



We strongly recommend that new users learn Lisp at the same time that they learn PatchWork. See the list of Lisp references later.

## Prerequisites

Apple Macintosh — Experience with the Macintosh environment is assumed: using the mouse, organization of folders and files, using menus, standard commands, etc.

MIDI — The user is assumed to be familiar with the MIDI standard (channels, note messages, etc.), MIDI synthesizers, and Standard MIDI Files.

Lisp — PatchWork is written in Macintosh Common Lisp. It is possible to use PatchWork on a superficial level without knowing Lisp; however, we strongly recommend that Lisp be learned in parallel with PatchWork .

# The PatchWork Documentation Set

## ***The basic PatchWork documentation***

The basic PatchWork 2.5.1 documentation set includes the following components

*PatchWork - Introduction* — Introduces the different parts of PatchWork and its basic characteristics. The first section explains how to open and start working with PatchWork, then each element of PatchWork is presented in detail. You are reading this document.

*PatchWork - Reference Manual* — Documents the Kernel and Music menus functions.

*PatchWork - Programming Guide* — Explains how to create new PatchWork modules. Accessed online as PW Prog Guide.

*PatchWork - Tutorial* — Hands on introduction to making music with PatchWork. Set of 40 commented tutorials.

## ***Documentation of PatchWork libraries***

There is an always growing list of documentation for PatchWork libraries. See the latest PatchWork Newsletter

## ***Online musical patches***

*Example Patches* — Several dozen working patches that demonstrate musical applications of PatchWork. See the READ ME file in the directory for more details.

## ***Online documentation***

*Doc on-line* — A screen of textual documentation describing a selected module. Accessed by typing *d* with any module selected.

*Module Help* — Example patches that present each module found in the two menus PWkernel and PWmusic. Accessed by selecting a module in a patch and typing *t*.

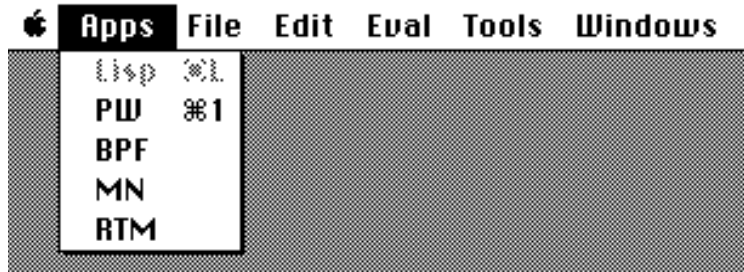
*Window Help* — A page of useful information, including editing commands, etc. Type *H* in a PatchWork window to bring it up.

# 3 Getting Started with PatchWork

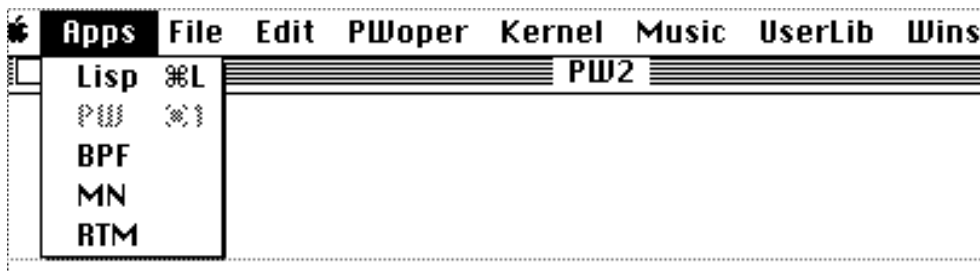
## Opening a PatchWork Image

To load the program PatchWork, double click on the file PW-Music.Image, located in the folder images in PW-2.5.1. It is essential that all the folders related to PatchWork must be correctly arranged (refer to the PatchWork Newsletter). An *image* is another term for a loadable application program in Lisp.

When the question mark appears in the listener window, you are ready to begin your work. The Listener window appears at the bottom of the screen (which is actually a Lisp window) and a menu bar at the top :



this indicates that we are in Macintosh Common Lisp. To move into PatchWork, there are two choices: either select PW under the **Apps** menu, or type the keyboard equivalent Command-1 (one). After switching to PatchWork, a new window appears named **†PWxxx** ( xxx is the number of the window; it is 1 when the program is first loaded) and the menu bar changes to the following :



To return to Lisp, select **Lisp** in the menu **Apps**, or type Command-L on the keyboard.

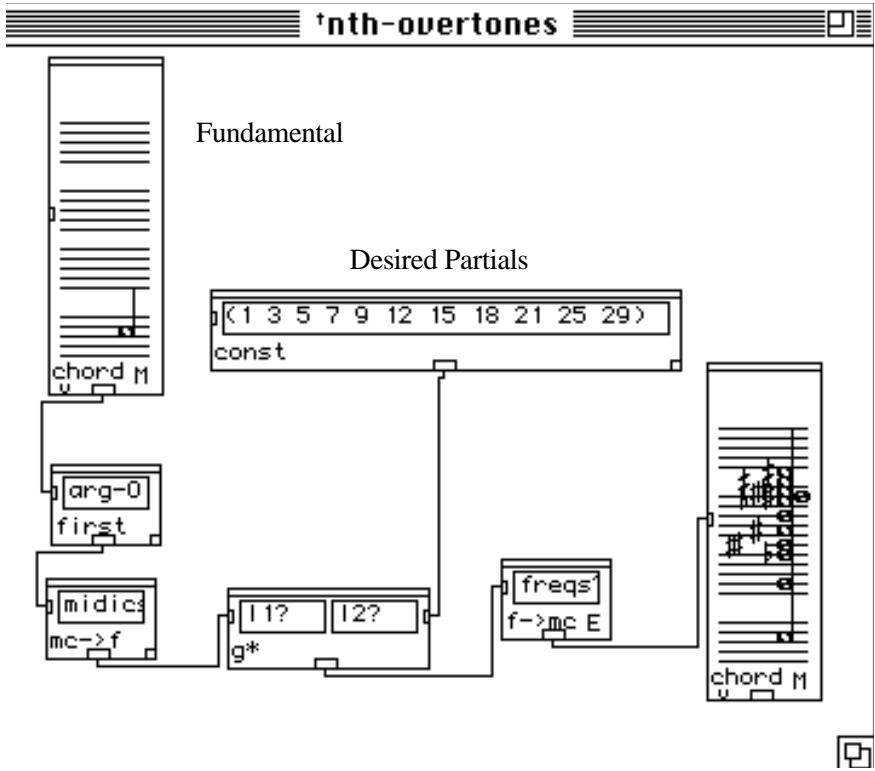
Another method of switching between the two applications is to click on an open window associated with either Lisp or PatchWork .

### Building Patches

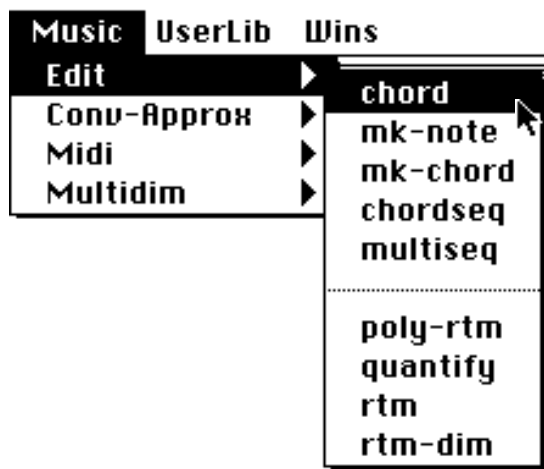
As we have already seen in the introduction (Basic elements of Patchwork), a patch is a group of interconnected modules within a graphic PatchWork window. This chapter looks at the process of building and saving a patch called **nth-overtones**.

# A Walk Through Nth Overtones

Here is an example of a patch, constructed by the composer Tristan Murail. It calculates the harmonic series symbolically. Users simply specify the fundamental and a list of the harmonic partials desired (by rank). You will find this patch in the Documentation folder; follow this path: **PW 2.5.1>Documentation>Sample Patches>Spectral Music>Désint. X.**



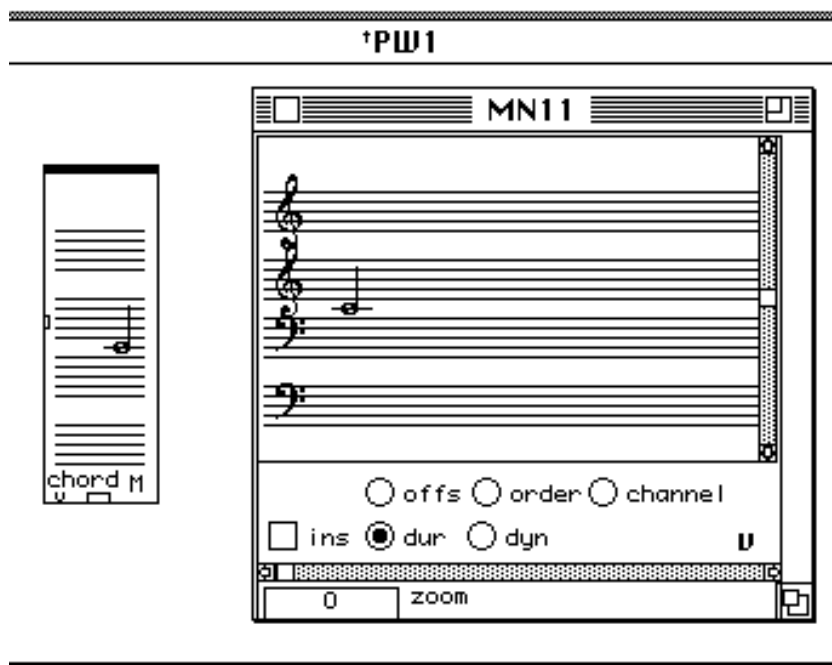
We will now rebuild this patch step-by-step, in order to demonstrate the basic mechanisms for building a Patch-Work patch. The first step in building an harmonic series is to choose the fundamental. In our case we will make that selection inside of the musical notation editor **chord** (select the module **chord** under the submenu **Edit** under **M u s i c** ) .



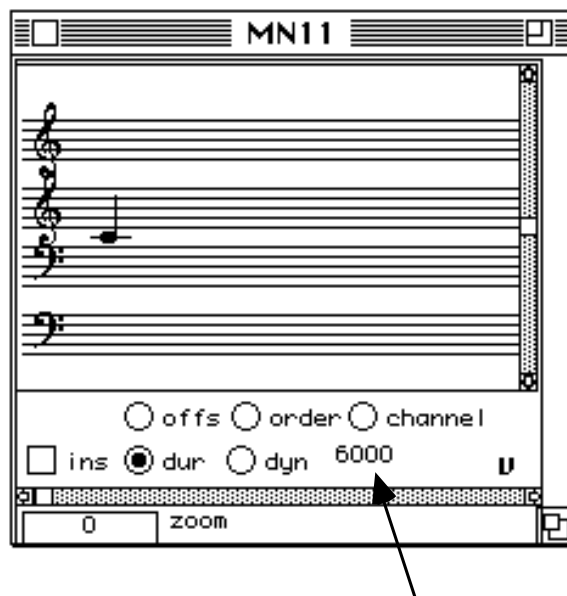


This causes the module to appear in the current Patchwork Window.

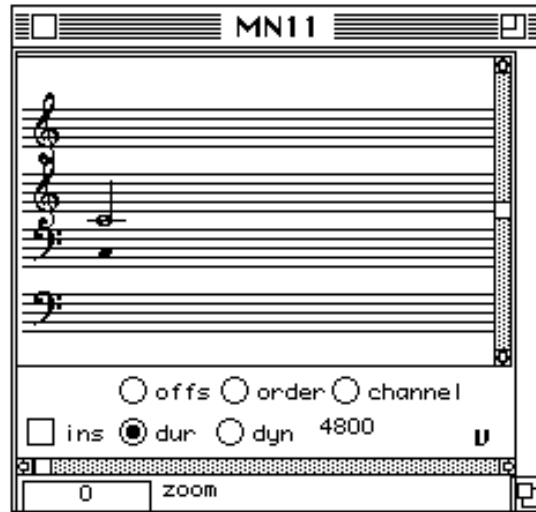
After this module has been loaded, select the module. To select a module use the standard Macintosh selection, clicking and dragging around it, and then type the letter **a**; this opens the associated editor window.



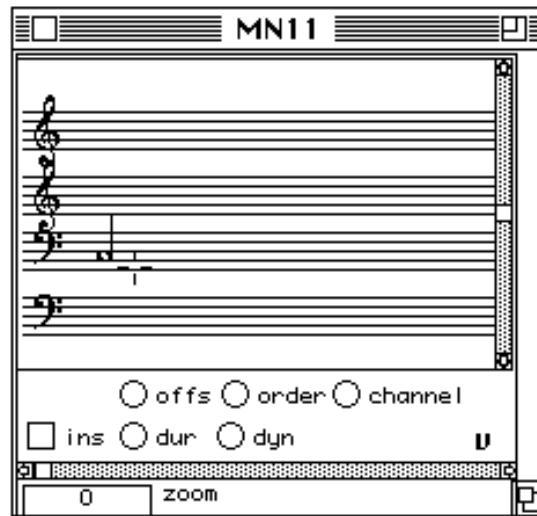
Position the cursor on the note C (the value by default) and click the mouse. This pitch can now be changed.



Once the cursor is positioned on a note, its value in midicents<sup>1</sup> appears at the bottom of the window, underneath the *channel* indication.



Moving the mouse (with the button depressed) changes the value.



After editing the fundamental to the desired value, the editor window can be closed by typing Return. You can tell if you have properly edited the module by evaluating it.

---

1. **Please note** : The midicent is the basic unit of pitch in PatchWork; it is derived from MIDI note numbers. The note middle C (C4 in the American system, C3 in the French, which is used in PatchWork) has the value 60 in the MIDI pitch scale; that same middle C would be represented as 6000 in midicents. In other words, 60 midi plus '00' (zero) cents. 6050 would represent middle C plus 50 cents (a quarter tone). The cent divides a semitone into one-hundred equal parts (equal along a logarithmic scale).

To evaluate a module, place the cursor on the small output rectangle, located at the bottom-center of the module, and Option-click!



The result of the evaluation is displayed in the Listener window:

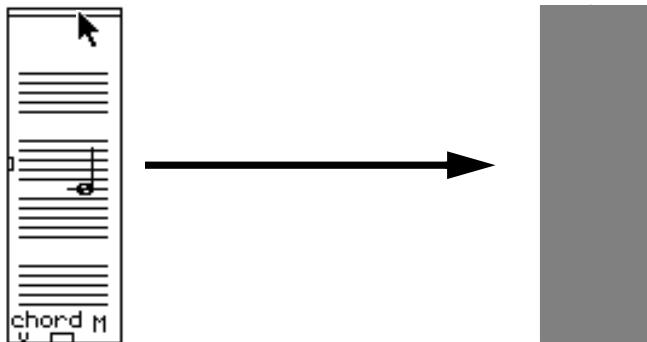


4800 is the midicents value corresponding to the note now shown in the chord module.

Always be aware of the type of the output. All inputs and outputs in PW have types and the default output type for this module is a list. (See the **PatchWork Programming Guide** for details on the typing scheme.)

Before we go any further, let's take a look at how modules are moved about the window.

To move a module: click on the upper band, move the mouse and then release the mouse button. If more than one module is selected, all the selected modules will move together. *ctrl click* anywhere within the body of the module will also allow the movement of a module.

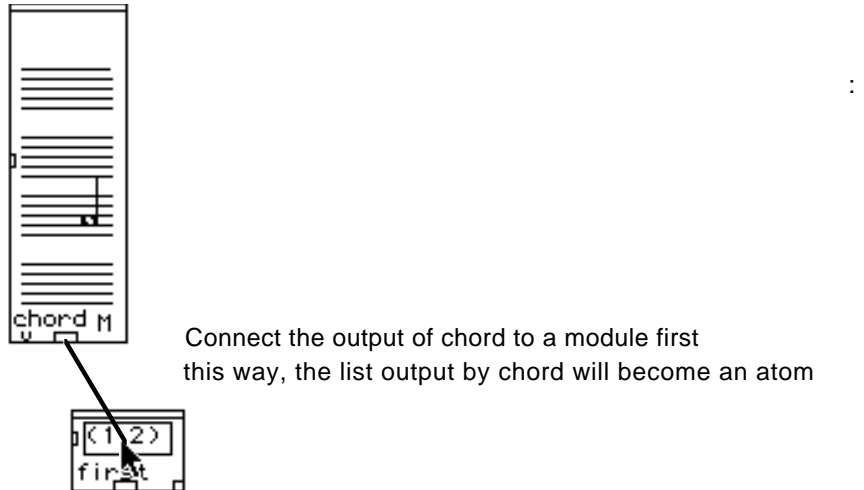


Although the list output from the 99 module contains just one element, we must, nevertheless, extract that element, so as to have simply a value. To do this, we connect the output of our **chord** module to the input of the module **first** (select the module **first** under the submenu **Lisp** under **Kernel**).

Kernel	Music	UserLib	Wins
Data			
Arithmetic			
Num-series			
Function			
control			
LISP			first

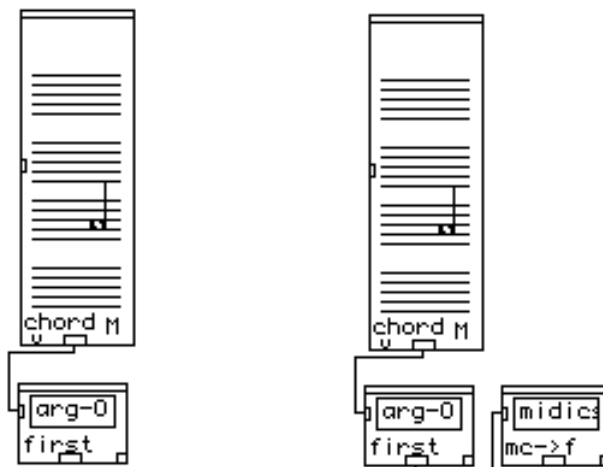
You must connect the output of chord to a module **first**; then the list output by chord, which appears as  $\rightarrow(4800)$  will become an atom:  $\rightarrow 4800$ . This operation is necessary because of the way the module **g\*** operates. In this case, we want to multiply the output of chord as atom by the list of partials, i.e. by each element of the partial list. This is how the module **g\*** works, and you might want to check your Reference manual.

To connect two modules, click in the output rectangle of a module and drag the mouse (keeping the button depressed) to the desired input rectangle:



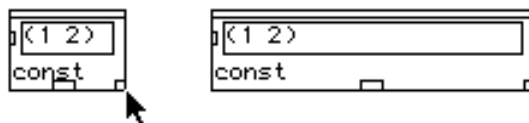
Attention! To cut a connection, Option-click in the input rectangle where the line is connected.

The output of the **first** will, then, be connected to the input of the module **mc->f** (select the module mc->f under the submenu **Conv-Approx** under **Music**) in order to convert our value of 4800 midicents (midi \* 100) into a frequency in hertz.



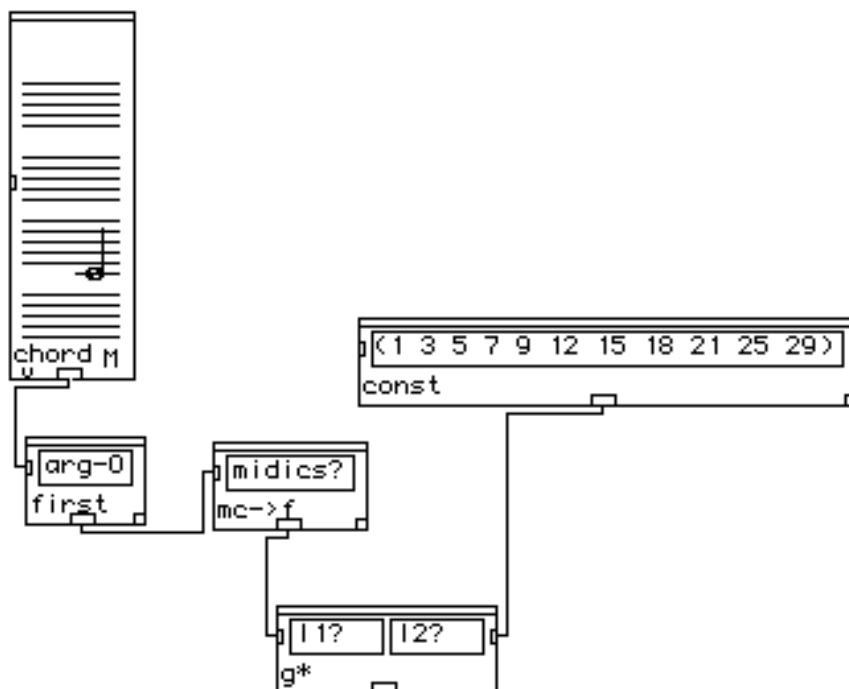
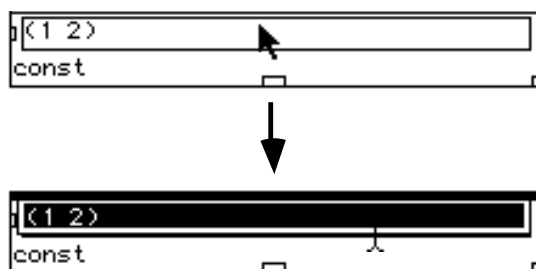
This value (our fundamental frequency) must now be multiplied by a list of whole numbers, corresponding to the harmonic partials we wish to calculate. Thus, we will connect the output of the **mc->f** module to one of the inputs of the module **g\*** (select the module **g\*** under the submenu **Arithmetic** under **Kernel**). The other input of this module must be connected to our list of partial rankings, to make this list we use the module **const** (accessed under the **Kernel->Data** menu). After having loaded the module **const**, drag it to its desired position, change its size and edit it.

The length of some modules can be changed. These modules have a small square in the bottom right corner. To change the size, click on the small square and (keeping the button depressed) drag the edge of the box to the desired size.



Double-click on the input rectangle; this opens the input box for editing, allowing the new value to be entered directly from the keyboard. To exit the edit mode and save the change, type Return.

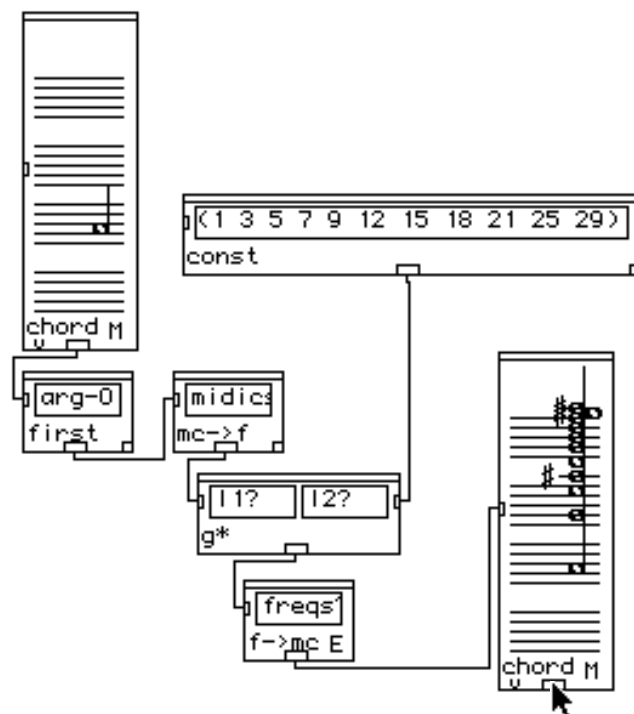
Attention. You should type into the module a list like this: (1 3 5 7 9 12 15 18 21 25 29)



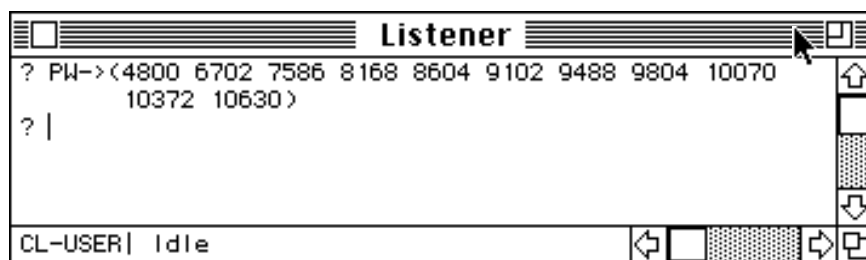
If we evaluate this patch (by Option-clicking on the output rectangle of the module "g\*"), we obtain, in the Listener window, a list of values, representing the frequencies (in Hertz) of the partials we are calculating:

```
? PW->(82.4068892282175 247.22066768465248 412.03444614108747
576.8482245975224 741.6620030539575
988.8826707386099 1236.1033384232624 1483.324006107915
1730.5446737925674 2060.1722307054374
2389.799787618307)
```

It is now necessary to convert this data into midicents, in order to be able to display it, in musical notation, within a **chord** module. Like in the previous steps, we will connect the output of the **g\*** module to the input of a **f-mc** module, which will, in turn, be connected to a second **chord** module.

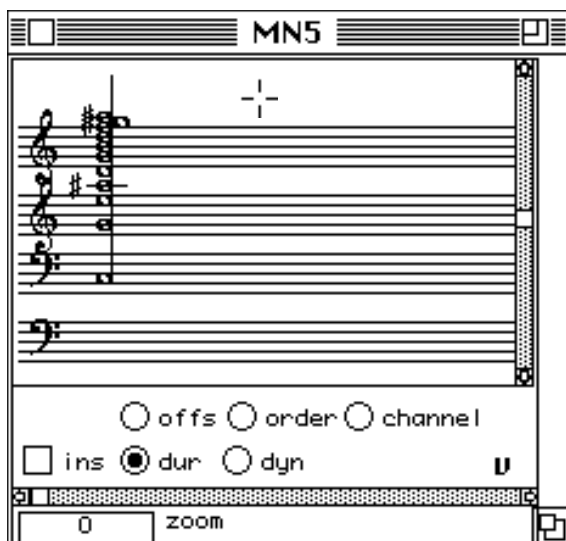


To evaluate this entire patch, simply Option-click on the output rectangle of the last module in the chain (in this case, the second **chord** module).

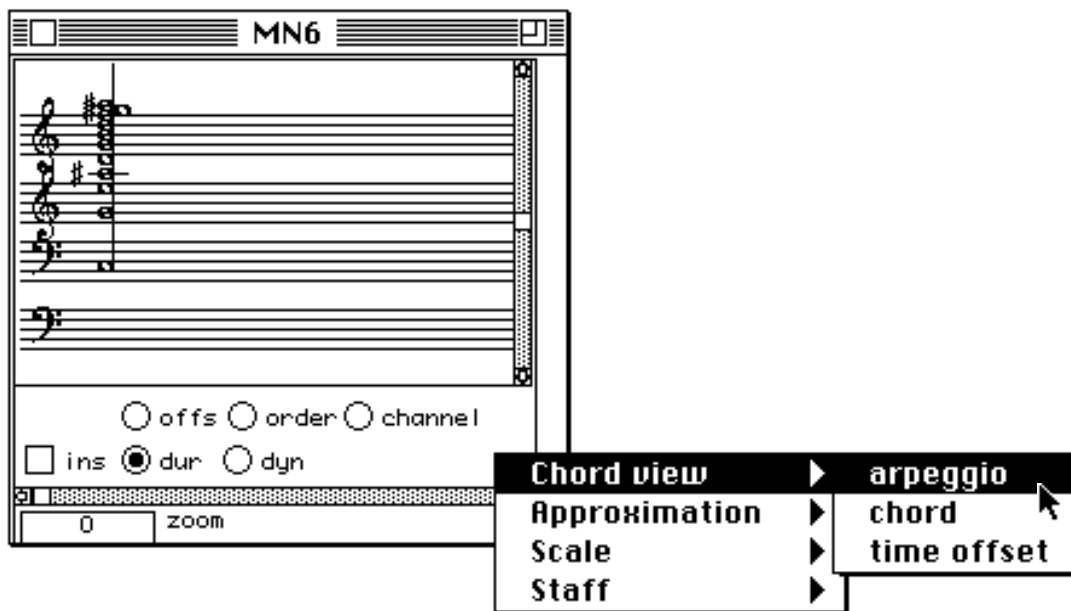


## To listen

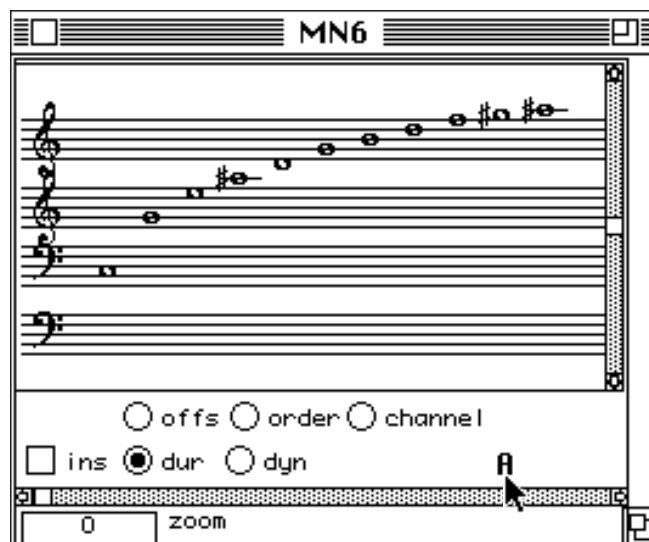
If your computer is hooked up to a synthesizer through MIDI, and if you have set up PatchWork for MIDI in the **PWoper>MIDI SetUp** menu, you can hear a harmonic series. In order to get this result, you should open the window of the second chord module: type o, and you will see the display of the series as a chord; type the letter p, and the chord will play through MIDI.



It is also possible to play the notes of this chord one by one. Make sure the window of the chord module is opened; click on the letter V which is located on the lower right hand side of the window, while keeping the mouse button pressed down.



Select the **arpeggio** option in the **Chord View** item. The chord is displayed as an arpeggio.



To hear it again, type *p* while the window is open.

To play micro-intervals

You are able to compute, visualize and hear the chord with an approximation of a quarter or an eighth of tone.

1) set up your synthesizer in multi-timbral mode;

2) Tune channels 1, 2, 3, and 4 an eighth of tone apart (i.e. 25 cents), in this fashion:

channel 1 : no special tuning;

channel 2 : 25 cents higher (i.e. one 1/8 tone);

channel 3 : 50 cents higher (i.e. one 1/4 tone);

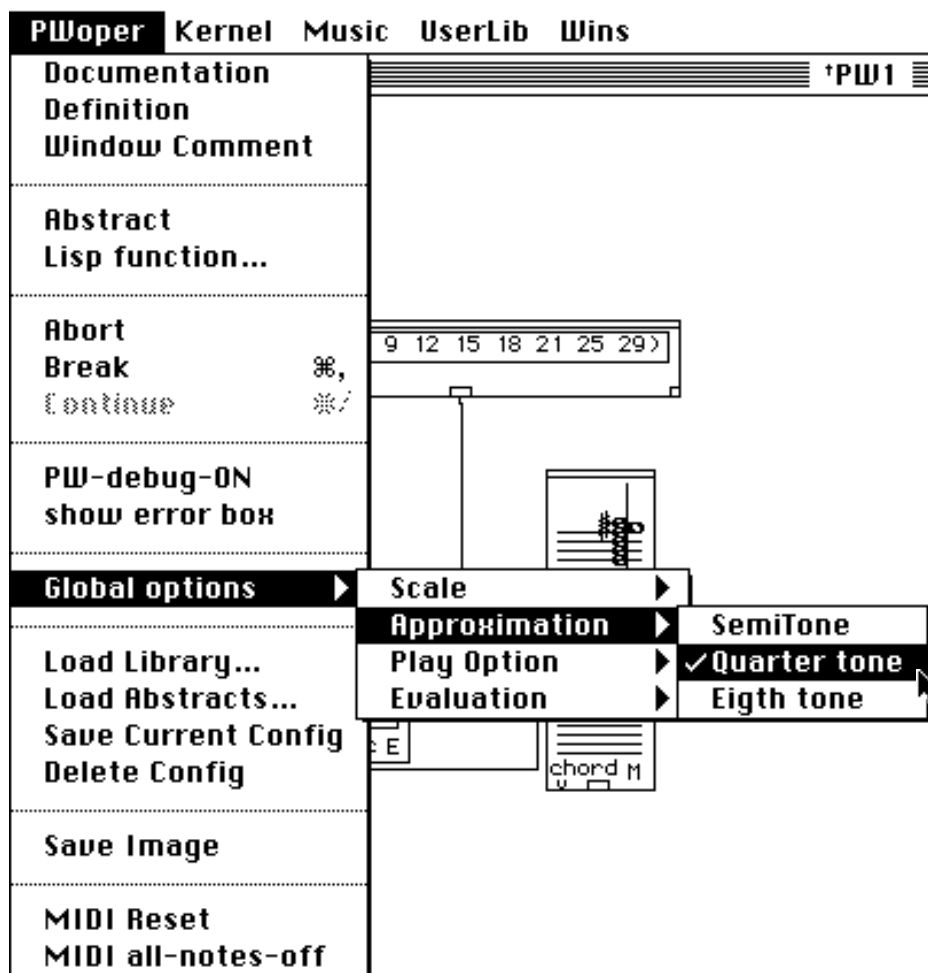
channel 4 : 75 cents higher (i.e. one 3/8 tone);

Re-evaluate the patch, using the approximation you have selected.

PatchWork's default approximation is the semi-tone. You may change it by choosing the **Quarter Tone** option in the **Approximation** item, which you'll find in the **Global Options** of **PWoper**. Still with me?

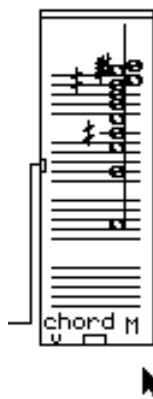


Here is the visual map of this operation:



When you select **Quarter tone**, all the pitch evaluations are made at that approximation, which will in tuen be used in the notation editors.

Let's get back to our patch. We need to re-evaluate it.



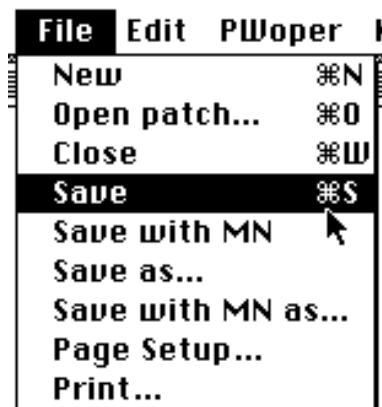
In the window of the **chord** module, notes are displayed with the selected pitch approximation.

To hear the result, either as a chord or as an arpeggio, follow the procedure described above, but this time using your synthesizer in a multitimbral mode, and after having set it up with the channels microtuning.

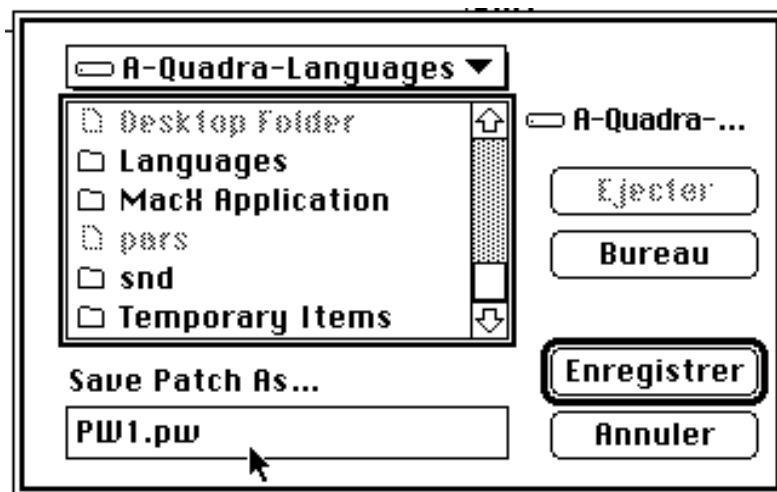
You can find this patch in the form of a module of the same name (**Esquisse->Freq harmony->Harm series->nth-harm**) in the Esquisse library. This is an example of a useful patch which can be turned into a module for general use.

## Saving and Opening Patches

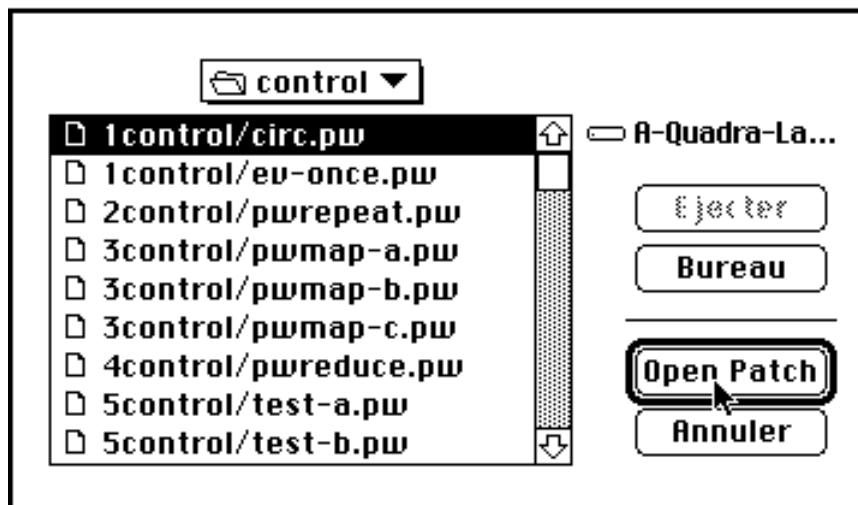
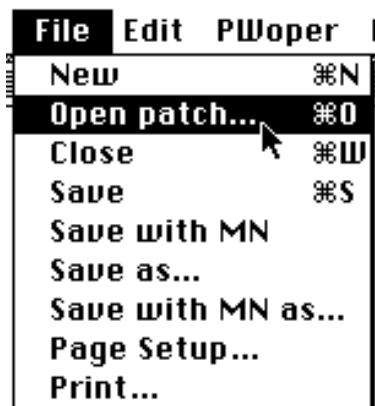
Once a patch has been constructed, you can save it by selecting **Save** or **Save with MN** (this option preserves the current contents of the musical notation editors, along with the patch) under the **File** menu.



Either selection invokes the following dialog box for naming the patch and placing it in the appropriate folder.



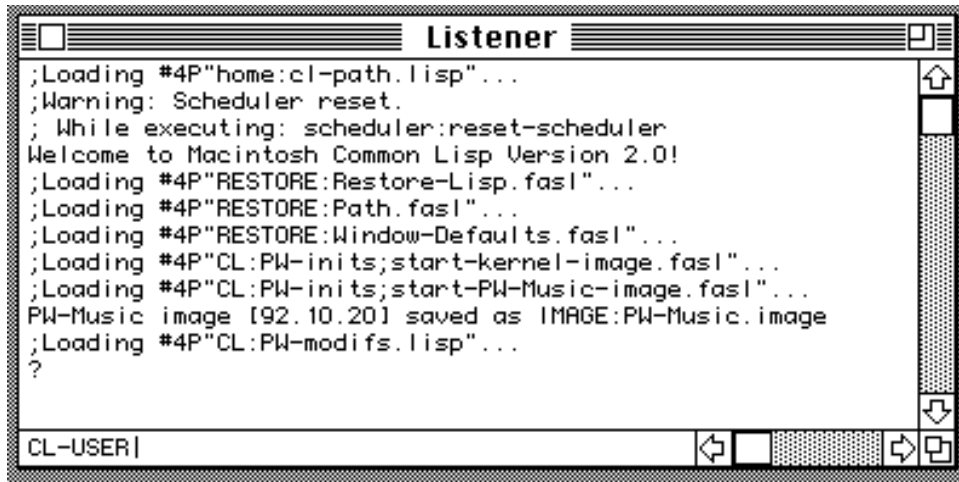
Use **Open Patch** under the **File** menu to load an existing patch:



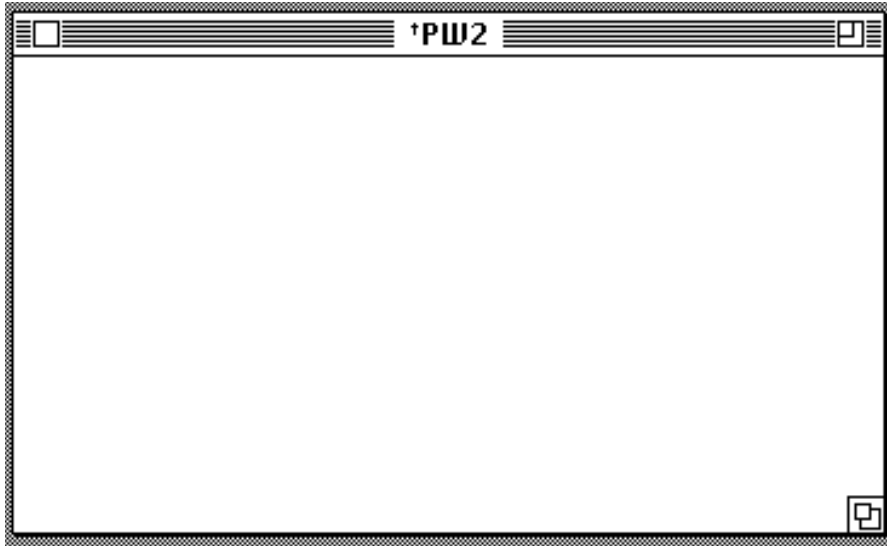
# 4 PatchWork Windows and Menus

This chapter describes the windows and menus of the PatchWork environment. Lisp and PatchWork have different types of windows. The Macintosh Common Lisp environment is characterized by the following types: the Fred window for text editing, where you write and run Lisp programs; documentation windows; and various windows and dialog boxes for advanced Lisp programming: inspector, browser, and trace.

The Listener window is a basic Lisp interaction window. It displays the results of the evaluation of PatchWork patches and Lisp programs.



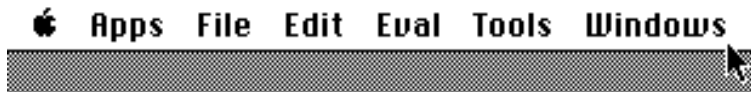
The characteristics of patch and subpatch windows, the help and documentation windows, as well as the windows for editing are documented below. Each window has a specific role and functionality; most also have their own menu bar. Moreover, the same menu can produce different results depending on what window is currently active (selected), this is indicated by the striped upper bar of the window.



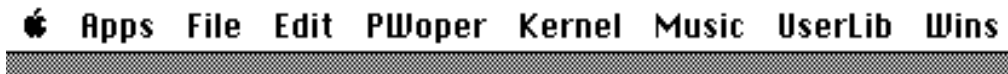
A PatchWork patch window ( $\uparrow$ PWxxx) does not have scroll bars; instead, there is an resize zone in the bottom right corner. (Once a window has been saved as a patch it takes the name under which it was saved.) It presents a graphic area in which patches — graphic representations of Lisp programs — can be assembled and run. This window has a small arrow sign ( $\uparrow$ ) preceding its name. This indicates that the window (or the corresponding patch) has been modified. The sign disappears as soon as the window is saved.

PatchWork supports two basic kinds of menu bars:

- the **Common Lisp** menu bar:



- the **PW** menu bar:



The **Lisp** menu bar is identical to the standard Common Lisp menu bar with the addition of the menu **Apps** which allows movement back and forth between : **Lisp**, **PW**, **BPF** (for breakpoint functions), **MN** (music notation editors) and **RTM** (rhythm editors). Since the other Lisp menus are standard, we do not describe them here. See the documentation of Macintosh Allegro Common Lisp.

# Quick Tour of the PatchWork Menu Bar

The PatchWork menu bar contains the following items:

 **Apps** **File** **Edit** **PWoper** **Kernel** **Music** **UserLib** **Wins**

---

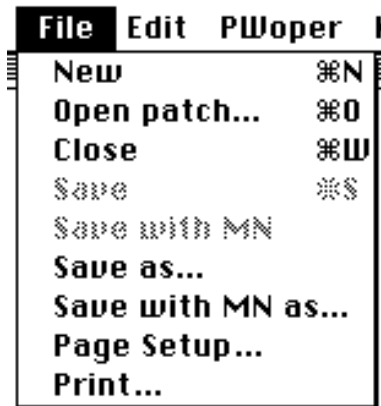
## ***Apps menu***

Contains options for switching between PatchWork, LISP, MN (music notation editors such as chord), the BPF windows (breakpoint functions) and the rhythm editor (RTM) windows.



## **File menu**

Contains all the basic operations concerning opening, saving and printing windows.



<b>New</b>	Create new PatchWork window
<b>Open patch</b>	Open existing patch, spatches are located with a dialog box
<b>Close</b>	Close current patch
<b>Save</b>	Save current patch as a file
<b>Save with MN</b>	Save current patch with contents of music notation editors
<b>Save as</b>	Rename current patch
<b>Save with MN as...</b>	Save and rename current patch with contents of music notation editors
<b>Page setup</b>	Specify printing parameters
<b>Print</b>	Print current patch window
<b>Edit</b>	Contains the standard Macintosh editing commands



PWoper	Kernel	Musi
Documentation Definition Window Comment		Documentation options
Abstract Lisp function...		Operations that create modules
Abort Break Continue	⌘, ⌘/	Interrupt operations
PW-debug-ON show error box		Debugging Operations
Global options	▶	Global options set certain aspects of global behavior
Load Library... Load Abstracts... Save Current Config Delete Config		Operations for loading libraries
Save Image		Operation that affects the PW program image
MIDI Reset MIDI all-notes-off MIDI SetUp		MIDI Operations

PWOper contains a series of operations specific to PW, divided into eight categories:

*Documentation options*

- Documentation** Brings up the on-line documentation of the selected module
- Definition** Brings up the Lisp file containing the definition of the selected module - this function is not available for with the standard distribution of PatchWork which is in the form of a compiled application and does not provide the source code. For versions containing the code or for libraries which are distributed with the source code, as well as for user defined functions, this function will operate as long as the code remains within the path which was present when it was conmpiled.
- Window Comment** Brings up a comment module that can be added to a patch.



## ***Operations that create modules***

- Abstract** creates a new module built from other modules
- Lisp function...** creates a module containing any standard Common Lisp function.

## ***Interrupt operations***

- Abort** returns the user to the top-level
- Break** interrupts an evaluation
- Continue** continues an evaluation that had been interrupted

## ***Debugging Operations***

- PW-debug-ON** activates the debugging
- show error box** selects the module which caused the error

## ***Global options set certain aspects of global behavior***

These options can also be set on a local basis within a PatchWork editor, overriding the global option.

- Scale** allows a choice of how chromatic alterations are displayed. Notes can be shown either as part of a chromatic scale or as alterations of the C major scale.
- Approximation** allows a choice of what pitch approximation is used for display and MIDI playback in the music notation editors. Choices for approximation are: semitone, quarter-tone and eighth-tone.
- Play Option** allows the choice of the method for MIDI playback of micro-intervals. The choices are *pitch-bend* or *multi channel* (four MIDI channels tuned successively higher by one-eighth tone).
- Evaluation** allows the manner in which modules are evaluated to be changed. The two possibilities are Option-click or simply click.

## ***Operations for loading libraries***

- Load Library...** loads a library
- Load Abstracts...** loads a library of abstractions
- Save Current Config** saves the current libraries so that they are loaded automatically the next time PW is opened.
- Delete Config** removes the menus from the current configuration.

## ***Operation that affects the PW program image***

- Save image**

## ***MIDI Operations***

- MIDI Reset** causes a reset of the PatchWork MIDI driver.
- MIDI all-notes-off** sends MIDI note-off messages to all MIDI equipment connected to PW.
- MIDI SetUp** PatchWork 2.1.5 is compatible with Apple Midi Manager, which means it can send and receive MIDI instructions to and from other applications running in the background as well as internal boards like Digidesign SampleCell. See the *PW-Midi-Manager* manual. You may still use the PatchWork MIDI driver: for this, choose the **Use PW driver** option, click OK, and again OK in the dialog box that appears. Do not choose this driver if you have another MIDI application running. You can use OMS (Open Music System by Opcode) as long as you allow "Non-OMS Applications" in the "OMS Midi Setup" window.

# Kernel

The core of PW; it contains all the basic modules (except those that deal specifically with musical objects) for calculating and manipulating symbolic structures, as well as those for the visualization and editing of data in the form of graphic tables.

Kernel	Music	UserL
Data		▶
Arithmetic		▶
Num-series		▶
Function		▶
control		▶
LISP		▶
List		▶
Set Operations		▶
Combinatorial		▶
Abs		▶
BPF		▶
Extern		▶
Multidim		▶

## Music

Contains all standard modules that are specifically musical, including modules for editing notes, rhythms and chords, modules that convert between different types of notation and the modules that manage MIDI data.



## UserLib

Receives the menus of different libraries; which are composed of modules relating to particular applications (for example the Esquisse library), or for the modules created by the user (these constitute a personal library).

## Wins

Shows all open PW windows and lets users select among them. Important! Wins does not give access to Lisp windows, it simply displays and allows movement between PW windows. However, the Windows menu in Lisp displays and allows movement between all open windows (Lisp or PW).

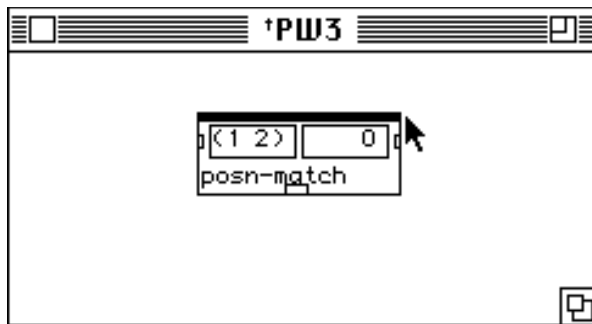
# The PWoper Menu Detailed presentation

We have already introduced this complicated menu, which contains operations specific to PW. Its menu items can be divided into eight categories, which we now look at in more detail.

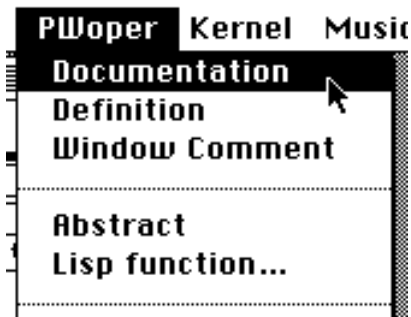
## Documentation Operations - Detailed Presentation

This first part of the menu contains three options: **Documentation**, **Definition**, and **Window Comment**, described next.

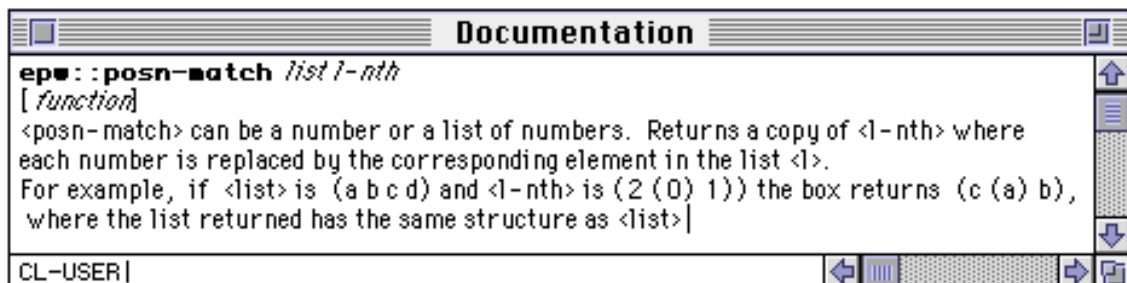
**Documentation** brings up the on-line documentation of the selected module. After selecting a module inside of a PW window (a blackened upper bar indicates that a module is selected),



select the option **Documentation**

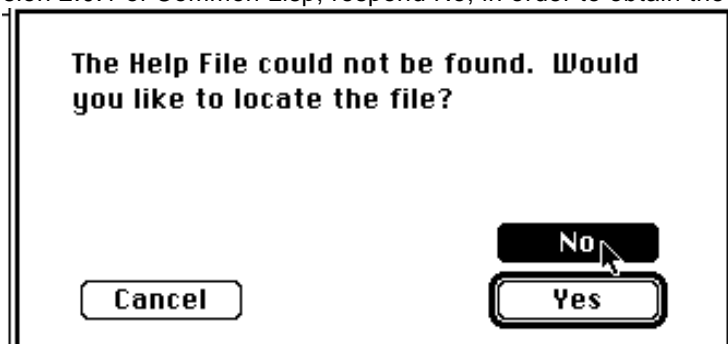


this opens a new window containing the on-line documentation. Selecting **Documentation** in the menu is equivalent to selecting a module and then typing *d*.

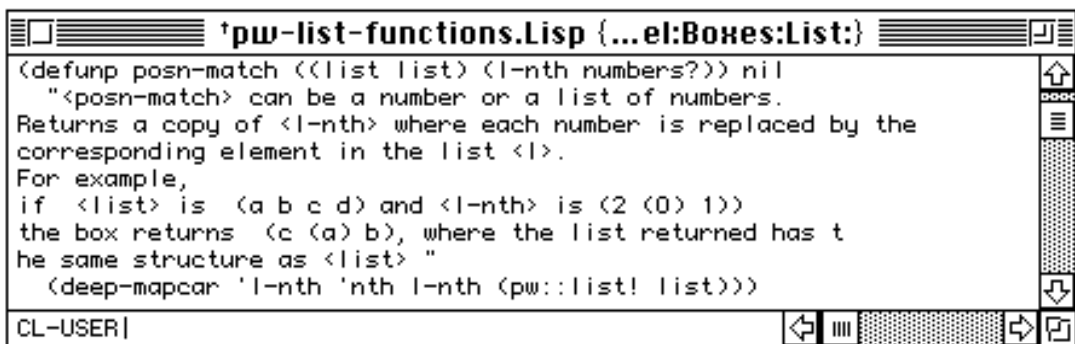
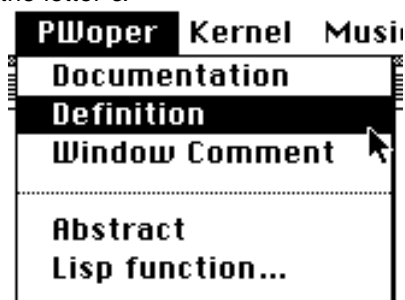


**Important:** If Common Lisp is not installed on your computer, the first time (during a given session) that on-line

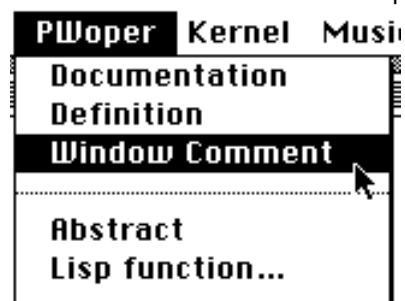
documentation is requested, a dialog box appears and asks whether you would like to locate the help file. As a result of certain problems in version 2.0.1 of Common Lisp, respond No, in order to obtain the documentation.

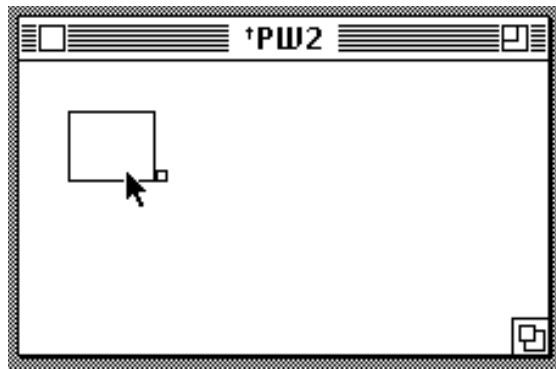


**Definition** brings up the Lisp file containing the definition of the selected module. The equivalent keyboard command is the letter *e*.



**Window Comment** brings up a comment module that can be added to a patch.





Double-click inside this module to edit its content :



this causes a black band to appear inside the module, the black band indicates that the module is ready to receive text.



This module works in exactly the same manner as other PW modules as far as selecting, moving and resizing are concerned.

## Operations that Create Modules - Detailed Presentation

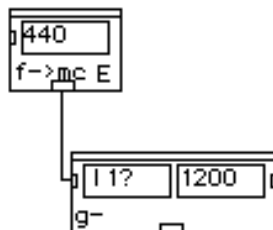
The **Abstract** option in the **PWoper** menu creates a new module containing many other modules. In PatchWork, an abstraction is simply a patch which is, itself, contained within a module. Thus an abstraction is a way to condense your patches and modularize your programming. The rest of this section explains how you can create your own abstraction.

### Creating an Abstraction

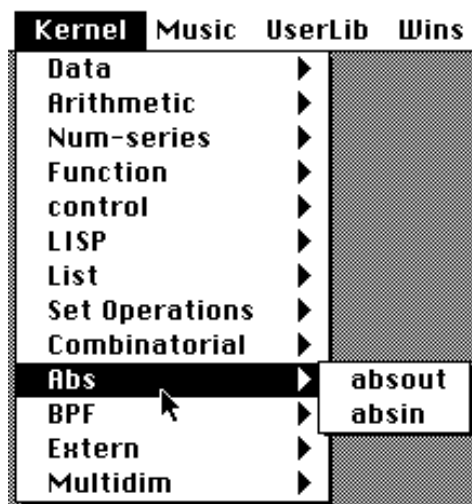
To create an abstraction the following elements are needed:

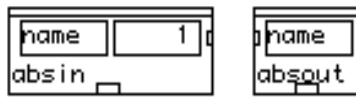
- A patch, in other words, the function that is going to be abstracted into the form of a module.
- From zero to many **absin** modules. These modules permit data or parameters to be entered into the abstraction.
- One **absout** module. This module represents the output of the abstracted module, this is the module that allows the results of the function within the abstraction to be used in the patch in which it is placed.

We will describe a patch that converts frequencies in Hz into *midicents* and then transposes the results one octave lower. A midicent is PatchWork's data type for pitch. MIDI divides the pitch gamut into the range of semitones [0, 127]. A midicent divides each semitone by 100. Thus the MIDI note 60 (C3) becomes midicent 6000. As mentioned earlier, PW's notation follows the French system in which middle C is named C3.



Suppose that you need to use this patch in different places; we can make this patch into an abstraction. To do this we must first add an **absin** module for entering the frequency to be converted. We then must add an **absout** module (this module must always be present in an abstraction) which will output the result. We should specify that the **absin** and **absout** modules are located in the Kernel menu under **Abs**

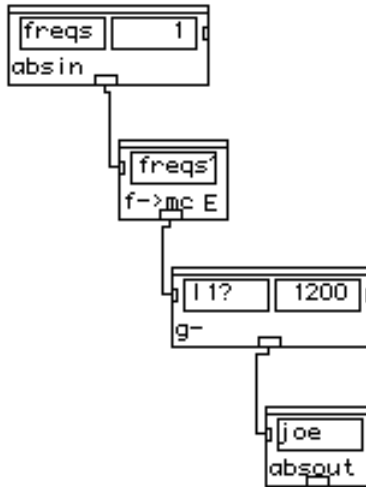




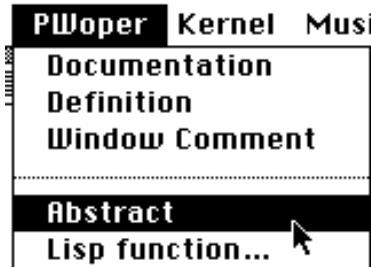
Note that the two **abs** module contain the argument *name*. For **absin**, *name* corresponds to the name of the input parameter, as it appears in the input box of the module to which the **absin** is connected (within the abstraction). The name given for the **absout** argument becomes the name of the abstraction.

Problems (as well as errors) can occur in the following two situations:

- (a) if the same name is given to two different arguments, or
- (b) if two different **absin** modules are connected to two modules that have the same name for their inputs. Special names can, however, be used for these arguments by editing the *name* input. See the section on *Editing Input Parameters*.

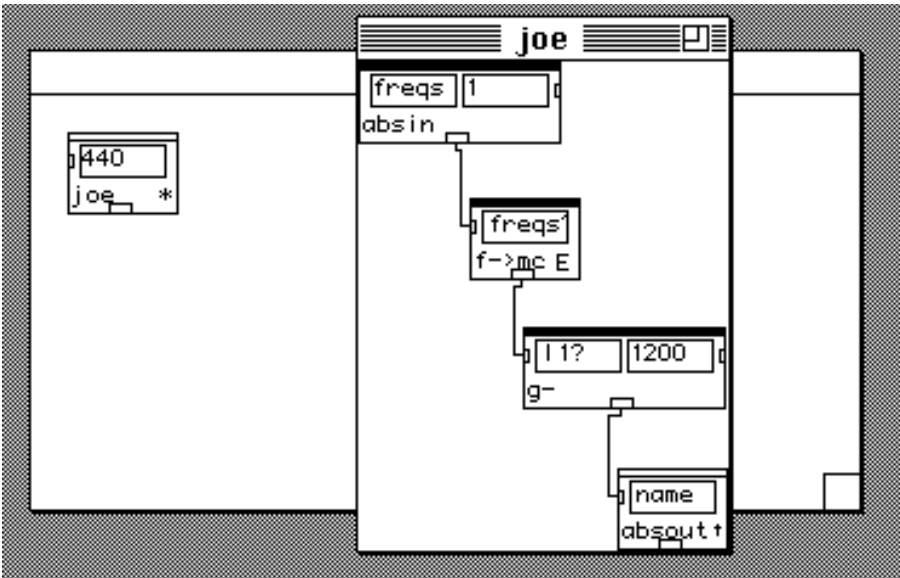


After adding the **absin** and **absout** modules necessary for the patch, you must select the entire patch and then choose the option **Abstract** in the menu **PWoper**.

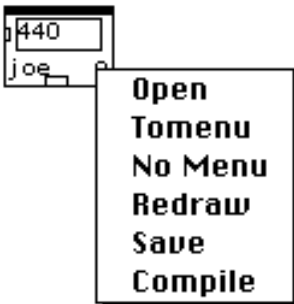




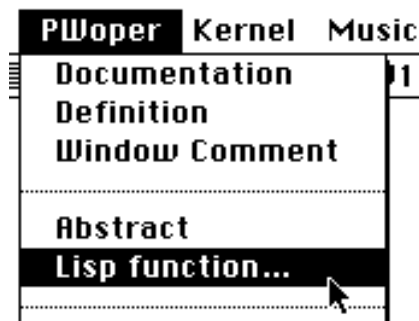
The following result appears in the Patchwork window.



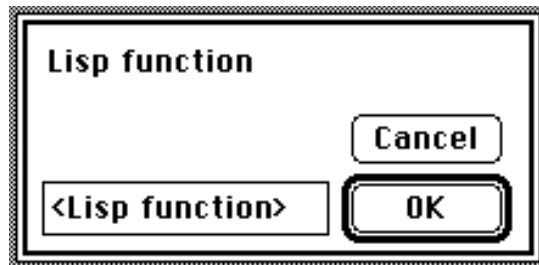
It is now necessary to close the window joe by pressing Return. Clicking on the A in the lower right corner of the module causes this menu to appear:



<b>Open</b>	open the abstracted patch for editing (or consultation)
<b>To menu</b>	add the abstracted module to the user's library under the UserLib menu.
<b>No Menu</b>	remove the module from the UserLib menu.
<b>Redraw</b>	redraw the module.
<b>Save</b>	save the module as a file.
<b>Compile</b>	transform the abstraction into the equivalent of a single Lisp function (when possible).
<b>Lisp function...</b>	creates a module representing any standard Common Lisp function.



a dialog box appears that asks for the name of the Lisp function.



This can be the name of any desired standard Lisp function or the name of any Lisp function that the user has defined. A module representing this function is added to the active PatchWork window. It is normally preferable to define new functions using the function **defunp** (see the documentation in the *PatchWork Programming Guide*).

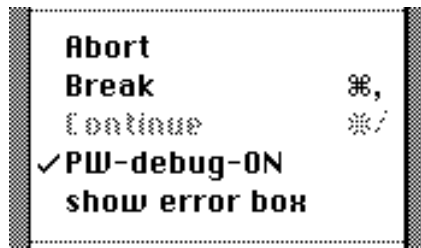
Note: Although any keywords are acceptable (&rest &optional &aux &key &key\* &allow-other-keys &body &environment &whole), functions that use the keywords &rest and &optional are a special case. These keywords are the only ones that the PatchWork typing scheme recognizes. Modules with other keywords can cause problems when used in abstractions. A good rule to follow is to always pass the inputs of those modules through the data module `const`. They must never be directly connected to the module `absin`.

## ***Interrupt Operations - Detailed Presentation***

<b>Abort</b>	stops the execution of a Lisp process. This stops the execution of any PatchWork patch; it is important for returning to the top-level after having entered the error mode.
<b>Break</b>	interrupts the execution of a Lisp process. This can be used to interrupt the execution of any PatchWork patch. Execution can be continued with the command <b>Continue</b> or stopped altogether with the command <b>Abort</b> .
<b>Continue</b>	continues the execution of a Lisp process or PatchWork patch that was previously interrupted.

## Debugging Operations - Detailed Presentation

The PW-debug-ON option puts PW into debugging mode



A checkmark to the left of the option indicates that it has been selected. In this mode, after performing an evaluation that causes an error, the option Show error box can be selected, this will show which module was responsible for the error (the upper band of the offending module will turn black). To exit the debug mode (which slows down the evaluation of patches, considerably), select the option **PW-debug-ON** a second time.

**Show error box** can be used when PW is in debug mode to select the module that produced an error.

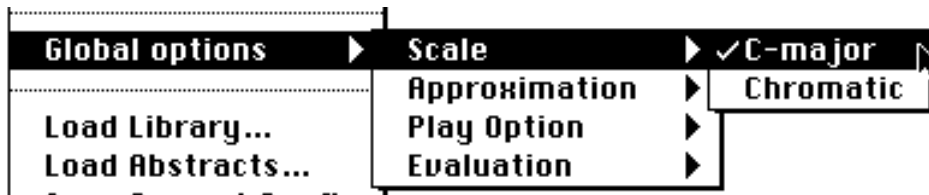
# Options that Determine PatchWork's Behavior - Detailed Presentation

**Global** contains four subitems that alter PatchWork's overall or global behavior: **Scale**, **Approximation**, **Play**, and **Evaluation**. **Scale**, **Approximation**, and **Play** options, however, can also be altered locally. In other words, if many editors are present within the same PatchWork window, it is possible to assign different display, approximation, and playback behaviors to each one.

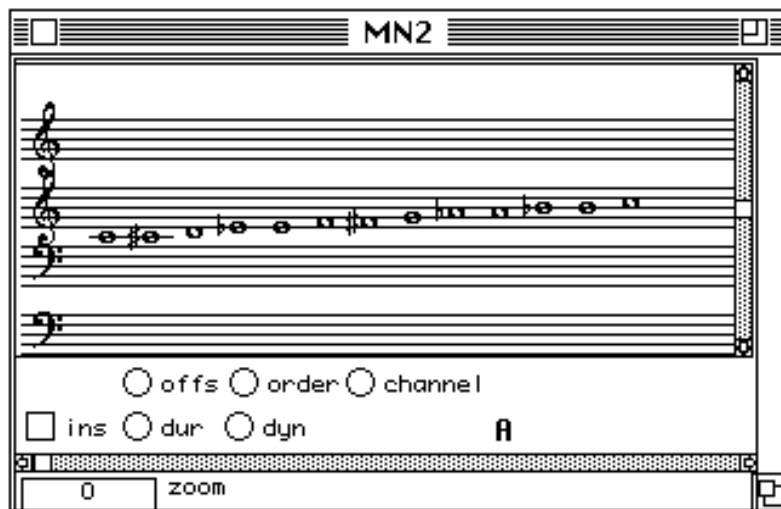
**Scale** allows the choice of how chromatic alterations are displayed. Notes can be shown either as part of a chromatic scale or as alterations of the C major scale.

To demonstrate these options we look at the example of a chromatic scale displayed in each mode:

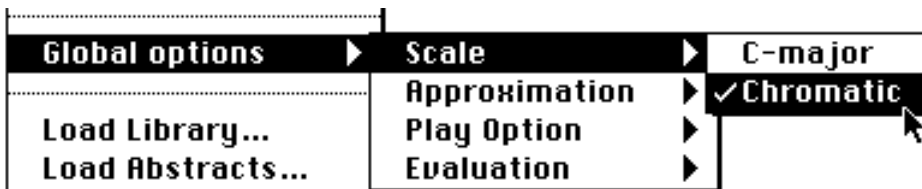
With the option **C-major**



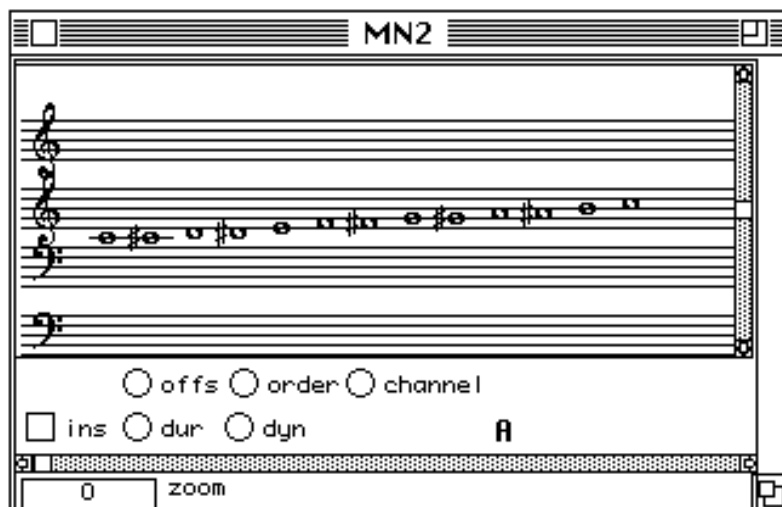
the result is :



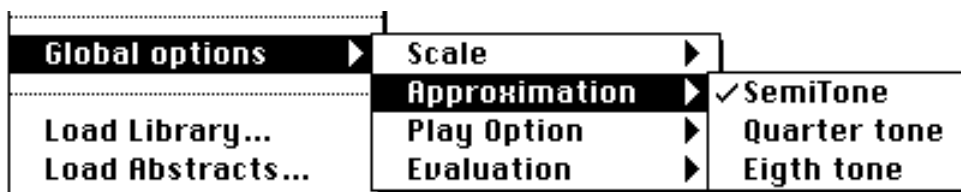
and with the option **Chromatic**



the result is :

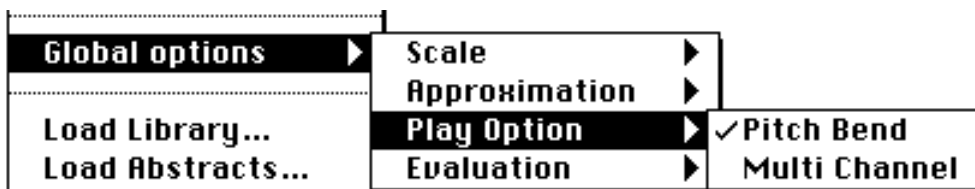


Approximation selects the pitch approximation (frequency quantization) to be used for the display and MIDI playback within the music notation editors. Choices for approximation are: **Semitone**, **Quarter tone** and **Eighth tone**.



## Play

selects the method for MIDI-playback of micro-intervals. The choices are pitch-bend or multi channel (four Midi channels tuned successively higher by one eighth of a tone).



Evaluation determines the manner in which modules are evaluated to be changed. The two possibilities are Option-click or simply click on the small output rectangle.



## ***Operations for Loading Libraries - Detailed Presentation***

- Load Library...** calls up a Macintosh dialog box where the user can choose the library-file that will be loaded. If the file has been properly configured (see the *PatchWork Programming Guide*) the menus corresponding to the library appear under the menu **UserLib**. Always load a .lib file.
- Load Abstracts...** calls up a Macintosh dialog box in which the user can choose the folder of abstractions to load. To select a folder, place the cursor on the folder and then choose the option Select-current-folder (in the dialog box). This loads all the abstraction files from the selected folder. In addition to the abstracts themselves, the command **Load Abstracts...** loads the names and structure of all folders contained within the selected folder; they will appear under the menu **UserLib**.
- Save Current Config** Selecting this option saves the current configuration of libraries so that they are loaded automatically the next time PW is opened.
- Delete Config** Selecting this option removes the menus from the current configuration.

## ***Operations on the PatchWork Image - Detailed Presentation***

- Save image** saves the current configuration of libraries so that they will be loaded automatically the next time PW is opened. The difference between this option and **Save Current Config**, is that **Save image** physically saves the application in its current state, where as **Save Current Config** saves only pointers which contain the addresses of the library files.

Important: It is impossible to erase the current PatchWork image. You must change the name of the old image before it can be erased.

## ***MIDI Operations -Detailed Presentation***

- MIDI Reset** causes a reset of the PatchWork MIDI driver. This can be useful when (under Multifinder) applications, other than PW, are used which define their own MIDI drivers (thus erasing PW's).
- MIDI all-notes-off** sends MIDI note-off messages to all MIDI equipment connected to PW.
- MIDI SetUp** You may now choose between the PatchWork MIDI driver and Apple Midi Manager (which must be installed in your system's extension folder). You may also use OMS (Open Music System by Opcode). Please refer to the *PW-Midi-Manager* manual.



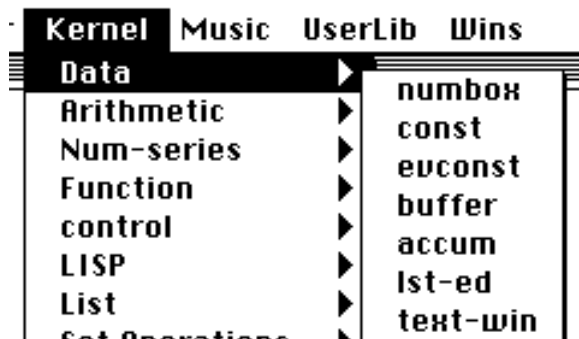
# The Kernel and Music Menus

This section offers a fast overview of the modules found under the Kernel and Music menus. Detailed descriptions of each module can be found in the online documentation , accessed by selecting the appropriate module and typing *d*, and in the PatchWork Reference Manual

## The Kernel Menu

The kernel menu includes the modules containing standard Lisp functions; the general modules for generating, manipulating, displaying and editing data; and the modules for controlling programs.

### Data



Contains modules for entering and buffering data, displaying and editing matrixes, and loading and writing text files.



<b>Arithmetic</b>	▶	<b>g+</b>
<b>Num-series</b>	▶	<b>g-</b>
<b>Function</b>	▶	<b>g*</b>
<b>control</b>	▶	<b>g/</b>
<b>LISP</b>	▶	<b>g-power</b>
<b>List</b>	▶	<b>g-exp</b>
<b>Set Operations</b>	▶	<b>g-log</b>
<b>Combinatorial</b>	▶	
<b>Abs</b>	▶	<b>g-div</b>
<b>BPF</b>	▶	<b>g-mod</b>
<b>Extern</b>	▶	<b>g-round</b>
<b>Multidim</b>	▶	<b>g-floor</b>
		<b>g-ceiling</b>
		<b>g-abs</b>
		<b>g-min</b>
		<b>g-max</b>
		<b>g-random</b>
		<b>g-average</b>

Contains simple arithmetic functions (as well as other simple functions such as logarithm, power, scaling, etc.) that can be applied to numbers, lists of numbers or matrices.

Kernel	Music	UserLib	Wins
Data	▶	PW2	
Arithmetic	▶		
<b>Num-series</b>	▶	arithm-ser	
Function	▶	geometric-ser	
control	▶	fibo-ser	
LISP	▶		
List	▶	g-scaling	
Set Operations	▶	g-scaling/sum	
Combinatorial	▶	g-scaling/max	
Abs	▶	interpolation	
BPF	▶	g-alea	
Extern	▶	x->dx	
Multidim	▶	dx->x	
		prime-ser	
		prime-factors	
		prime?	

Contains modules which generate numeric series and perform special numeric operations on existing lists.

Kernel	Music	UserLib	Wins
Data	▶		PW1
Arithmetic	▶		
Num-series	▶		
Function	▶	make-num-fun	
control	▶	sample-fun	
LISP	▶	lagrange-fun	
List	▶	linear-fun	
Set Operations	▶	power-fun	
Combinatorial	▶		
Abs	▶	g-oper	
BPF	▶	cartesian	
Extern	▶	inverse	

Contains modules that generate functions. The outputs of these modules can be connected to the **sample-fun** module (in the Function menu) in order to sample the values of the function, thus producing a numeric series. This menu also contains several modules that accept functions as input arguments (**g-oper**, **cartesian** and **inverse**).

*control*

Kernel	Music	UserLib	Wins
Data	▶		
Arithmetic	▶		
Num-series	▶		
Function	▶		
control	▶		
LISP	▶		
List	▶		
Set Operations	▶		
Combinatorial	▶		
Abs	▶		
BPF	▶		
Extern	▶		
Multidim	▶		

circ

ev-once

pwrepeat

pwmap

pwreduce

test

trigger

Contains modules that control programs: retrieving a single value from within a list, controlling loops, testing results, etc.

Kernel	Music	UserLib	Wins
Data	▶		
Arithmetic	▶		
Num-series	▶		
Function	▶		
control	▶		
<b>LISP</b>	▶		first
List	▶		rest
Set Operations	▶		butlast
Combinatorial	▶		reverse
Abs	▶		length
BPF	▶		mapcar
Extern	▶		list
Multidim	▶		apply
			funcall
			remove

Contains modules that perform standard Lisp functions.

<b>List</b>	▶	posn-match
<b>Set Operations</b>	▶	last-elem
<b>Combinatorial</b>	▶	x-append
<b>Abs</b>	▶	
<b>BPF</b>	▶	flat
<b>Extern</b>	▶	flat-once
<b>Multidim</b>	▶	flat-low
		create-list
		expand-list
		rem-dups
		list-modulo
		list-explode
		mat-trans
		list-filter
		table-filter
		range-filter
		band-filter
		interlock
		subs-posn
		group-list
		first-n
		last-n

Contains modules that generate and perform operations on lists. Some of these modules use numeric lists as arguments, while others work with groups of objects.

## Set Operations

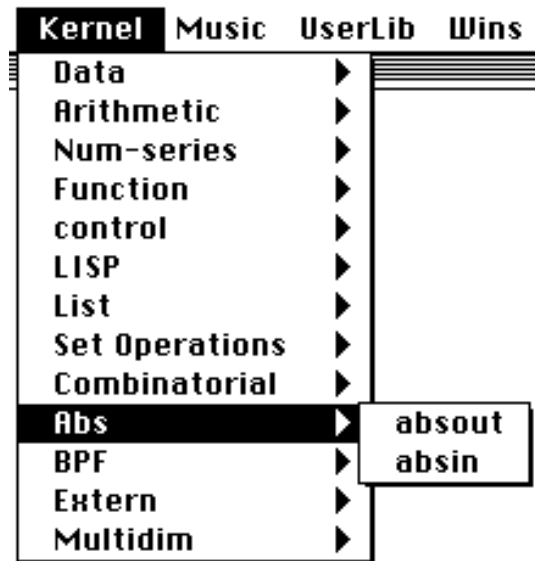
Kernel	Music	UserLib	Wins
Data	▶		
Arithmetic	▶		
Num-series	▶		
Function	▶		
control	▶		
LISP	▶		
List	▶		
<b>Set Operations</b>	▶	x-union	
Combinatorial	▶	x-intersect	
Abs	▶	x-xor	
BPF	▶	x-diff	
Extern	▶	included?	
Multidim	▶		

Contains modules that perform set operations on lists.

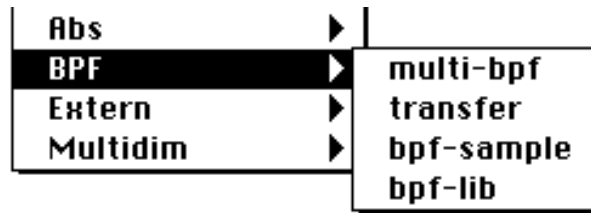
Kernel	Music	UserLib	Wins
Data	▶	↑PW1	
Arithmetic	▶		
Num-series	▶		
Function	▶		
control	▶		
LISP	▶		
List	▶		
Set Operations	▶		
Combinatorial	▶	sort-list	
Abs	▶	posn-order	
BPF	▶	permut-random	
Extern	▶	permut-circ	
Multidim	▶	nth-random	

Contains modules that perform combinatorial operations on lists. These operations effect either the ordering of the list, or choose random elements from a list.





Contains modules that build the inputs and outputs of abstractions.

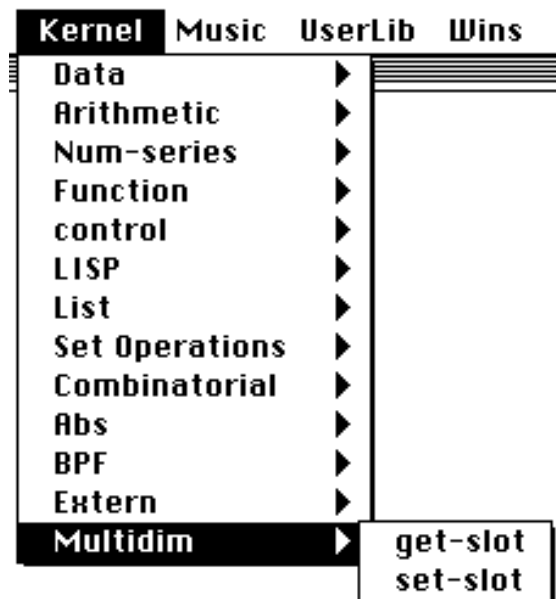


Contains modules for generation and display of breakpoint functions; and for loading libraries of breakpoint functions.

**Extern**

Kernel	Music	UserLib	Wi
Data	▶		
Arithmetic	▶		
Num-series	▶		
Function	▶		
control	▶		
LISP	▶		
List	▶		
Set Operations	▶		
Combinatorial	▶		
Abs	▶		
BPF	▶		
Extern	▶	in	
Multidim	▶	out	

Contains the modules that permit communication between patches through the use of global variables.



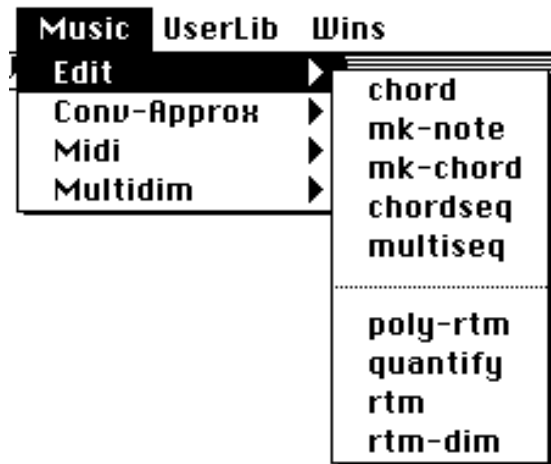
Contains modules that access the internal fields of objects.

# The Music Menu

The **Music** menu contains the options and submenus specifically designed for music.

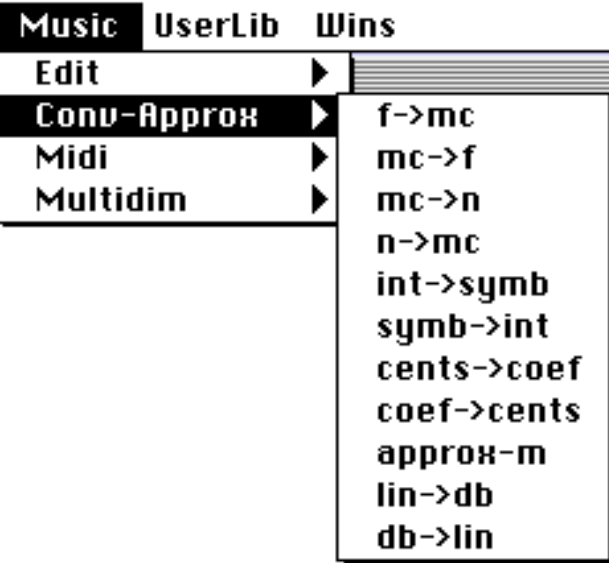


## *Edit*



Contains modules for displaying and editing music. This includes editors for sequences of notes, sequences of chords and polyphonic sequences.

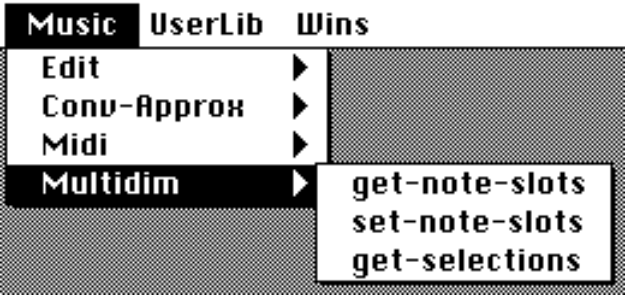
# Conv-Approx



Contains modules for the conversion of values between different units and for the approximation of notes in *midicents*. The midicent is a unit of pitch derived from the MIDI note numbers. The note C3 (middle C) has the value 60 in MIDI; that same C3 would be represented as 6000 in midicents. In other words, 60 midi plus '00' (zero) cents. 6050 would represent C3 plus 50 cents (a quarter tone). The cent divides a semitone into one-hundred equal parts (equal along a logarithmic scale).

Music	UserLib	Wins
Edit	▶	
Conv-Approx	▶	
Midi	▶	play-chords
Multidim	▶	play/stop
		play-object
		midi-o
		pgmout
		bendout
		volume
		delay
		microtone
		raw-in
		note-in
		chord-in
		status
		midi-chan
		midi-opcode
		data1
		data2

Contains modules for sending, receiving, and parsing MIDI data.



These modules are a special case of the **Multidim** modules in the **Kernel** menu. They allow access to the internal fields of objects, the difference between this menu and the **Multidim** menu under **Kernel** is that these modules return only data concerning the notes.

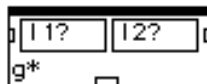


# 5 PatchWork Modules

This chapter explains the basics of PatchWork modules, including practical aspects such as invoking modules, opening abstraction, invoking menus, adding optional inputs, connecting modules, displaying default values, selecting and moving modules, changing their size, and evaluating modules.

## What is a Module<sup>1</sup>?

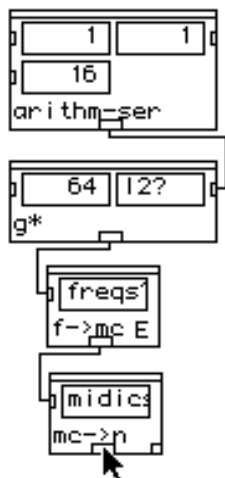
A PW module is just a graphic representation of a Lisp function. Take the example of the PW module **g\***:



This module calculates the product between two numbers (or between the elements of two lists). It is, in fact, a Lisp function:

```
(defunp g* ((l1? numbers?) (l2? numbers?)) numbers?
  "product of two numbers or trees. See tutorial 1arithmetic.pw"
  (arith-tree-mapcar (function *) l1? l2?))
```

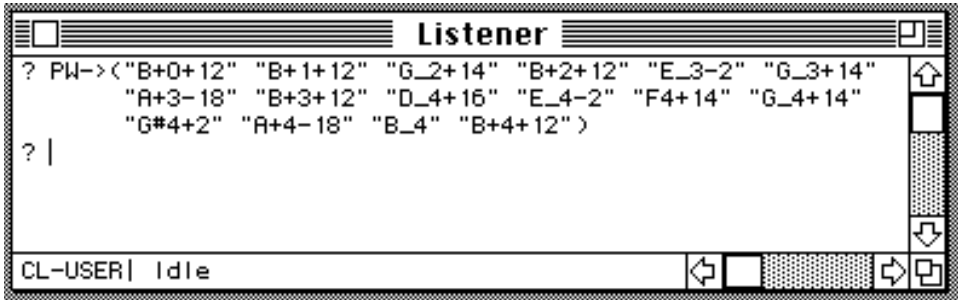
Modules can be interconnected to transmit data. The following example calculates the first 16 partials of the harmonic series with a fundamental of 64 Hz. These partials are then converted into symbolic notation for the output.



---

1. Modules are also called "boxes".

After evaluating the patch, (Option-click on the output rectangle), the following result appears in the Listener window :



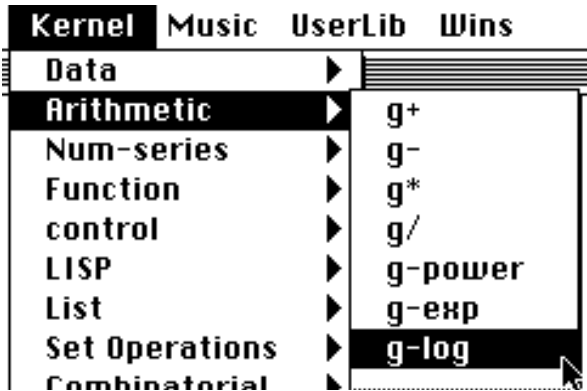
Here we begin to see some of the advantages of PatchWork

- Graphical representations of Lisp functions and objects (in the computer science sense of objects)
- Manipulation of parameters for functions, notably through the use of the specialized editors
- Interconnecting modules — a graphical approach to programming
- Programming a hierarchy of functions — abstractions and user-defined modules

Before entering into the practical aspects of manipulating PW modules, we take a moment to explain a point of prime importance: the relationship between Lisp object types and PatchWork interconnections.

### Invoking Modules from PatchWork Menus

To invoke a module from any of the three libraries (**Kernel**, **Music** or **UserLib**); place the cursor on the desired menu (**Kernel**, for example), press the mouse button and drag the cursor (keeping the button depressed) until you reach the desired module (for example **g\***, under the **Arithmetic** submenu), then release the mouse button (this causes the name of the module to blink). The module appears in the current PW window in the top left. Now select the module **g-log** under the submenu **Arithmetic** under **Kernel**:



This causes the module to appear in the PW window. PW 2.5 will display this :

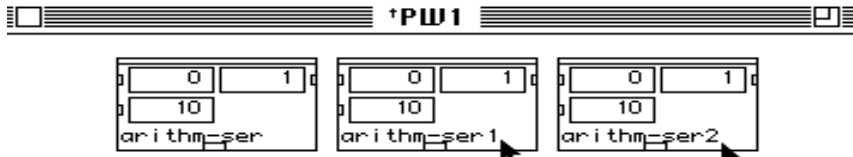


Drag then click at the desired location in the PW window.



## ***Multiple instances of a module***

When loading several identical modules, an index number is added to the name of the module:



The numbering changes according to the addition or the suppression of a module.

Caution: the mere duplication of a module (using the key combination Command-d for instance) does not create or increment an index.

# Structure and Behavior of PatchWork Modules

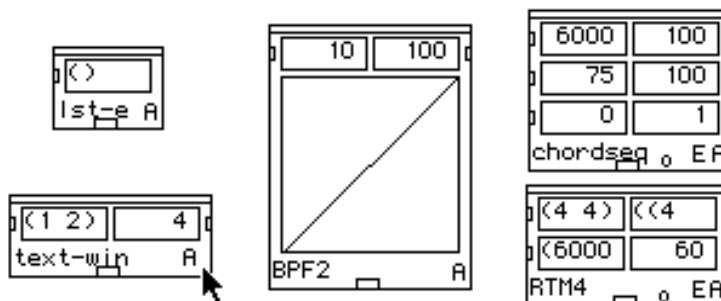
Inside a PatchWork patch, modules appears as boxes containing different parts :

- the box itself, which represents the module
- at the bottom left, the module's name
- at the bottom right, a *status marker*: *A*, *E*, *M*, or *\** .

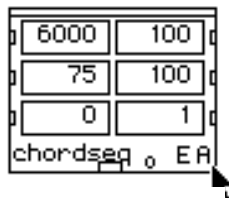
Some module boxes can invoke editor windows, open subpatches or abstractions, invoke pulldown menus, or add optional inputs. The following sections explain these behaviors.

## Invoking Editor Windows

The modules **lst-ed**, **text-win**, **chordseq** and **bpf** can invoke editor windows.

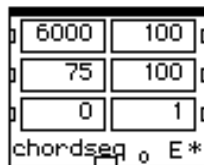


Once a module is opened, the letter *A* changes into an asterisk *\**, which remains as long as the module stays open with its window selected. For example, the **chordseq** module

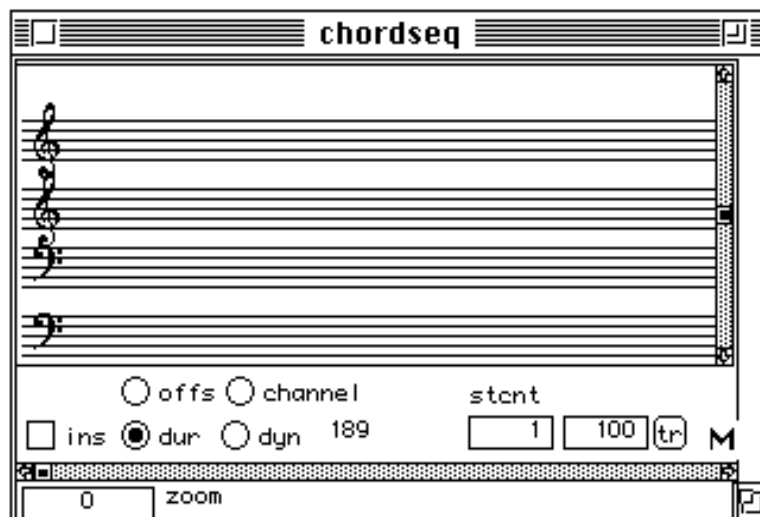


The module with its window closed

The A changes into an \*, when the window is opened.

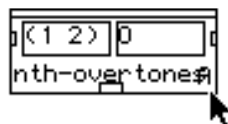


The module with  
its window opened  
and selected

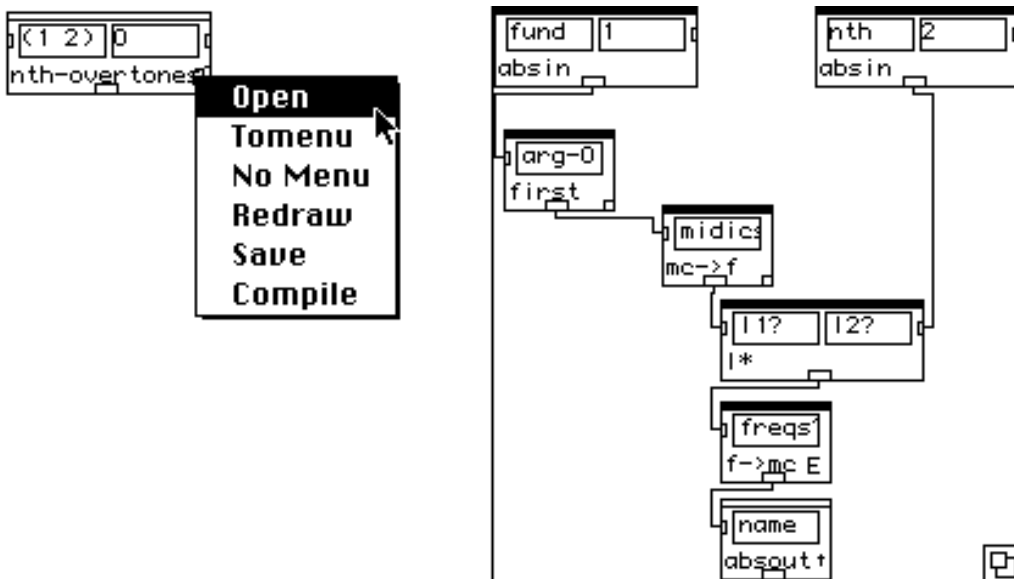


## Opening an Abstraction

An abstraction is a kind of subpatch or application, encapsulated into a module. For example, take the abstraction **nth-overtone** created by Tristan Murail:

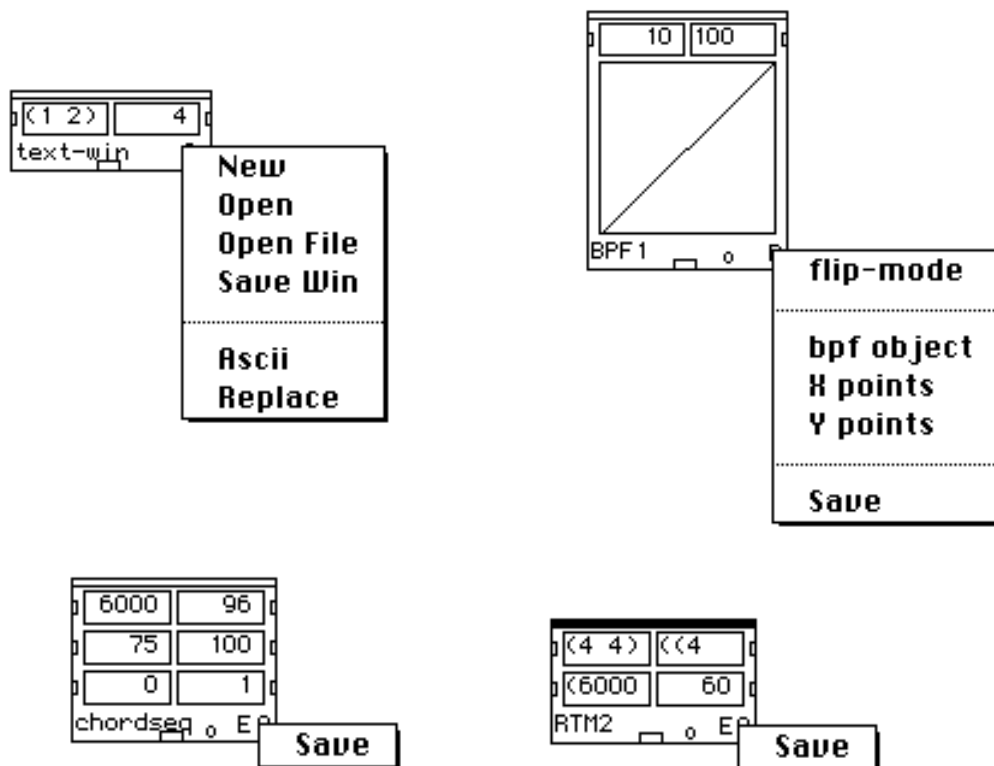


Clicking on the letter A brings up the following menu; selecting **open** opens the sub-patch contained in the abstraction.

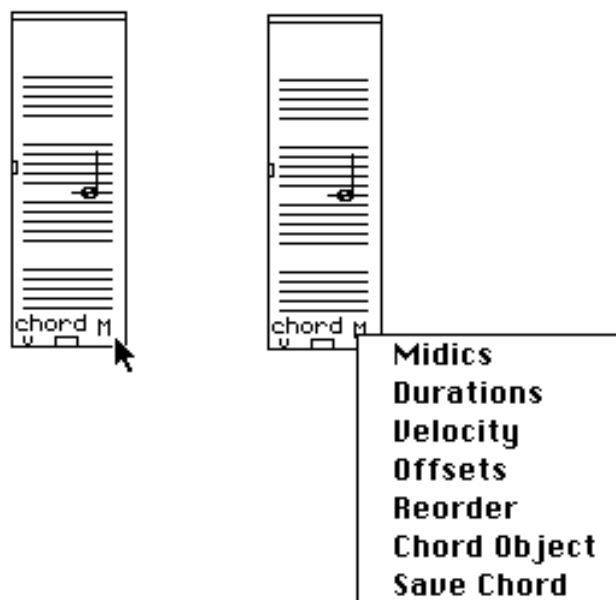


## Invoking a Menu from a Module

As we just saw with **nth-overtones**, the letter *A* often indicates that the module has a pulldown menu, accessed by clicking on the letter *A*.

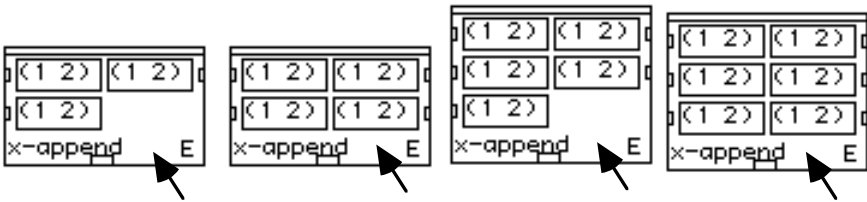


The **chord** module, although an application, appears with the letter *M* instead of *A* (See the description of the **chord** editor in the Reference manual.)



## Adding Optional Inputs: Extensible modules

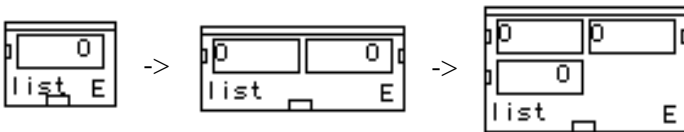
The letter *E* indicates that the module is *extensible*; in other words, it possesses optional inputs that can be added. To add an optional input, Option-click on the body of the module (or on the letter *E*); not on the input and output rectangles or on the name of the module. Take, for example, the module **x-append**. By Option-clicking, as many inputs as desired can be added:



In some cases, such as **chordseq**, modules have both an *E* and an *A*. This indicates that the module is an application and that it is extensible.



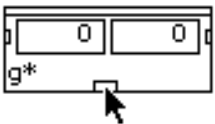
You have the ability to remove extensible module inputs. Take for example the module **list** (from **Kernel>lisp**).



It is possible to remove inputs with the following key combination: **ctrl+option+click** inside the box.

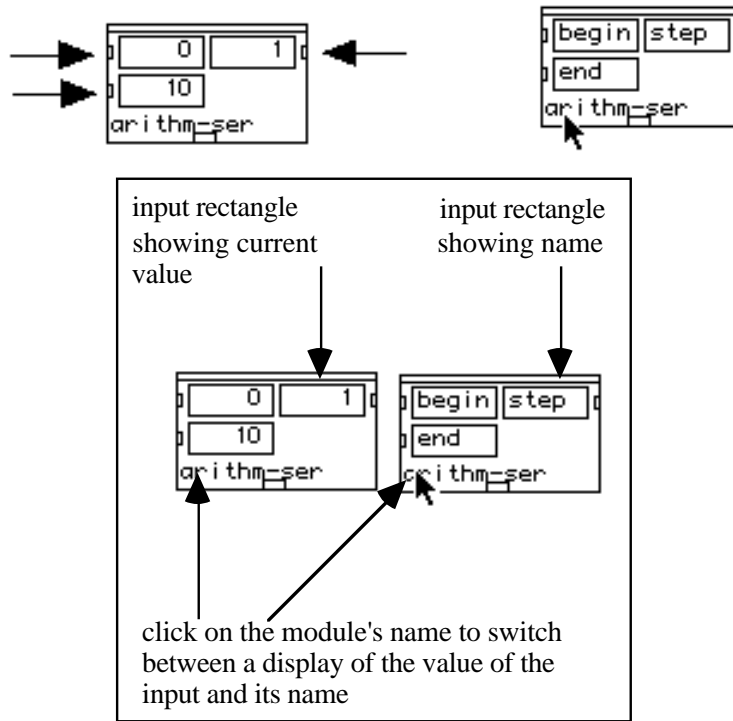
## Connecting Modules

All modules contain a small rectangle at the bottom-middle of the box. This is used to evaluate the module or to connect it to other modules.



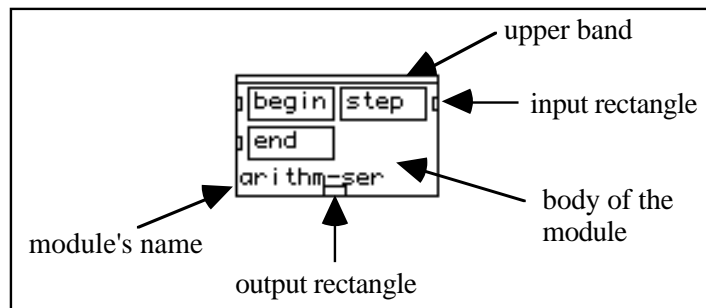
## Displaying Default Values

Inputs are indicated by small rectangles on the sides of modules. Often these rectangles display default values. Clicking on the name of a module switches back and forth between the display of parameter values and a display of the parameter names represented by the inputs.



## Anatomy of a Module Box

The top of the module has an upper band; this band has two main functions: selecting the module and moving it. We'll explain these operations in a moment. Now let's review the principal parts of a graphical module :



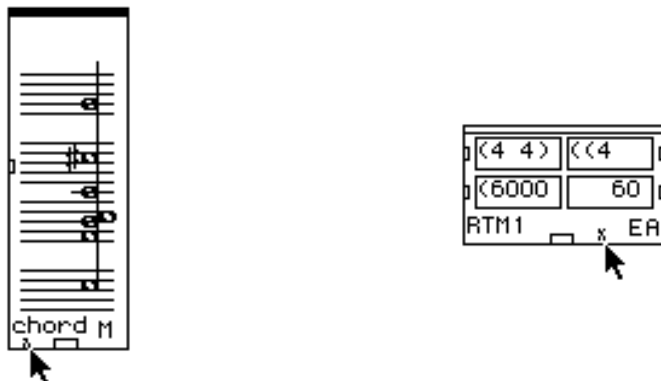


## Locking Modules

Certain modules (such as editors, buffers, etc.) have a *lock* that prevents the evaluation (and a possible loss of data) of the module. This lock is shown as a small letter *o* when the module is open:



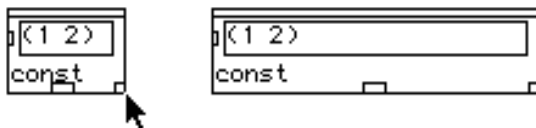
and by a letter *x* when the module is locked :



To toggle between states (locked and open) click on the lock.

## Stretching and Shrinking Modules

The length of some modules can be changed. These modules have a small square in the bottom right corner. To change the size, click on the small square and (keeping the button depressed) drag the edge of the box to the desired size.

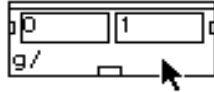


# Operations on Modules

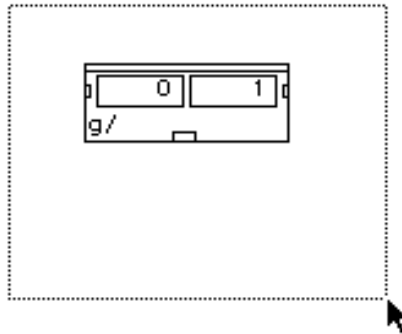
## Selecting a Module

A selected module is differentiated from the other modules in the same window, allowing special operations to be performed on it without effecting the others. There are two ways of selecting a module :

(a) Click on the body of the module (not on the rectangular lines)



(b) Or use standard Macintosh selection, clicking and dragging around it

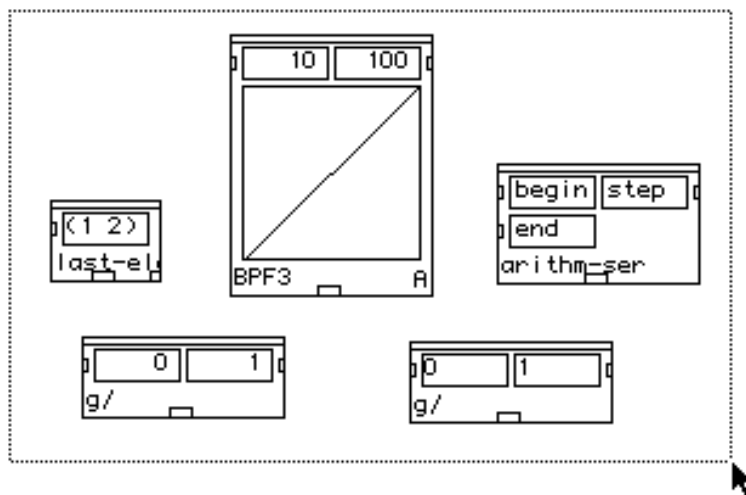


In both cases, once the module is selected, the upper band turns black.



**Note:**

Selecting a module deselects the previous selections, unless the Shift key is pressed. Clicking on the empty portions of a PW window deselects all selections. More than one module can be selected at a time, either by a Macintosh selection encompassing many modules:



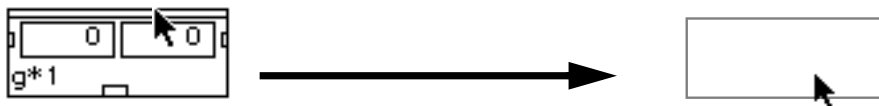
or by selecting the modules one by one while keeping the shift key depressed.

## Moving a Module

There are two techniques to move a module :

- click on the upper band. The upper band remains white; move the mouse and then release the mouse button.
- click anywhere on the "body" module, that is not inside its input windows: the upper band turns black.

If more than one module is selected, all the selected modules will move together. *ctrl click* anywhere within the body of the module will also allow the movement of a module.



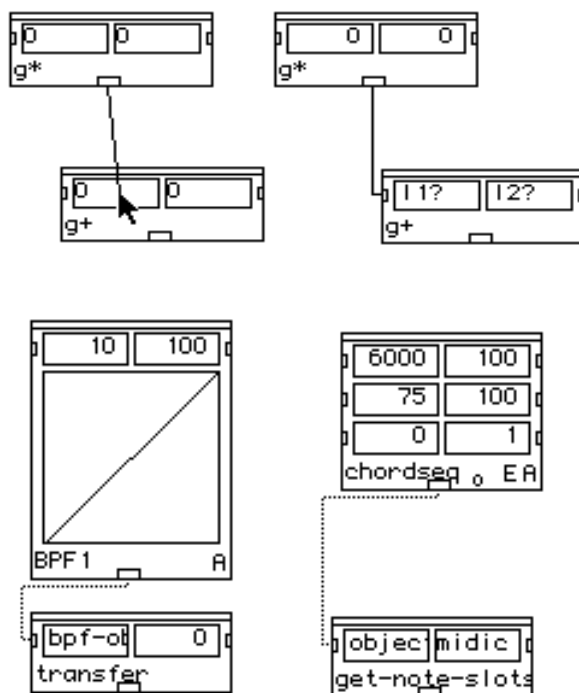
## Editing a Module

Cut, Copy and Paste apply to a selected module or modules. These commands are found under the edit menu and the equivalent keyboard commands are Command-X, Command-C, and Command-V (as is standard for Macintosh applications). Copying and pasting between different PW windows is also possible.

### Connecting Modules

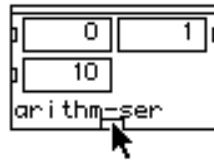
To connect two modules, click in the output rectangle of a module and then drag the mouse (keeping the button depressed) to the desired input rectangle. To cut a connection, Option-click in the input rectangle where the line is connected.

A single output can be connected to many different inputs, but if a second line is connected to an input rectangle, which is already connected, the second connection will replace the original. If the type of output is not compatible with the type of input needed; a message appears in the Listener window and the connection will not be allowed. A solid line indicates a connection which passes values, while a dotted line refers to a connection which is actually a *pointer* to an object. This distinction between value-passing and pointer-passing derives from the data types of the underlying Lisp functions. See the section on Lisp types and PatchWork interconnections at the end of this chapter.



## Evaluating a Module

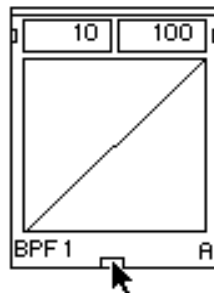
Evaluating a module means to calculate the function represented by a module using the parameters within the inputs of that module. To evaluate a module, place the cursor on the small output rectangle (located at the bottom-center of the module)



and Option-click (or select the module and type v). The results of the evaluation is displayed in the Listener window. In the example above, the **arithm-ser** module with the parameters shown above calculates an arithmetic series between 0 and 10 with a step of 1. This module is found under **Kernel / Num-series** menus. The result of the evaluation is :



Some modules, such as the editors, return a result which is neither numeric nor symbolic, instead, the evaluation of these functions creates an *object* (in the computer sense) Take the example of the graphic table **BPF**:



When this module is evaluated (place the cursor on the small output rectangle, located at the bottom-center of the module, and press Option-click) the result is not numeric; it is, rather, an address indicating the location of the object that the **BPF** module has constructed.



The **chord-seq** module behaves in exactly the same way:

6000	100
75	100
0	1
chordseq 0 EA	

after evaluation:



You may also evaluate a module by selecting it and by pressing the key "v".

## Editing Input Parameters

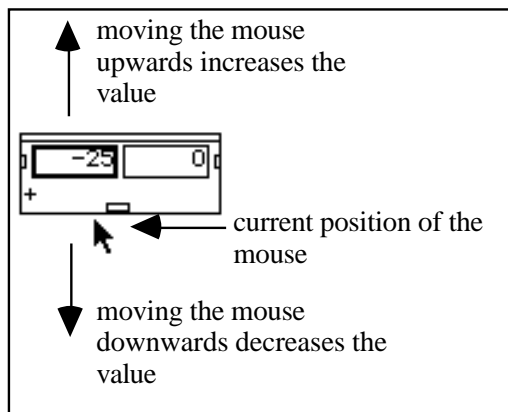
Editing the parameters to be fed to a module can be done in several ways. *Text input boxes* are used to enter either a Lisp expression or a symbolic name. These are edited by double-clicking on the input rectangle and typing the new name or Lisp expression. To exit the edit mode and save the change, type Return. If you click elsewhere in the window before Return is pressed, changes to the text box are erased.



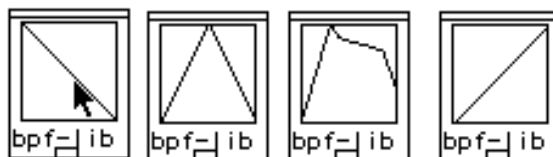
Editing numeric values can be done in two different ways:

- Double-click on the input rectangle; this opens the input box for editing, allowing the new value to be entered directly from the keyboard.
- Click on the input rectangle and then, while keeping the mouse-button pressed, move the mouse up or down (see the diagram below). This changes the values in the corresponding direction (up or down). The increment of change can be determined by pressing a key while dragging the mouse:

- no key : increment = 1
- shift key : increment = 10
- option key : increment = 100
- command key : increment = 1000



Some modules have inputs that are circular lists; these are actually a sort of menu. For the **bpf-lib** module, clicking on the input rectangle and dragging the mouse up or down causes the module to cycle through the available curves in a circular fashion (a loop).



Note: the special commands relative to PW's editor modules: **BPF**, **chord**, **chordseq**, **multiseq** and **rtm** are given later, in sections dedicated to each one of those modules (or editors).

# Lisp Data Types and PatchWork Interconnections

Since PatchWork modules are based on Lisp functions, it is important to know the data types that these modules handle. You encounter these types when you look at the output of patches and when you try to interconnect two PatchWork modules that handle incompatible types: PatchWork will not allow such an interconnection.

Some modules handle *generic* inputs. That is, they can accept a variety of types. The main PatchWork data types are the following:

- *Integers* — whole numbers such as 10, -789. An example of a module that handles integers is **g+**.
- *Rationals* — ratios between two integers, as in 21/5 and 56/3. To compute the division in decimal form, the arguments to the function must be in floating-point format. For example, (/ 21 5) fi 21/5, while (/ 21.0 5.0) fi 4.2. An example of a function that handles rationals is **approx**.
- *Floating-point numbers* — real numbers with decimal points, like 3.7, 1., and 5.987654. An example of a module that handles floating-point numbers is **g/**.
- *Lists* — a collection of objects enclosed in parentheses, like (1 2 "Bach" 99.99). Objects in a list can themselves be lists as in (87 (65 43) 99). Many PatchWork objects take the form of lists: a chord combines a list of pitches, a list of durations, etc. An example of a module that handles lists is **chord**.
- *Objects* — Modules that pass objects are connected by dotted lines, rather than solid (value-passing) lines. Objects in PatchWork let users change internal parameters or access otherwise inaccessible data. Some modules, such as **bpf**, always return the same object. Other modules, such as **chord** (in object mode) return a newly created object each time they are evaluated. For example, a chord object is printed as follows in the Lisp Listener:

```
#<pw::c-chord 0x46DBA1>.
```

Integers, rationals, floating-point numbers, lists, and objects are the main types that users must reckon with. Advanced users who would like to extend PatchWork's functionality by programming in Lisp need to know about other types, such as strings, hash tables, and so on, that we will not go into in this introduction. See the *PatchWork Programming Guide* for details.



# On-line Documentation

Fundamental to the use of PatchWork are the commands that access documentation directly from within PW.

## Obtaining On-line Documentation for a Module

To see a brief description of a module, often with instructions for its usage, select the module and type the letter *d* or choose the item **Documentation** in the **PWoper** menu. This opens a Fred window with the on-line documentation for the selected module.

## Obtaining the Lisp Code Definition of a Module

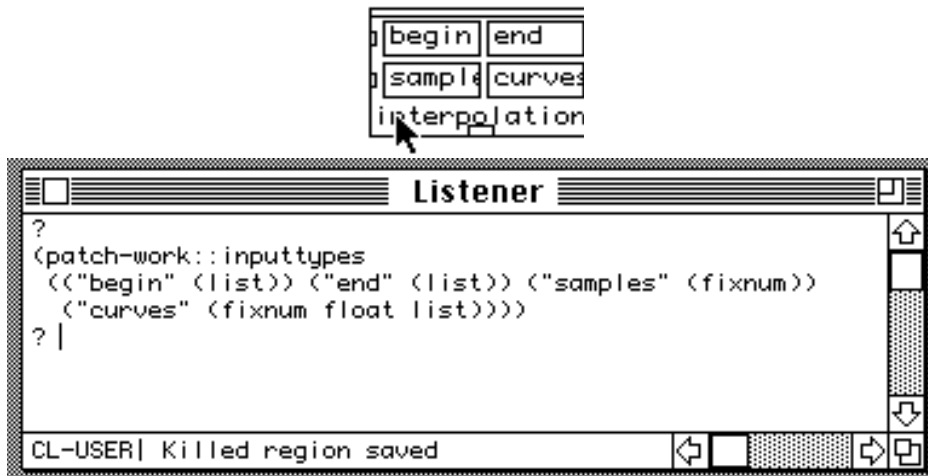
To obtain the Lisp code defining a module, select the module, and type the letter *e* or choose the item **Definition** under the **PWoper** menu. This opens a file containing the Lisp code describing the selected module — if it is present and if the path is unchanged since the file was compiled.

## Obtaining the Tutorial file of a Module

To obtain the Tutorial filea module, select the module, and type the letter *t* .

## Obtaining Input and Output Type Information on a Module

To obtain information on the types of input a module can receive and the type of data it outputs, Command-click on the name of the module; this causes the module's input types to be displayed in the Listener window:



In the above example, for the module **interpolation** the input types are :

- a list, for the parameter *begin*
- a list, for the parameter *end*
- a whole number, for the parameter *samples*
- a whole number, a floating point number or a list for the parameter *curves*

Or Option-click on the name of the module; this causes the module's output types to be displayed in the Listener window :



In this case the output is in the form of a list.

## ***Module Help Files***

To obtain the help patch for a module, select the module, type the letter *t*. This opens a tutorial patch that uses the selected module.

**Important:** Since these patches are not protected, it is essential that they not be changed. For study purposes, the patches can be copied and the copies altered.

# 6 Additional Keyboard Commands

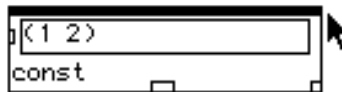
PatchWork and its modules respond to other commands that can be activated from the keyboard:

*h* Opens the window help display. From within any PW main window, *h* invokes window help containing a summary of all keyboard commands

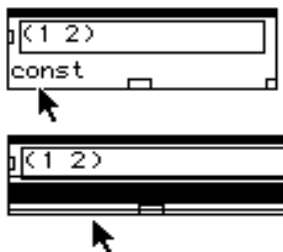
Note the difference between the uppercase and lowercase form of this command. *h* opens the window help, while *H* with an editor open causes behavior specific to that editor.

Return Hides the PW help window and returns the user to the main PW window.

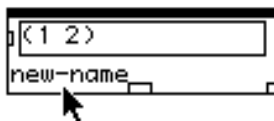
*r* Rename the selected module.



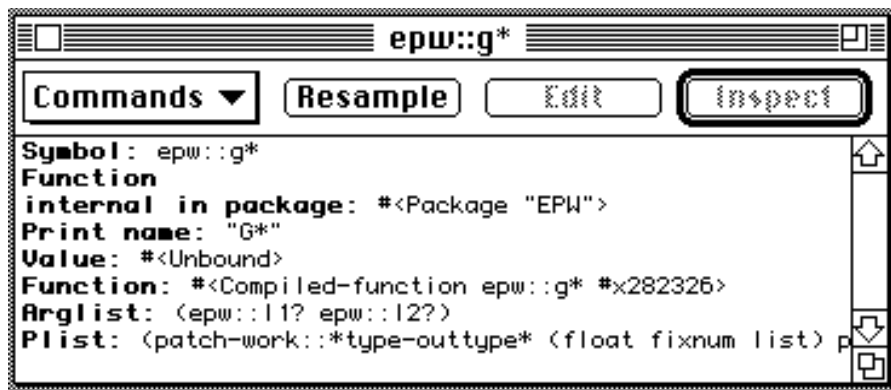
Type the letter *r*; this highlights the strip where the old name was.



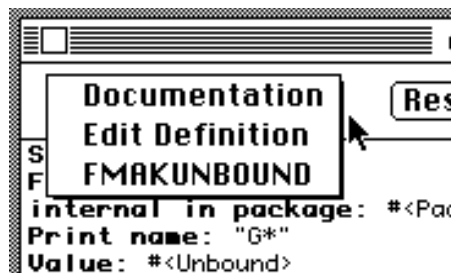
Finally, type the new name, followed by Return. Now the module is renamed.



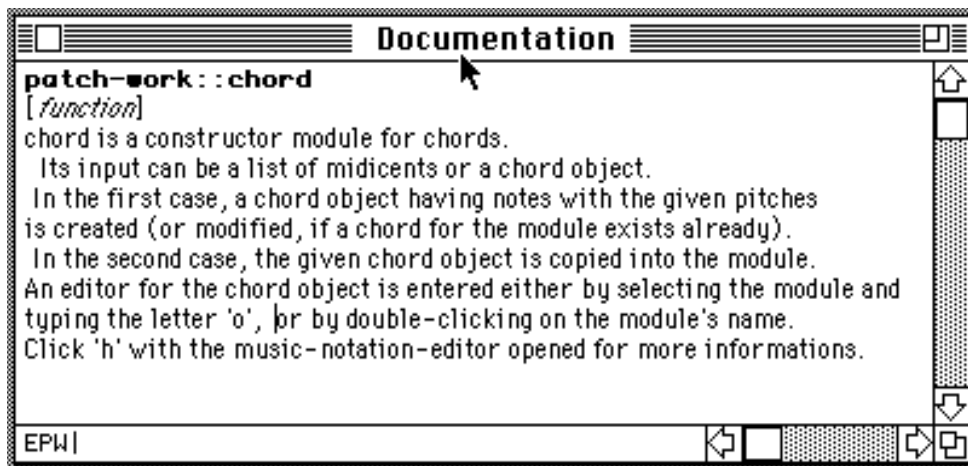
*i* Examine the selected module. This accesses information about the selected module. Select the module and type *i*. This opens Fred inspection window



This window contains a menu called Commands, which accesses the module's on-line documentation, the Lisp definition (if the non-compiled files are present) or to the command FMAKUNBOUND. Note: the FMAKUNBOUND option is extremely dangerous, because it unbinds the function.



*d* calls up the on-line documentation for the selected module. Here is an example of the on-line documentation for the **chord** module:

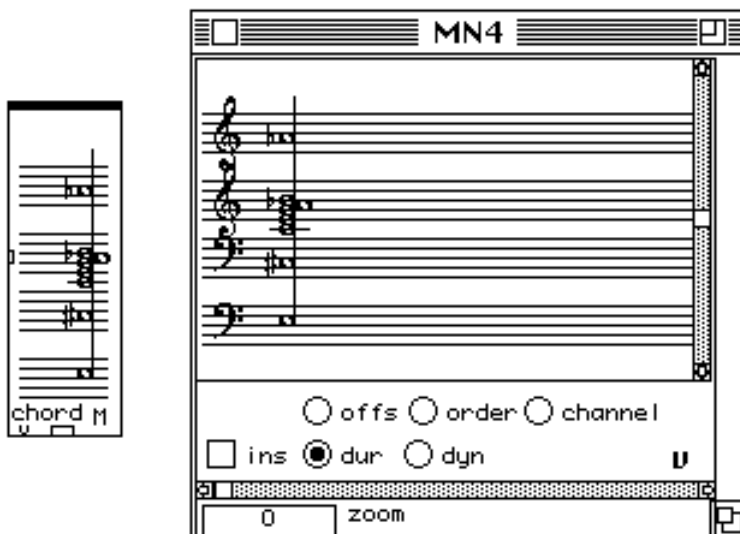


Edit the definition of the selected module This command gives direct access to the Lisp definition of the selected module (if the non-compiled Lisp files are present). Take, for example, the Lisp definition of the module **arithm-ser**:

```
(defunp arithm-ser ((begin fix/float) (step (fix/float (:value 1)))
                  (end (fix/float (:value 10))))
  list
  "Returns a list of numbers starting from begin to end with step step.
  See tutorial "arithmetic.pw"
  (let ((l ()))
    (for (i begin step end) (newl l i))
    (nreverse l)))
```

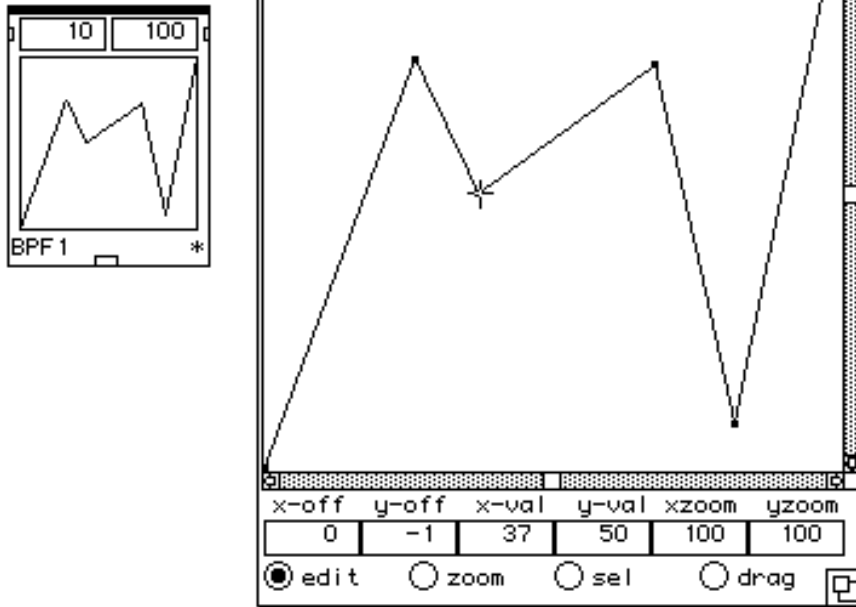
Brings up a Module Help file, a tutorial patch that demonstrates the use of the selected module.

Opens the edit window of the selected application. This command opens the windows contained within application modules such as **bpf**, **chord**, **chordseq**, **rtm**, **poly-rtm**, **multiseq** and abstractions in general. Take, for example the module **chord**:



As a second example of an editor window associated with the module, the following editor appears when you type *o* with the module **bpf** selected.

Open the tutorial file for a module.

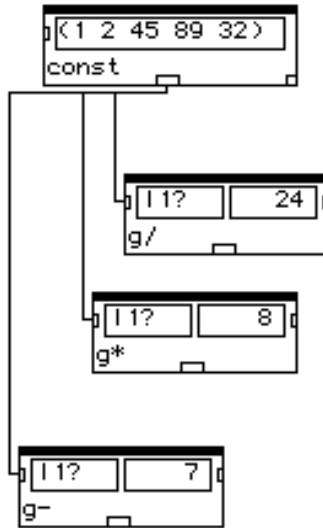
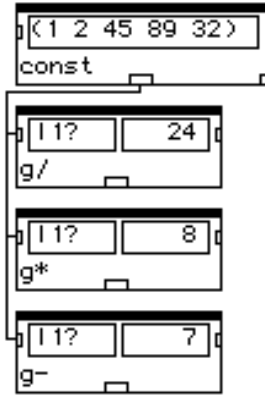


To close these application windows, type Return.

- A* Abstracts the selected modules. Transforms the selected modules (or patch) into an abstraction.
- D* Redraw window. Causes the current PW window to be redrawn.

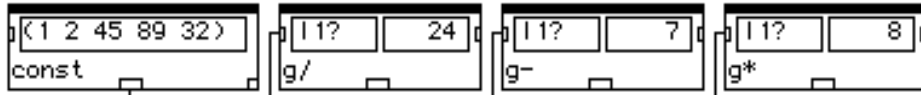
X

Vertically aligns a group of selected modules. First select the modules.

Then type the letter *X*.

Y

Horizontally aligns the selected modules.



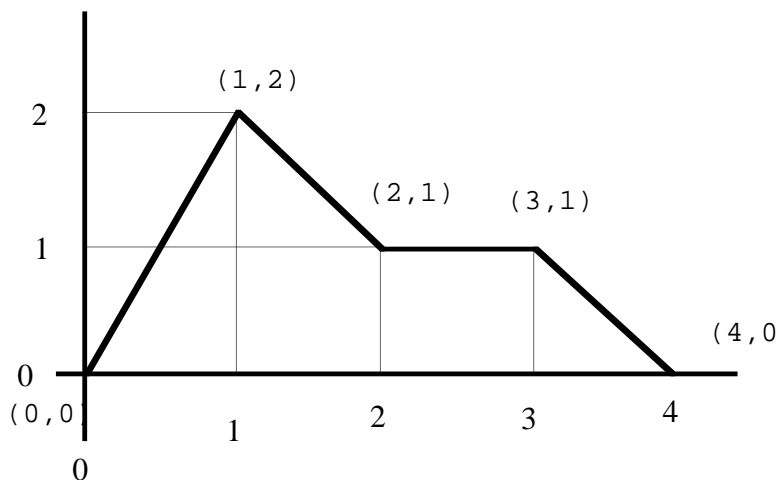
Other keyboard commands in PatchWork are specific to certain modules (the editors, in particular). These commands are documented in the Window Help display, accessed by typing *H*. For the commands associated with a given editor, the editor must be open to see the Window Help file associated with it.

# 7 PatchWork Editors and Musical Objects

This chapter looks at the main editing modules and the musical objects supported by PatchWork. The editors include the **multi-bpf** (break-point function) editor, the **chord** editor, the **multiseq** editor, the **chordseq** editor, the **RTM** editor and the **poly-rtm** editor.

## The multi-bpf Editor Module

A breakpoint function (*BPF*) is a function made up entirely of continuous straight-line segments. The structure of a BPF can be defined completely by a series of points representing the endpoints of each line segment. Therefore a BPF contains its information much more economically (in terms of memory) than a continuous (or rather, a discrete version of a continuous) function. Although they are an inherently economic way of displaying functions, BPFs are especially well suited to material that makes use of straight lines (as opposed to approximating curves with numerous line segments). Here is an example of a BPF defined by the following points:  $((0,0) (1,2) (2,1) (3,1) (4,0))$  .



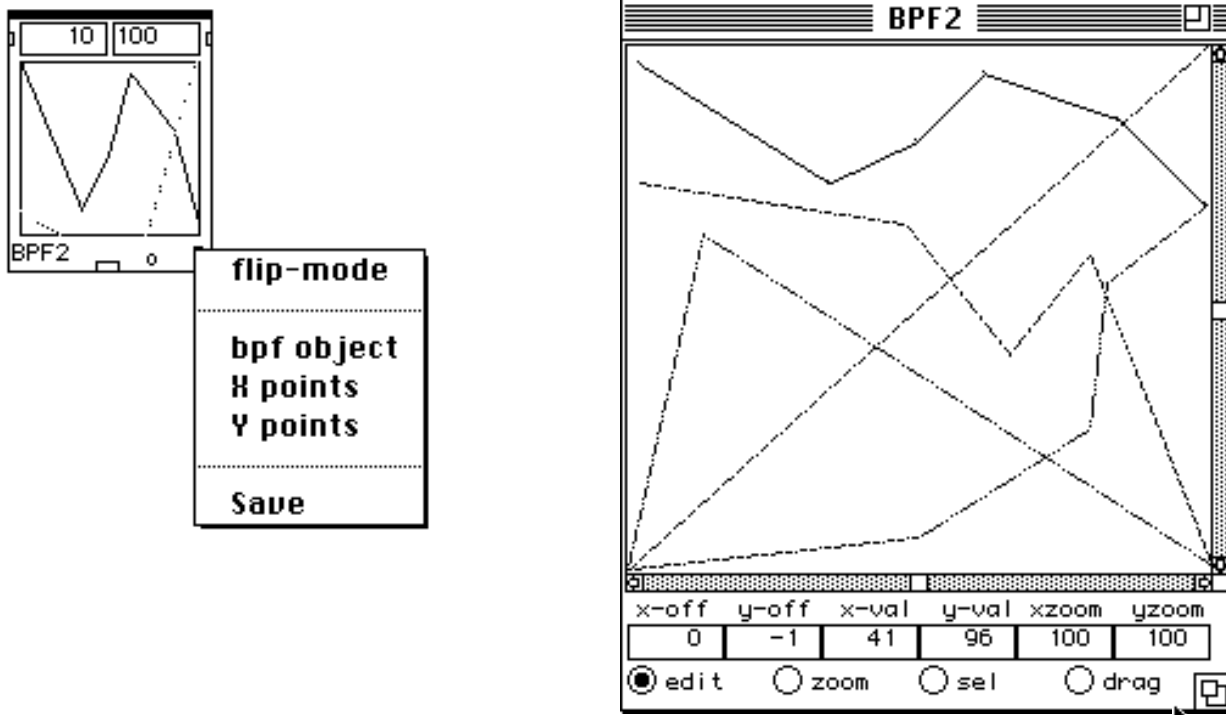
BPFs are most often used as temporal functions (for example: envelopes, periodic waves, etc.) or as transfer functions (functions where the output is equal to the function applied to the input).

The **multi-bpf** module can be used to:

- create and edit simultaneous breakpoint functions at once
- create and edit coordinate pairs  $(x,y)$
- display a series of coordinate pairs, either as a BPF or as series of  $x, y$  points
- save and load BPFs to and from a library
- retrieve data concerning the points contained in the BPF

The following is an image of a **multi-bpf** editor module and its window :





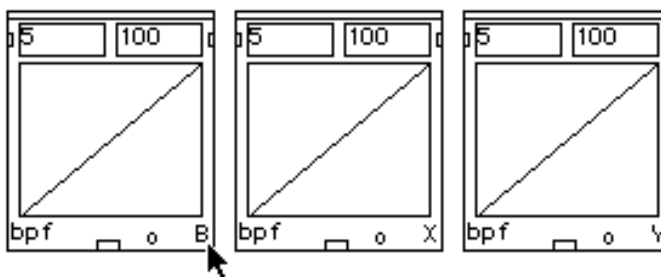
The **multi-bpf** has two input rectangles, labeled *tlist* for data along the *x*- axis, and *vlist* for data along the *y*- axis. It also has a pulldown menu whose items are:

*flip-mode* changes the mode of display and editing between line segments or simply points.

**multi-bpf** object, *x* points, *y* points — determine what the module outputs: the options are a BPF object (or a list of **multi-bpf** objects) a list of coordinates along the *x*- axis, or a list of points along the *y*- axis (respectively). Remember that in the case of several BPF, the output is in a form of a list of lists, where each sub-list is associated with one of the curves contained in the editor.

*Save* — saves the module along with its current contents, whether calculated or edited.

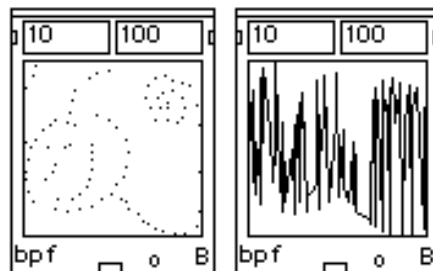
The type of output selected is indicated by a letter displayed in the bottom right corner of the module:



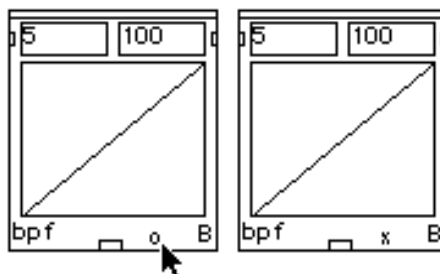
Try evaluating the BPF module with each of the three output options.

The BPF's visualization window displayed either a break point functions or points. Select flip-mode in the pul-

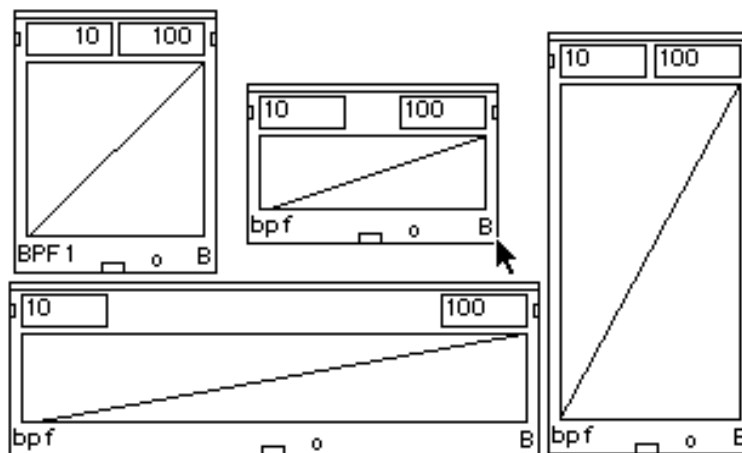
down menu to switch, back and forth, between the two modes; the default setting is BPF.



The BPF module has a lock that protects the values it contains (whether calculated or edited), and to avoid recalculating the module. The lock appears as a small letter to the right of the output rectangle; an *o* when open or an *x* when locked. Clicking on the letter changes the status (lock or unlock the module).

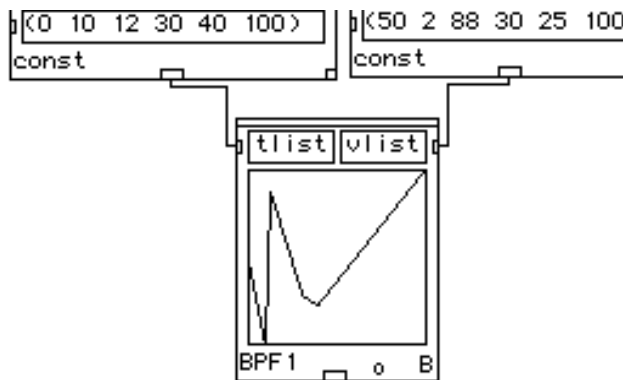


Clicking on the bottom-right corner, then dragging the module's outline resizes the module.



# Displaying Data with the multi-bpf Editor

The simplest use of the **multi-bpf** module is for display of data. In this case, the module receives data through it's two input rectangles: *tlist* and *vlist*. *tlist* will receive the data corresponding to the x- axis and *vlist* the data for the y-axis:



The curve thus formed corresponds to a break-point function passing through the following coordinate pairs :

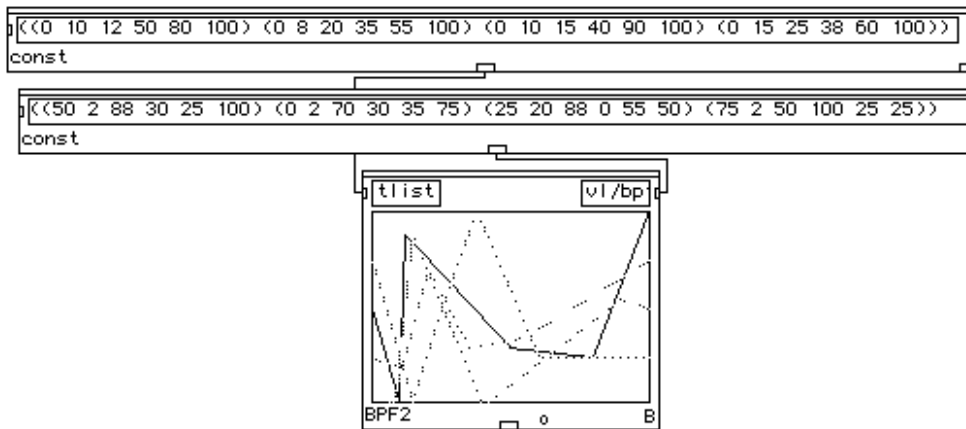
((0 50) (10 2) (12 88) (30 30) (40 25) (100 100))

To display several BPF, the data received by 'tlist' and 'vlist' are lists of lists, where each sub-list corresponds to the parameters of a BPF. Of course, the data of both lists must match..

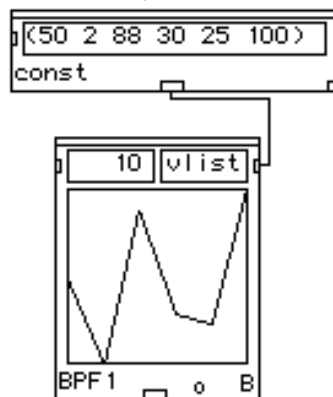
For example, let's consider these four BPF defined by the points:

BPF1: ((0 50) (10 2) (12 88) (50 30) (80 25) (100 100))  
BPF2: ((0 0) (8 2) (20 70) (35 30) (55 35) (100 75))  
BPF3: ((0 25) (10 20) (15 88) (40 0) (90 55) (100 50))  
BPF4: ((0 75) (15 2) (25 50) (38 100) (60 25) (100 25))

We must group the x coordinates and the y coordinates in lists of lists.

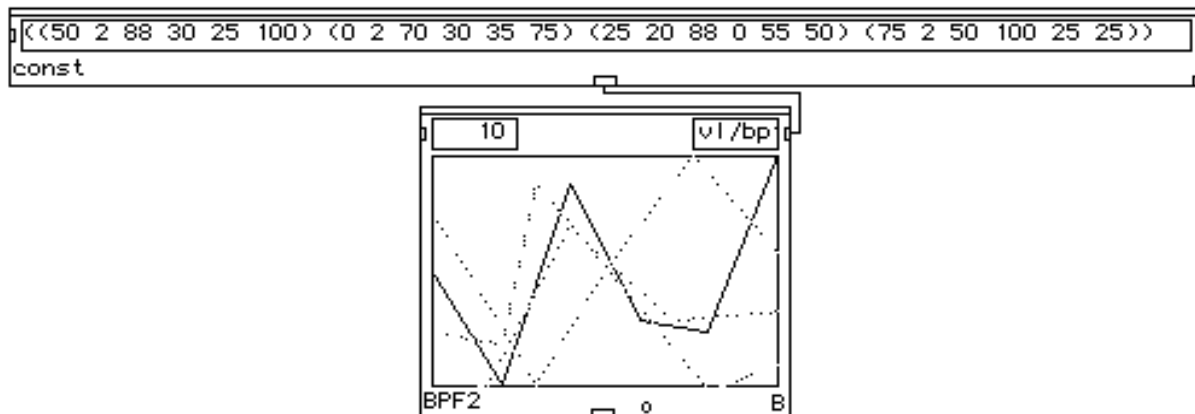


It is possible to enter data only for the *vlist*; in this case *tlist* will become the step for the x- axis values to be matched with the incoming y- axis values. In the example below, the value for *tlist* is 10.



The x coordinates to be matched with our y coordinates are (0 10 20 30 40 50), and the coordinate pairs :  
((0 50) (10 2) (20 88) (30 30) (40 25) (50 100))

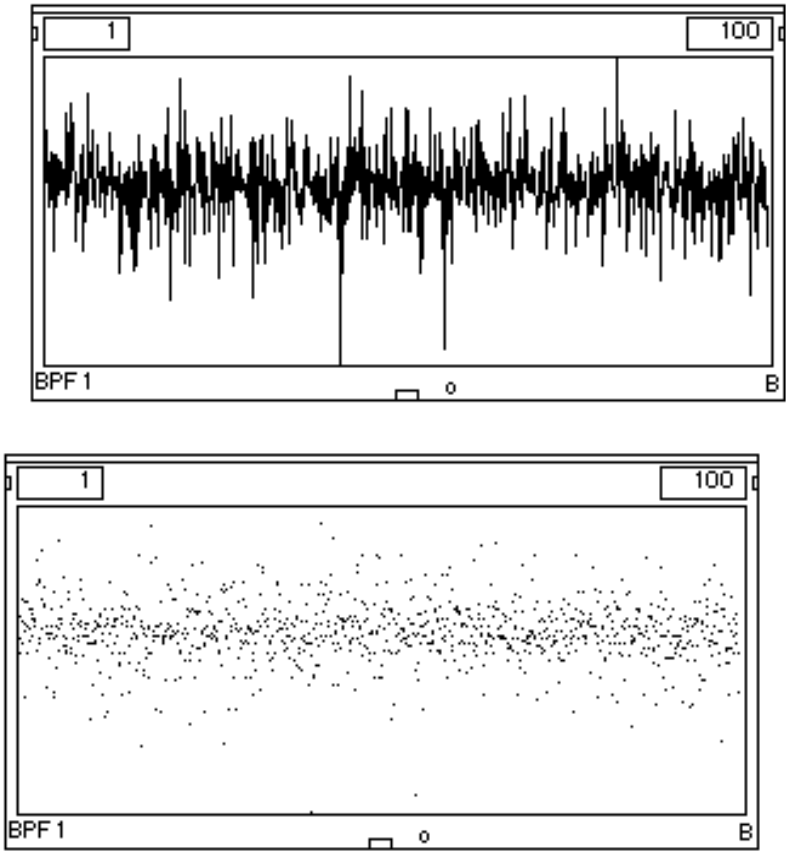
or for multiple BPF,



The x coordinates to be matched with our y coordinates are (0 10 20 30 40 50), and the coordinate pairs :  
BPF1: ((0 50) (10 2) (20 88) (30 30) (40 25) (50 100))

```
BPF2: ((0 0) (10 2) (20 70) (30 30) (40 35) (50 75))
BPF3: ((0 25) (10 20) (20 88) (30 0) (40 55) (50 50))
BPF4: ((0 75) (10 2) (20 50) (30 100) (40 25) (50 25))
```

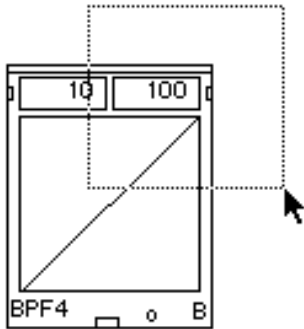
When working with a large number of points, it can be more useful to visualize the results as a cloud of points, rather than as a BPF.



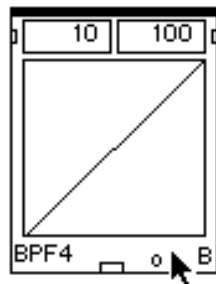
These examples contrast two displays of 1000 points generated with a bilateral exponential distribution.

***Using the BPF Editor Window***

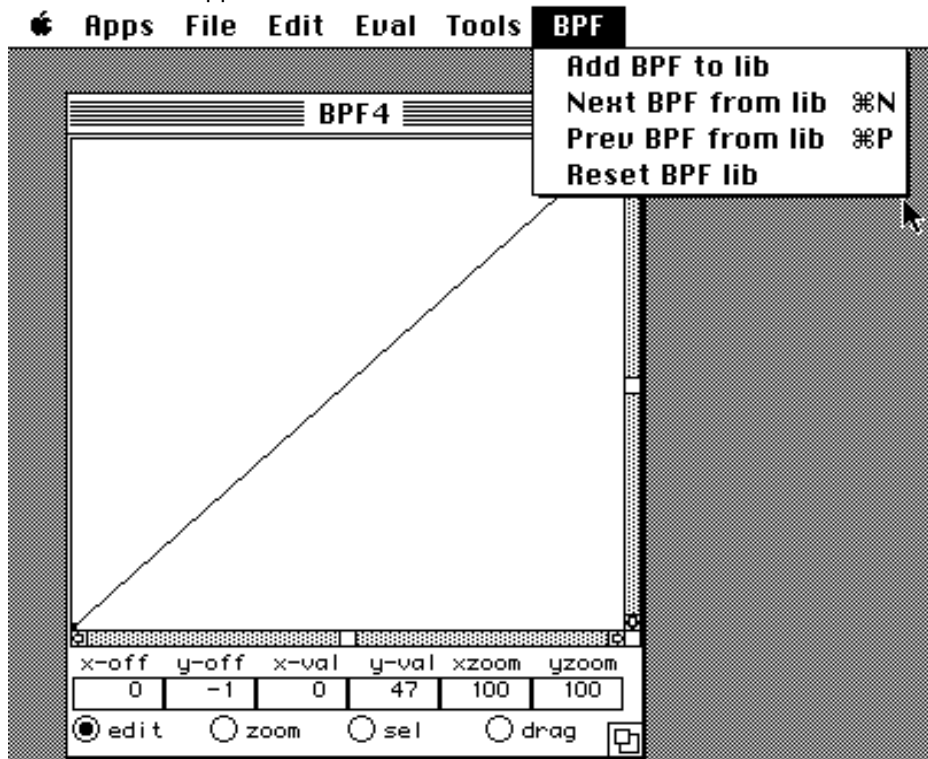
To open a Multi-BPF editor window, select the Multi-BPF module, then type the letter *o*.



Or double-click on the body of the Multi-BPF module:



In either case, the editor window appears.

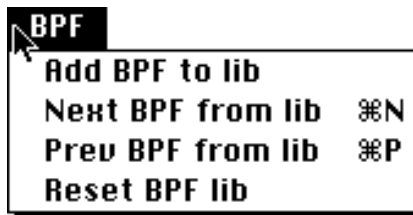


## ***The multi-bpf Menu Bar***

A Multi-BPF editor is always linked with a Multi-BPF module, when the module is opened the editor is "inside." This editor allows a Multi-BPF to be created and edited.

When a Multi-BPF editor window is selected, or when the option BPF is selected from under the menu Apps, the PW menu-bar changes into the BPF menu bar. The BPF menu manages the library of BPFs. It is described below.

Breakpoint functions can be saved in a library. This is a single library which is structured as a circular list (a loop). The item Add BPF to lib adds the current function to the library; Reset BPF lib resets the library, this removes all the functions that had been added. The Next BPF from lib and Prev BPF from lib items cycle through the functions stored in the library.



## The multi-bpf Display

The *x-val* and *y-val* rectangles display the values representing the current position of the cursor (along their respective axes).

*x-off* and *y-off* show how much the function has been displaced (along each axis) from the original BPF. The quantity of offset can be changed by clicking on one of these two rectangles and moving the mouse up or down.

The zooms: *x-zoom* and *y-zoom* allow the display to be scaled.

## The multi-bpf Buttons

EDIT	Select this button to edit the points.
ZOOM	Allows you to enlarge a specific region of the BPF. To zoom in on a specific region within the BPF, click on the button marked <i>zoom</i> and select the region to be enlarged. To return to a standard display of the entire function, type the letter <i>f</i> .
SEL	With SEL, you can select a set of points for editing purposes (cut, copy, paste, etc...)
DRAG	Select DRAG to move the BPF (or BPFs) simply by pointing the cursor et clicking in the window; while holding the mouse buton down, move the cursor to change the location of the BPF.

## ***Editing a Breakpoint Function***

To edit the function, the *edit* button must be selected from within the editor window.

- Adding a new point: place the pointer at the proper coordinates (x,y) and click the mouse button. The line segments are automatically recalculated so as to include the new point in the BPF.
- Selecting a point: place the cursor on the desired breakpoint; the cursor changes into a cross when the point is selected.
- Removing a point: select the point, as described above, and type the back space or delete key on the keyboard.
- Moving a point: select the point, click on the mouse button and move the cursor to the desired position, then release the button.
- To select multiple points at the same time: click on the *sel* button, then click and drag the mouse to encompass the desired section.
- To view a part of the BPF not currently visible in the edit window: click on the *drag* button, then click in the window to move the display towards a new position..
- To add a new BPF to the current window type the letter *a* (with the module opened). A new BPF will be added at the same location than the BFP previously selected.
- To Select a BPF, in order to switch between various BPF, for editing purposes: type the letter *s*. The selected BPF will be displayed with bold segments, while the other BPF will remain in the background with dotted lines segments.



## ***multi-bpf Editor Keyboard Commands***

<i>H</i>	opens the Window Help file, displaying commands
Return	selects the current PW window, hiding the BPF editor Enter selects the current PW window, hiding the BPF editor
<i>R</i>	renames the BPF window
Backspace	deletes the selected point
<i>f</i>	rescales the BPF so that the function fills the editor window
<i>K</i>	removes all point except the first
+	zoom out
-	zoom in
g	show/hide the grid
->	time-stretch the selected points
<-	time-contract the selected points
up-arrow	stretch the values of the selected points
down-arrow	contract the values of the selected points
tab	change to another edit mode following the sequence (edit - zoom - sel - drag ...)
a	add another BPF to the editor
s	select one BPF
d	deletes the selected BPF

# PatchWork's Musical Structure

At this point we switch gears to briefly describe the underlying musical representation scheme of PatchWork. This is important because the modules and editors we discuss after this introduction deal with several types of musical structures: notes, chords, sequences of chords, and sequences of measures.

Each is represented by a data structure, referred to in computer science terms as an *object*. An object is defined by a certain number of attributes, called *fields*. Each field has a name, a type, a default value and limiting values. Here is a list of the field names for each type of PatchWork object.

## Note Object Fields

Field-names	Description
midic	Pitch in midicents; default value : 6000
vel	Velocity of the note in MIDI, between 0 and 127. Default value :100
dur	Note duration in hundredths of a second. Default value :100
chan	MIDI channel number. Default value : 1
order	Order number of the note, relative to other notes of a chord object, when the note is part of a chord. Default value : 0
offset-time	Delay between the attack time of the chord of which the note is a part, and the note itself. Default value : 0
comm	A string of commentary related to the note. Default value : " "
ins	The instrument object that plays the note. Default value : MIDI

## Chord Object Fields

Field-names	Description
notes	A list of note objects. The default value is a single note object with the default values.
t-time	The attack time of the chord. Default value : 0

## Chord Sequence Object Fields

Field-names	Description
accord	A list. Default value : (6000)

## Measure Sequence Object Fields

Field-names	Description
measures	A list of measure objects (these contain the beats).
stop-flag	Indicates the stopping point

Each structure (note, chords and chord sequence) inherits all the fields of the lower level objects it contains. For example a chord object contains a note object (with all its fields) for each note of the chord.

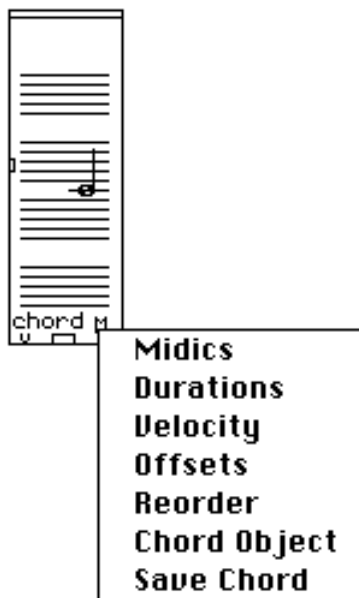
The abovementioned structures are represented in PatchWork by the modules chord, **chordseq** and **rtm**. The **chord** and **chordseq** modules represent chords and sequences of chords, respectively. The **rtm** module displays musical rhythmic notation with notes, chords, or sequences

# The Chord Editor Module - measure-line

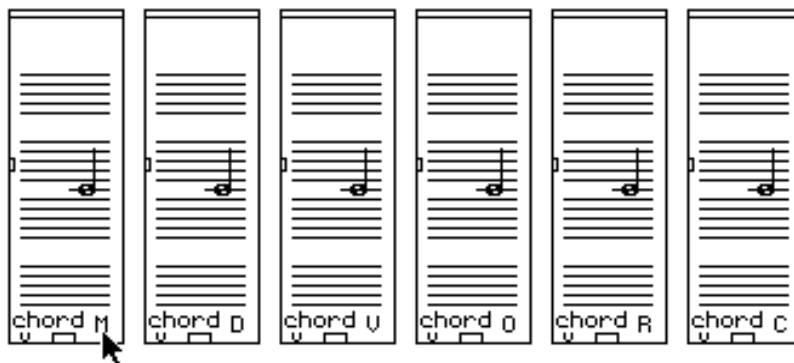
The **chord** module editor displays results in musical notation. It can also be used to create a chord object, or edit the pitches of a chord. Its input is either a list of pitches in midicents or a chord object. In the first case, it creates a chord object (or modifies one, if the module had previously contained a chord). This object is made up of notes with the *pitches* mentioned previously. In the second case, the chord object is simply copied into the modules with all its parameters intact. Since the **chord** module outputs a list, it can also be used to display or edit a sequence.

Note: **chord** accepts only simple lists as inputs (lists with only one level of parenthesis).

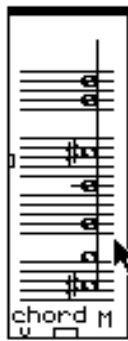
The module has one input rectangle and a pulldown menu that chooses the type of output **chord** produces:



The type of output selected is indicated by a letter at the bottom-right of the module.



Try evaluating **chord** with each of the different output options. The chord visualization window displays the notes of the chord:

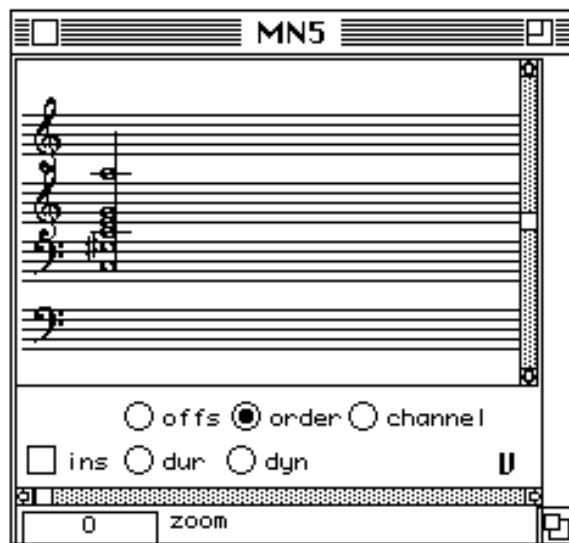


A lock can protect the contents of a chord module (whether edited or calculated), or to avoid reevaluation of the module. The lock appears as a small letter to the left of the output rectangle; an *o* when open or an *x* when locked. Clicking on the letter changes the status (lock or unlock the module).



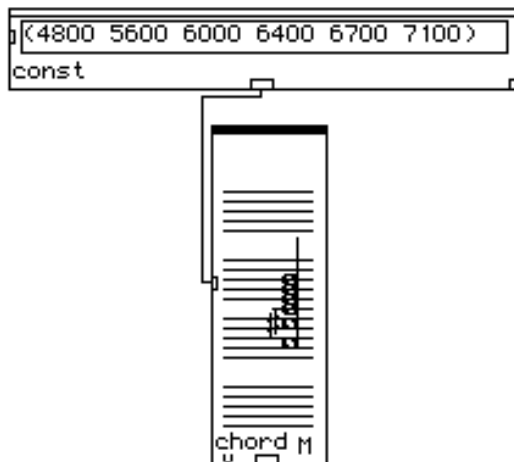
## The Chord Editor Window

The editor window associated with this module looks like this:



One of the main uses of the **chord** module is to display results. To do this, connect a list of midicents to the input of the module **chord**; the module interprets this as the notes of a chord (although it is also possible to interpret this as a melodic sequence).

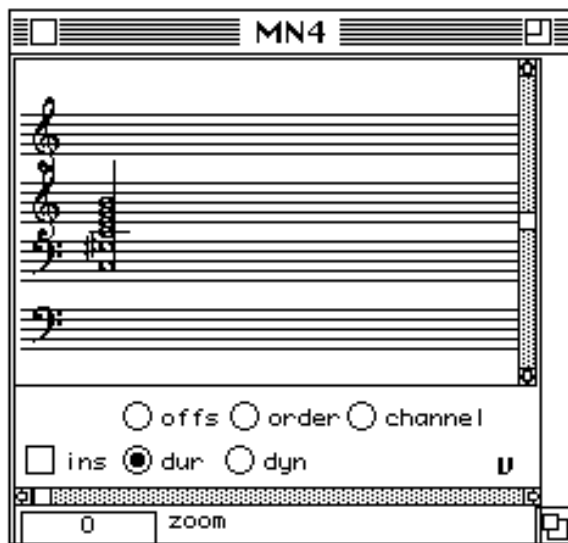
As an example, we connect a list of midicents (edited in the module **const**) to the input of the **chord** module :



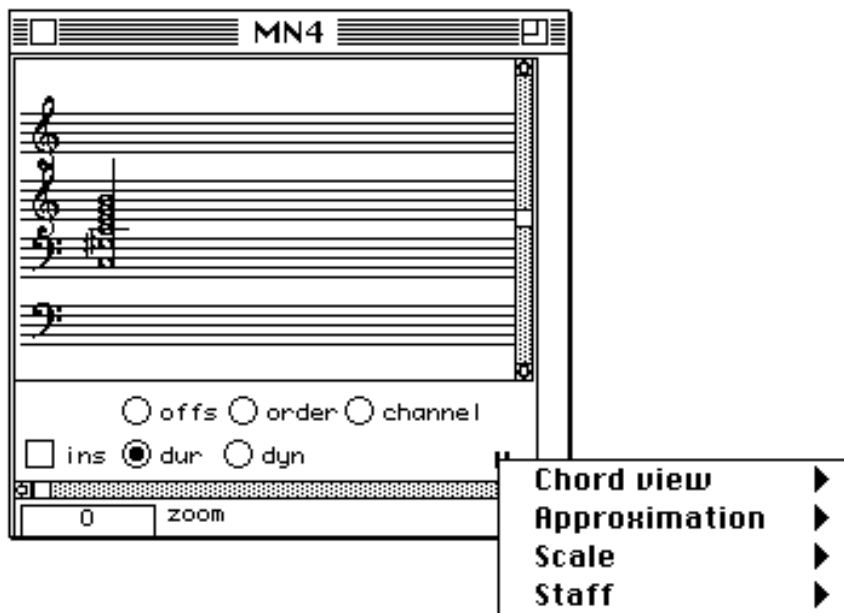
Now, if we evaluate the **chord** module, the chord will appear for visualization within it and the following output appears in the Listener window :

```
? PW-(4800 5600 6000 6400 6700 7100)
```

It can be interesting to select a different output option from the pull-down menu and then re-evaluate the module. To edit in the **chord** module, open the editor window by selecting the **chord** module, and typing *o*, or double-click on the body of the **chord** module.



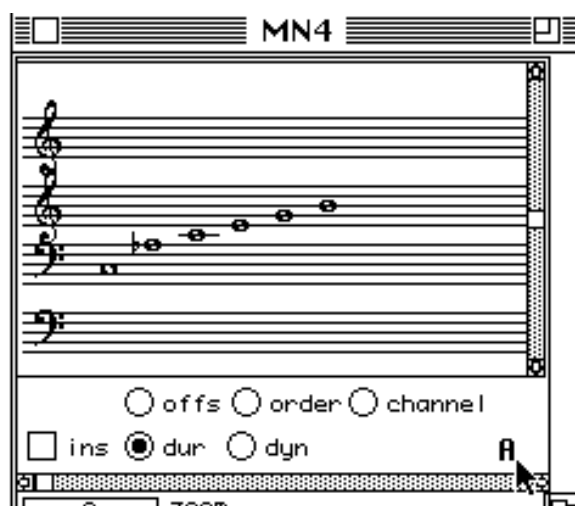
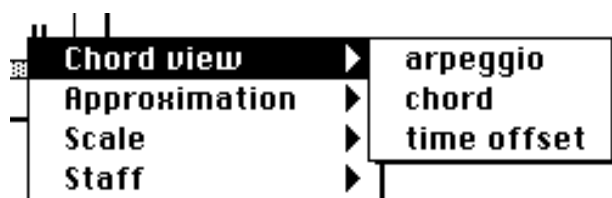
The editor window contains the following two scroll bars, one for vertical and the other for horizontal, a pulldown menu under the letter V at the bottom-right of the window, five buttons for the display of different parameters of the chord, and a resize box.



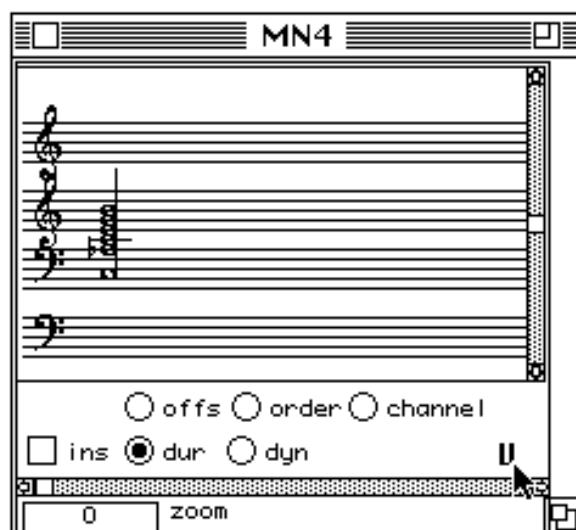
## Chord View Submenu

The **Chord view** specifies how the chord is represented, as an **arpeggio**, **chord**, or with a **time offset**. Each representation is explained next.

**arpeggio** the letter indicating this pulldown menu changes according to the representation chosen.

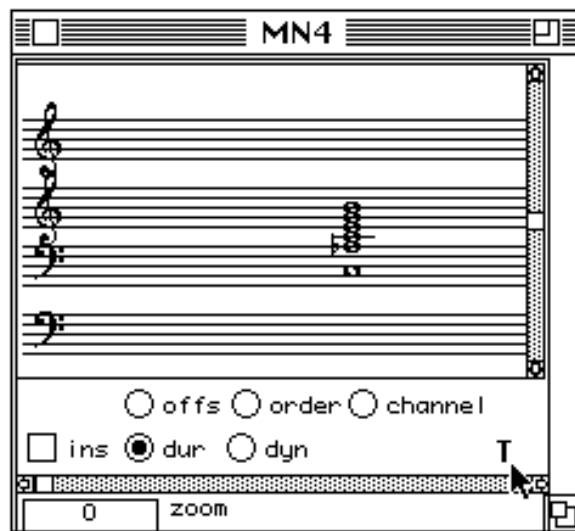


**chord** the default display



### ***time offset***

Shows the delay of attack-times between the different notes of the chord (in the following example, all notes have an offset of zero).

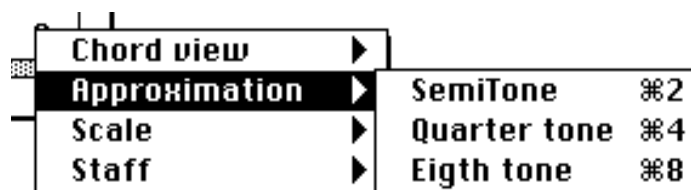


Time offset display with offset of 0.



## Approximation submenu

This item is equivalent to the approximation option in the submenu **Global options** under the **PWoper** menu, the only difference is that in this case the option is local (thus it effects only the particular **chord** module in which a change is made).



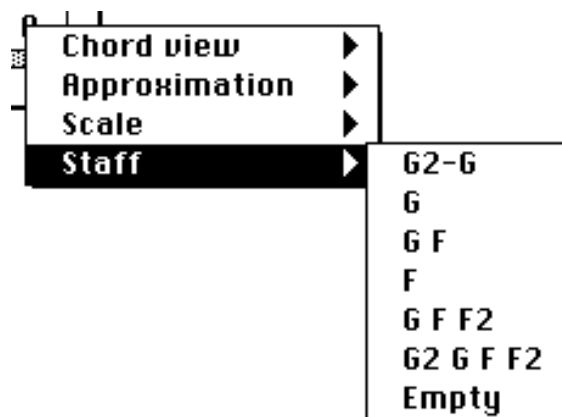
## Scale

is equivalent to the scale option in the sub-menu **Global options** under the **PWoper** menu, the only difference is that in this case the option is local (thus it effects only the particular **chord** module in which a change is made).

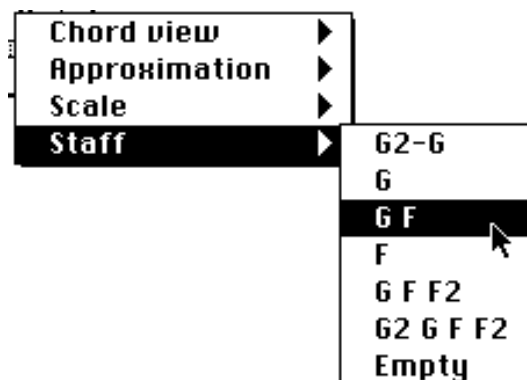


## Staff

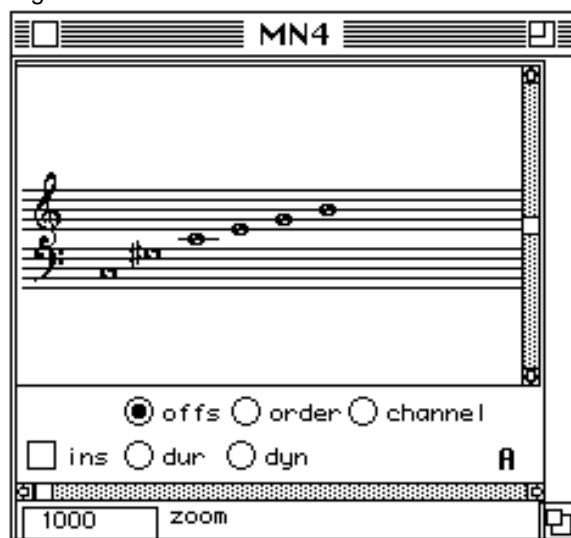
this menu presents different options for what musical staves are displayed within the **chord** module. It is possible to choose on what staves the notes will be shown, or even to display the notes without any staves.



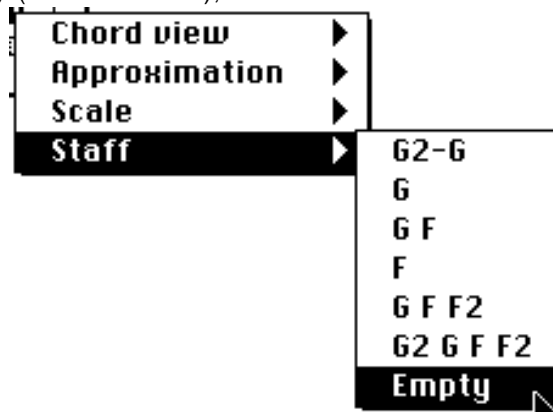
For example: the option **G F** (This option represents one staff with a treble ("G") clef and one staff with a bass ("F") clef.



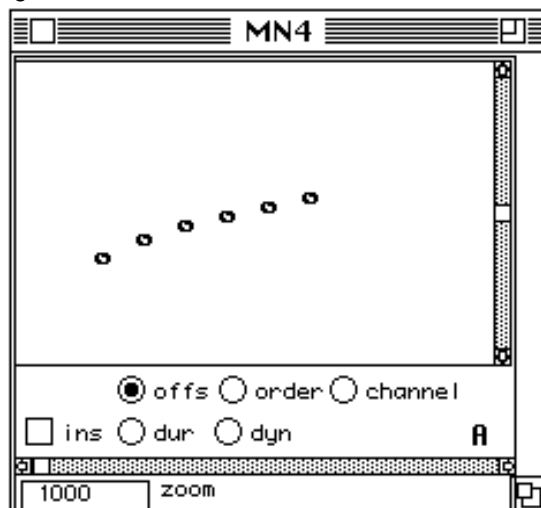
The display changes to the following :



When you select the item **Empty** (without staves),



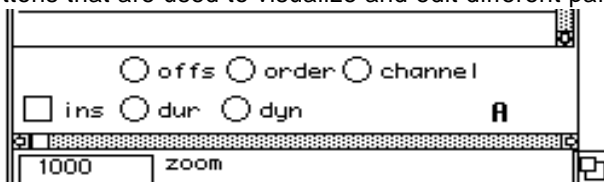
The display changes to the following :



Note: G2 and F2 represent staves with treble and bass clefs transposed up and down by two octaves

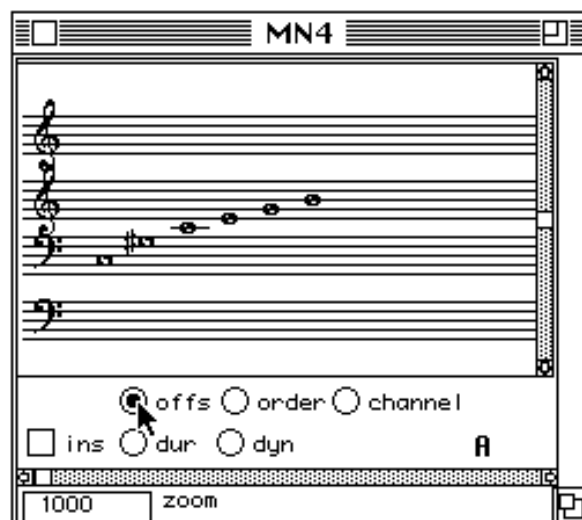
### ***The Edit Window Buttons***

The editor window has five buttons that are used to visualize and edit different parameters of the chord.

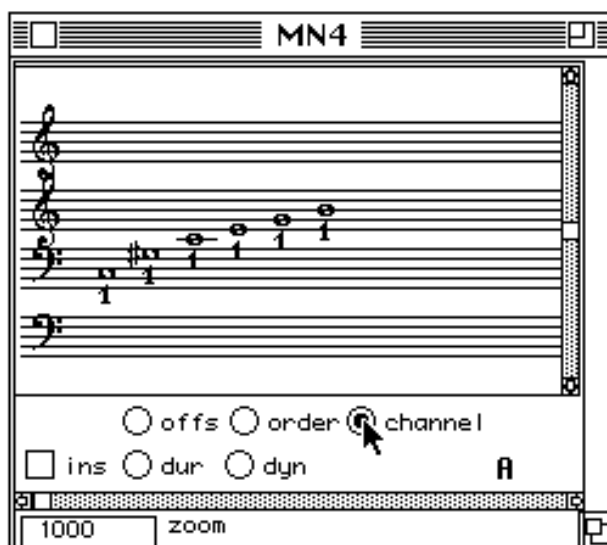


To see one of these aspects, click on the corresponding button and watch the display change.

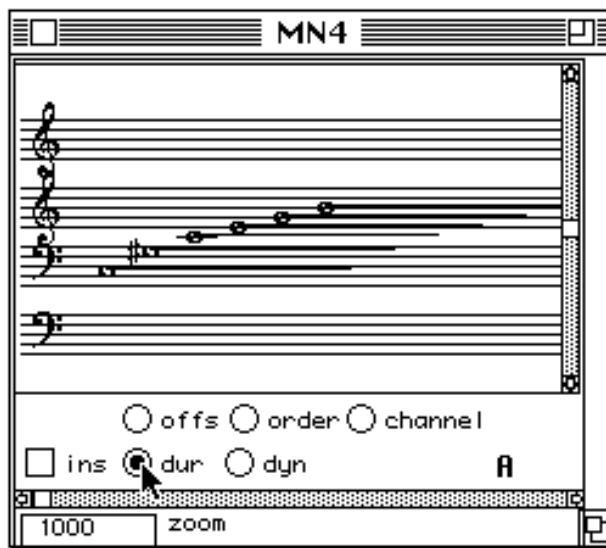
## *The offsets*



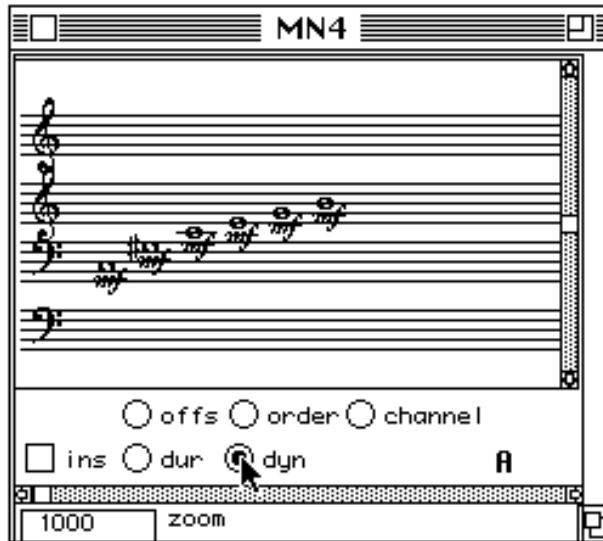
## *The channels*



## *The durations*



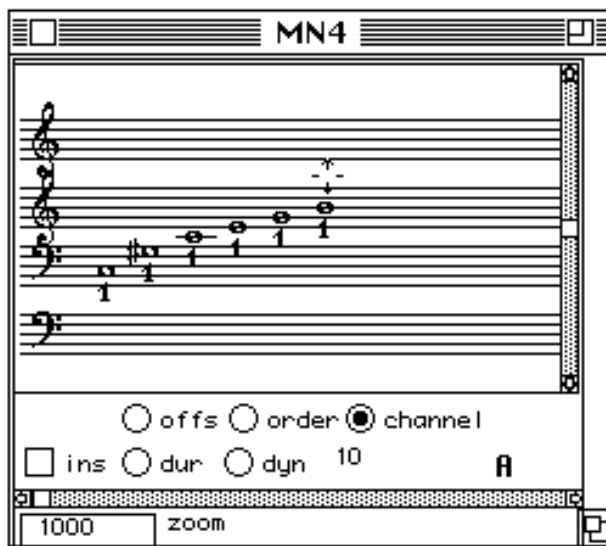
## *The dynamics*



In all cases, it is possible to edit each note individually for the selected parameter:

- Place the cursor on the note to be edited
- Press the mouse button
- Drag the mouse up or down; underneath the *channel* button a number appears that indicates the current value of the parameter being edited.

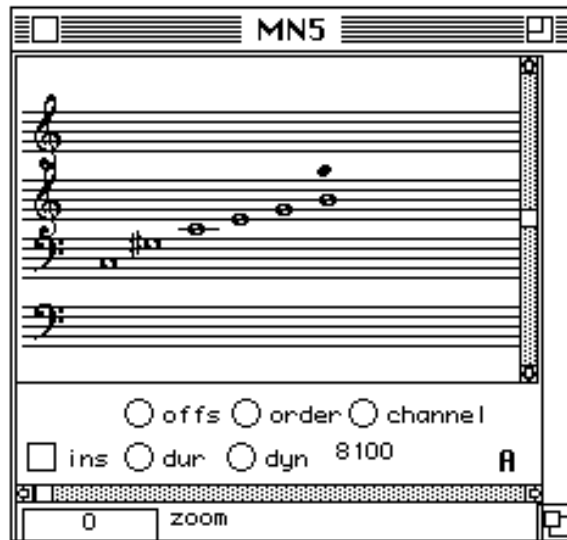
For example, to edit the MIDI channel associated with a note :



## Editing Notes

To edit notes (pitches), no button need be selected. Place the cursor on the note to be changed, click on the mouse and drag the it up or down to the desired location; underneath the *channel* button a number appears that indicates the current midicents value of the note.

In arpeggio or time-offset mode, several notes can be selected by clicking and dragging the mouse. Keyboard commands, such as up-arrow, down-arrow, etc., affect all selected notes.



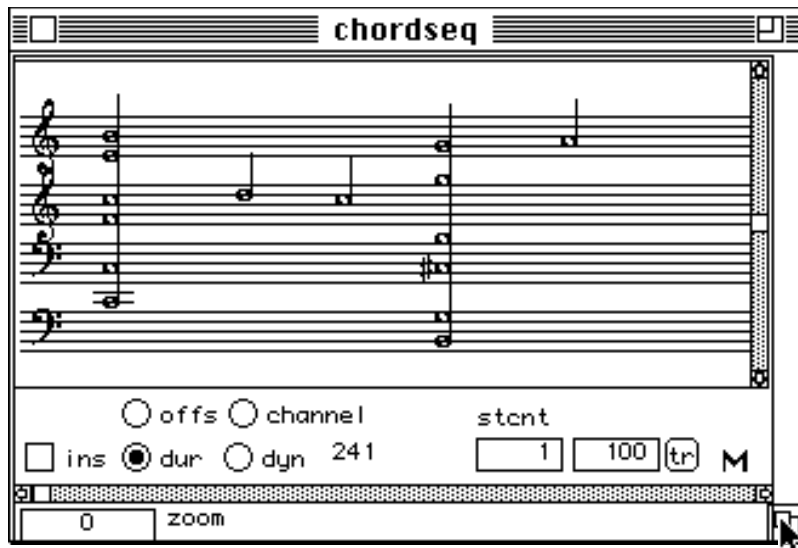
In time-offset or arpeggio mode, several notes can be selected by clicking and dragging the mouse. Keyboard commands, such as up-arrow, down-arrow, etc., affect all selected notes.

# The chordseq Module

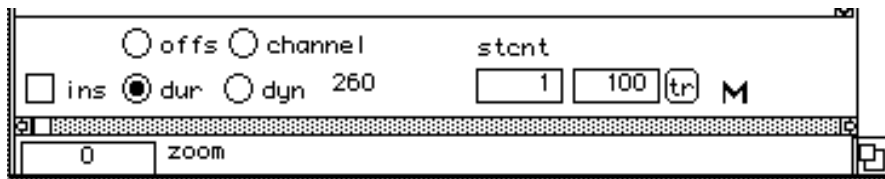
The **chordseq** module constructs *chord sequence objects*. It can also be used to edit notes, the gap between chords (or between notes of a melody), the duration of each note (or chord), the dynamic level of each note (or chord), the offset of each chord and the channels used for MIDI playback of the sequence.

## The chordseq Editor Window

The chordseq editor window edits and visualizes sequences of notes and chords.



It presents a series of buttons, input boxes and a pulldown menu (accessed by clicking on the letter *M*).



## Buttons

- |         |   |
|---------|---|
| offs    | Display chords with their offsets. Note offsets are only played when this option is selected. |
| dur     | Display chords with their durations   |
| dyn     | Display chords with their dynamics  |
| channel | Dislay chords with their MIDI channels  |

***Input Boxes***

- stcnt
- Number of staves in the display
- xxxx
- Interval of transposition, in midicents (positive or negative)
- tr
- Transposes the selected chord or chords by the interval xxxx cents. To select a chord, click once on its stem; each time the mouse clicks on tr, the selected chord is transposed by xxxx.
- The zoom changes the scale in which the sequence is displayed

***The Pulldown menu***

This menu is accessed by clicking on the letter *M* (at the bottom-right of the window) and possesses six options:

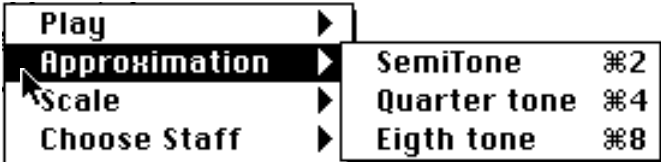
**Play**                    The Play submenu determines how the sequence is played.



- All
- Play entire sequence
- Visible
- Play the sequence currently visible in the editor
- Selected
- Play only the selected chords (or notes)
- Stop Playing
- Stop the sequence currently being played

***Approximation submenu***

Selects the local pitch approximation to be used within chordseq



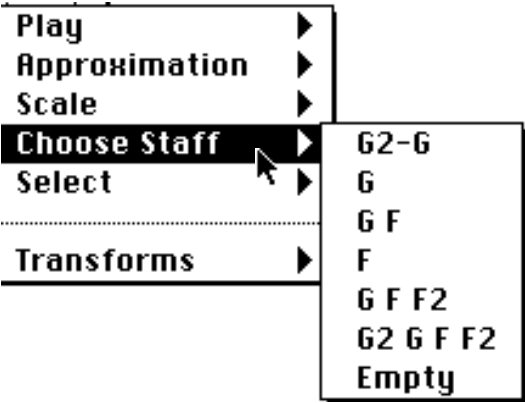
***Scale submenu***

**Scale**                    Allows the local choice of how chromatic alterations are displayed. Notes can be shown either as part of a chromatic scale or as alterations of the C major scale (see **PWoper** menu-5).





## Choose Staff submenu



This option determines the number and types of staves on which the sequence is displayed. Note that units can be subdivided into unequal parts.

## Select submenu

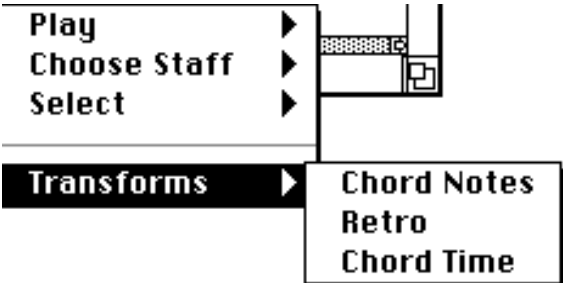


This item selects notes or chord:

- visible** Select the chords currently visible in the display
- all** Select the entire sequence

## Transforms submenu

This submenu contains several operations that can be performed on the sequence.



- Chord Notes** Allows changes (by addition) in the parameters of the chord objects. If selected, a dialog box with several choices appears:

			From	To
<input type="checkbox"/> velocity	<input type="checkbox"/> %	<input type="checkbox"/> range	0	0
<input type="checkbox"/> duration	<input type="checkbox"/> %	<input type="checkbox"/> range	0	0
<input type="checkbox"/> offset	<input type="checkbox"/> %	<input type="checkbox"/> range	0	0

The parameter to be changed is determined by selecting either velocity, duration, or offset. Parameters can be changed either as a percentage of the old value (click on the % square) or as absolute numbers (leave the % square blank). The range in which these parameters are effected can be specified in the "From" and "To" columns.

## Retro

Reverses the order of the selected chords

## Chord Time

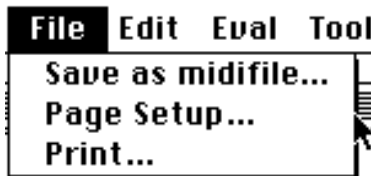
Alters attack times of the selected chords; invokes the following dialog box

<input type="checkbox"/> %	<input type="checkbox"/> range	0	0
----------------------------	--------------------------------	---	---

The type of change can be selected, either as a percentage or a range, and the limits of the range within which the modification takes place.

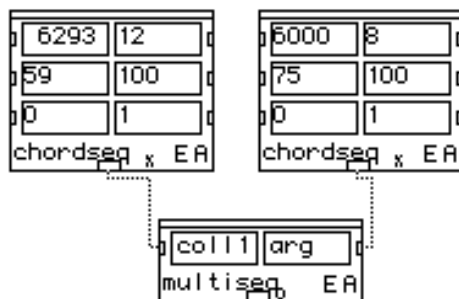
## The File Menu for chordseq

The **File** menu changes when you open a **chordseq** editor. This allows the possibility to save the contents of the editor as a Standard MIDI File, which can be read by sequencer programs, for example.

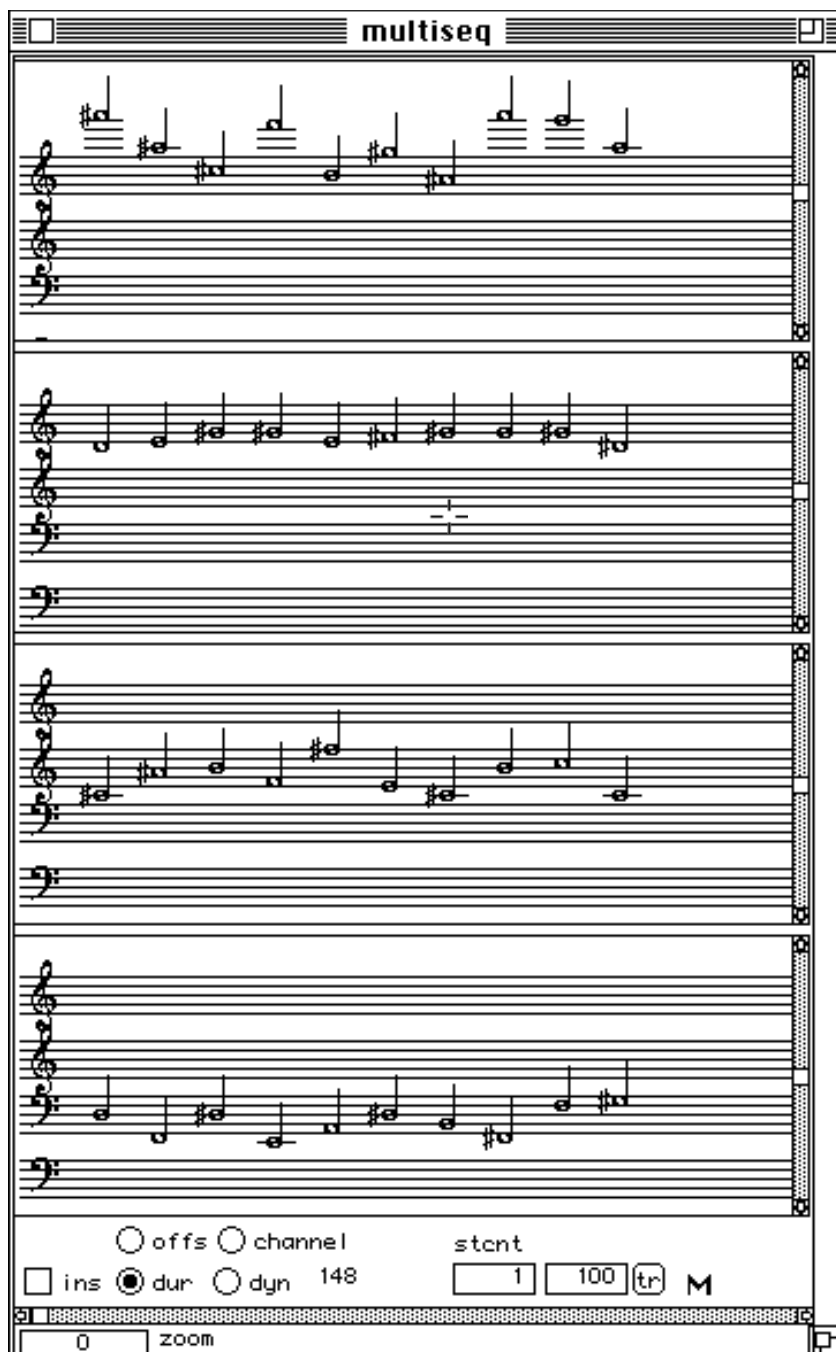


The other options in the **File** menu are standard print dialog boxes.

## The multiseq Module : Editing Polyphonic Sequences



The **multiseq** module (multiple sequence) box edits and plays polyphonic data. It accepts as input the output of one or more **chordseq** modules. The output of **multiseq** is a list of **chordseq** objects. This module works with the associated music notation editor, which displays as many systems as chord sequence inputs have been defined for the module.

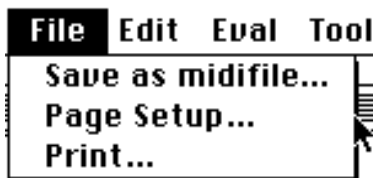


A pulldown menu is linked to the letter A just to the right of the output box. It offers the option of saving the module with all its chord sequences into a file. The multiseq module is state-preserving. It only changes its output if there is a module connected to one of its inputs (or if changes are made by hand in the editor, of course). This box can be opened by selecting it and pressing o from the keyboard or extended by Option-clicking bottom-right.

The editor window of the multiseq module contains buttons, input boxes and a pulldown menu on the letter M with functions which are essentially equivalents to those of the chordseq module.

## ***The File Menu for multiseq***

The File menu changes when you open a multiseq editor. This allows the possibility to save the contents of the editor as a Standard MIDI File, which can be read by sequencer programs, for example.



The other options in the **File** menu are standard print dialog boxes.

# The rtm Rhythm Editor Module

PatchWork's **rtm** module displays, edits, and plays a chord sequence. The rhythmic structure of the sequence is defined by a list of measures, a description of subdivisions of the beat, and a list of tempos.

The **rtm** module can be used to create a *measure sequence object*, to edit notes, the gap between chords (or between notes of a melody), the duration of each note (or chord), the dynamic of each note (or chord), the offset of each chord, and the channels used for MIDI playback of the sequence.

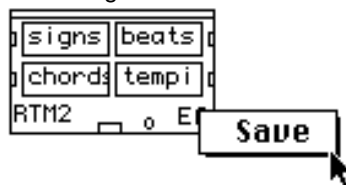
A lock protects the contents of an **rtm** module (whether edited or calculated), or prevents reevaluation.



The module is extendible.



It has a pulldown menu that saves the module along with its current contents.

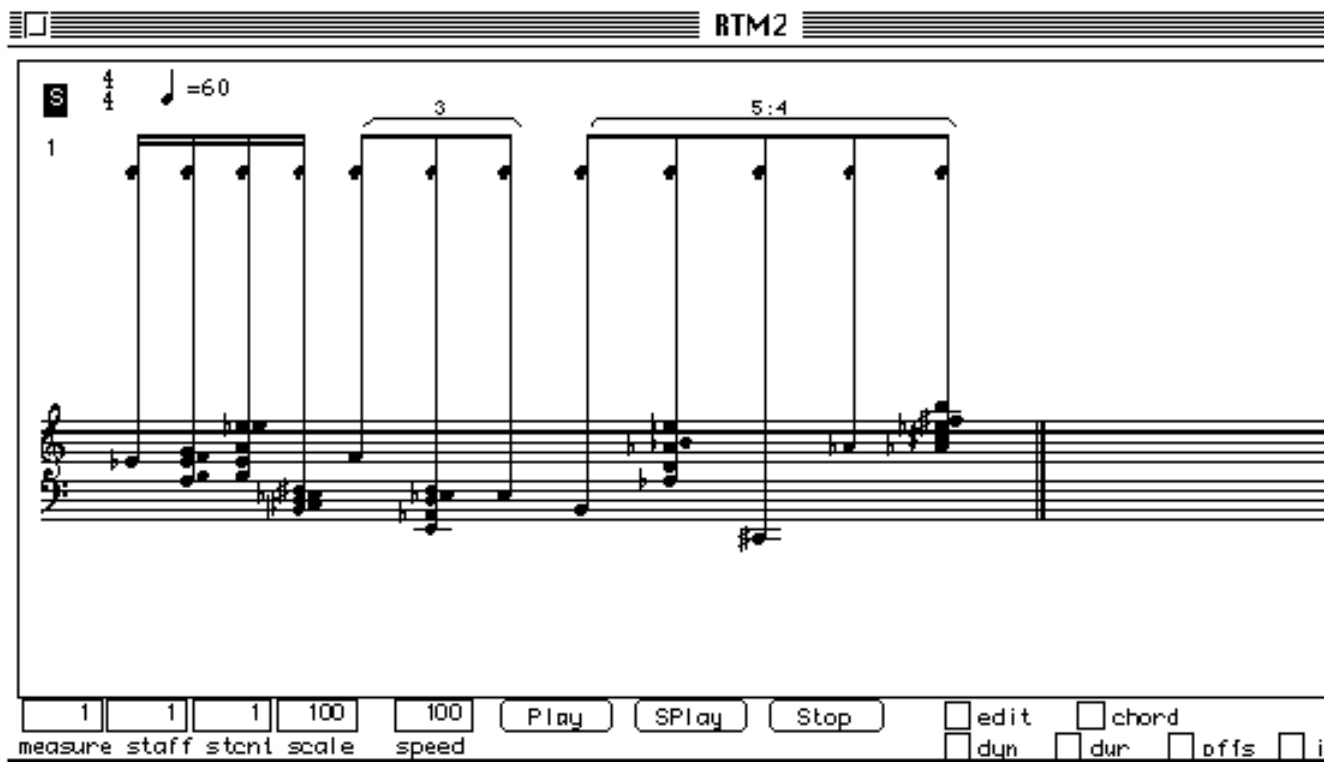


The associated **rtm** editor window can edit and display data coming from its four (or five) inputs.

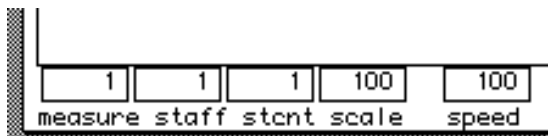
1 1 1 67 100 Play SPlay Stop

measure staff stcnt scale speed

# The RTM Editor Window



This window contains a series of buttons, input boxes and a pulldown menu (under the letter *M*) that invokes operations on the sequence being edited.



The first five input boxes let users edit (by dragging the mouse up and down) the following parameters :

- measure* Measure at which the display begins
- staff* Specifies staff the display begins
- stcnt* Number of staves displayed in the editor window
- scale:* Scale of the display
- speed* Speed at which the sequence is played back

The following buttons are activated when clicked on by the mouse :



*Play* Plays the sequence

*Splay* Plays the sequence while advancing the screen to display what is being performed (if the computer lacks sufficient speed, this mode can be less accurate rhythmically than the simple play mode)

*Stop* Stops the sequence currently being played

Next is a group of six small checkboxes:



These boxes control the display and editing of various parameters in this window. By clicking on:

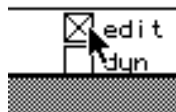
*dyn* Dynamics of the notes or chords are displayed in musical notation

*dur* Durations of the notes or chords are displayed in proportional notation

*offs* Offsets of notes or chords are displayed in proportional notation as grace notes. The Play button only plays these if this option is selected.

*ins* Instrument assigned to each note or chord is displayed.

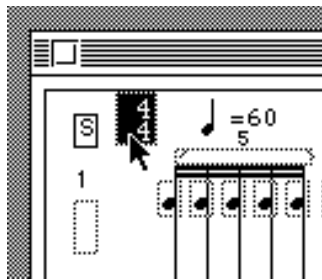
Clicking on the *edit* box puts the window into the edit mode :



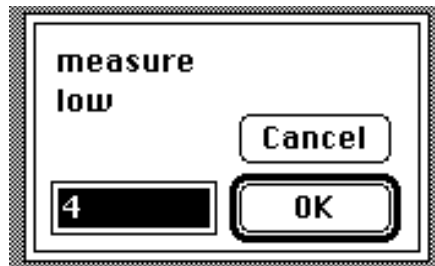
This causes a number of zones to appear within dotted boxes; these zones can be edited by clicking within them.



This lets users edit measures. Double-click on the square surrounding the time signature.

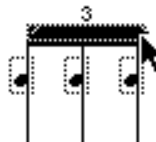


The following dialog box appears:

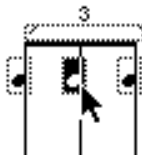


This edits the lower number of the time signature.

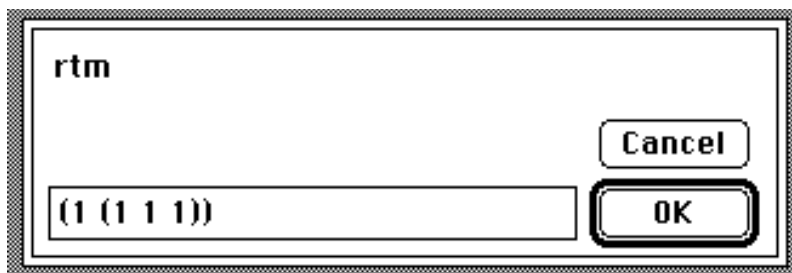
One can also edit subdivisions on several levels. As can be seen in the display, each figure contains many editable levels. The first level corresponds to the entire figure:



and, in the current example, a second level corresponding to each note of the figure:



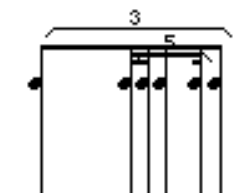
If we double-click on the edit zone corresponding to the first level, the following dialog box appears:



The first element in the list indicates how many beats are involved in the figure (in our case one beat). The second element (a sub-list) indicates how the concerned beats are divided (in our case, the one beat is sub-divided into three equal parts: indicated as (1 1 1) )

If we double click on the edit zone corresponding to the second level

the second element is an empty list, indicating that at this level there are no new subdivisions. (Don't forget that in order to select or edit material within the RTM editor the window must be in the "edit" mode.) We could, however, edit that empty list, changing it into the following ( 1 ( 1 1 2 1 )), after this change the rhythm would appear as :

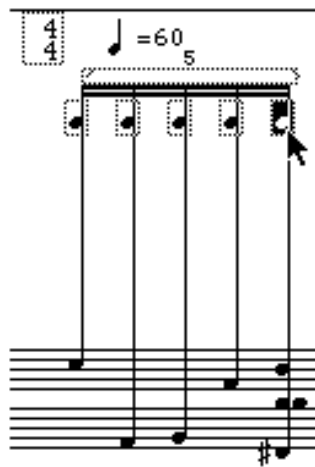


In our particular case, if we now click again on the first level, the new rhythm will be represented as the following tree structure:

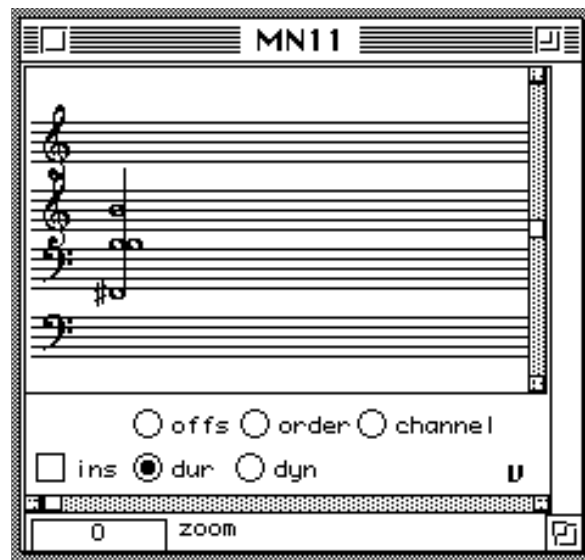
where the second element in the figure is represented as ( 1 (1 1 2 1)), the number of time units and their sub-division.

To edit chords, click on the *chord* box:

this enables editing on the note level; double-click on the edit zone representing the lowest level.

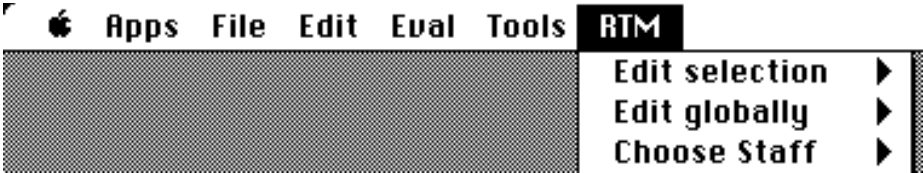


This causes an editor window (similar to that of the chord module) to appear:

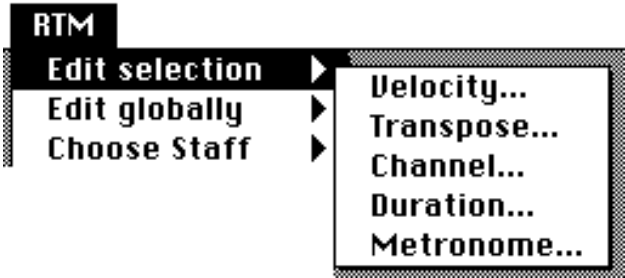


# The RTM Menu Bar

When a RTM editor window is active, a new menu appears in the menu bar :

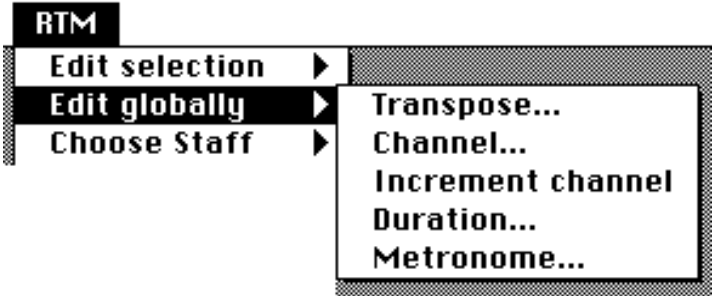


This menu contains three main options: **Edit selection**; **Edit globally**, and **Choose staff**. We explain each in turn.



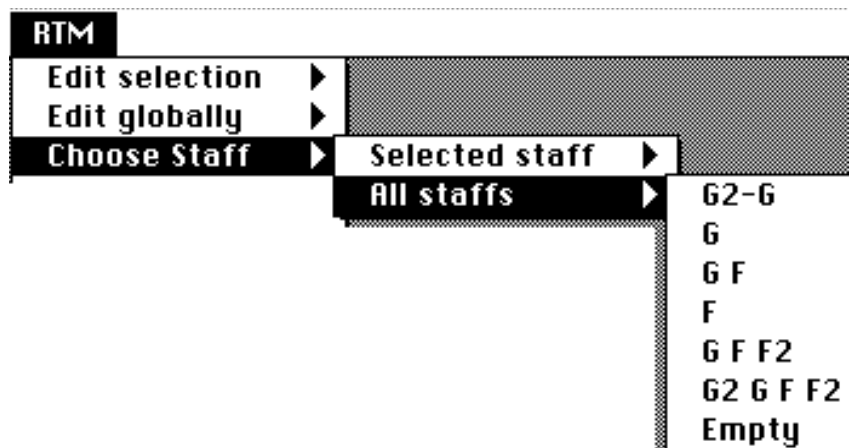
**Edit selection** Edit MIDI velocities and channels, durations (as percentages), metronome markings and to transpose the selected figures (by an interval expressed in midicents). (This option is useful only for the module **poly-rtm**.)

**Edit globally** Perform operations globally on the entire sequence (if the window is in the edit mode).

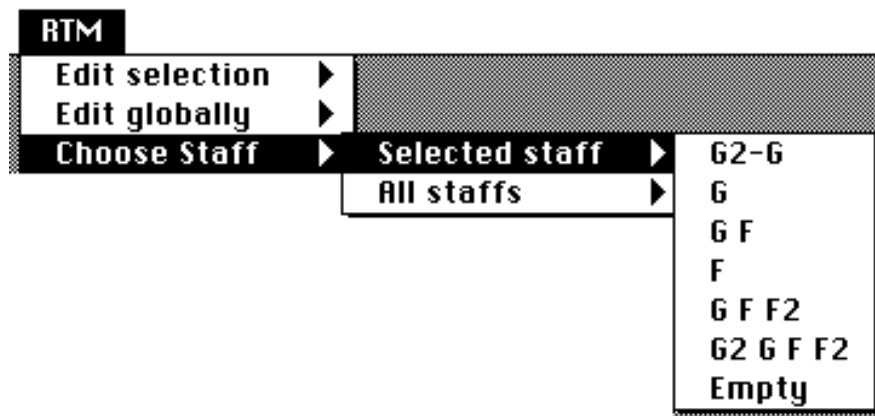


**Choose staff** Manipulate the staves in the edit window. Presents two items.

**All staffs** Select the staff configuration in the editor window's display

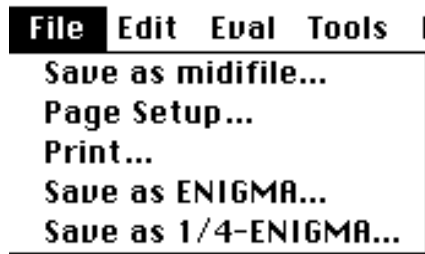


**Selected staffs** Allows certain staves to be selected for playback or editing of the selected voice or voices. (Useful only for the module **poly-rtm**.)



## ***The File Menu under RTM***

The File menu changes when you open an **RTM** editor window, offering several new possibilities:



**Save as midifile...** Save the contents of the file in Standard MIDI File format; this lets it be read by sequencer programs

**Page Setup...** Specify printing parameters

**Print...** Print the contents of the editor window

**Save as ENIGMA...** Save the contents of the editor in equal-tempered ENIGMA format; this lets it be read by the Coda Finale notation program

**Save as 1/4-ENIGMA...** Save the contents of the editor in quartertone ENIGMA format; this lets it be read by the Coda Finale notation program

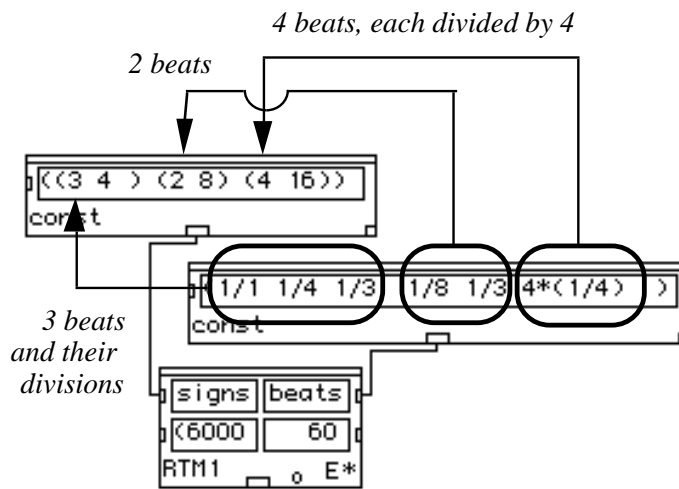
## ***RTM inputs: signs, beats, chords, tempi, and objs***

We will see the five inputs to RTM: signs, beats, chords, tempi, and objs.

### ***signs***

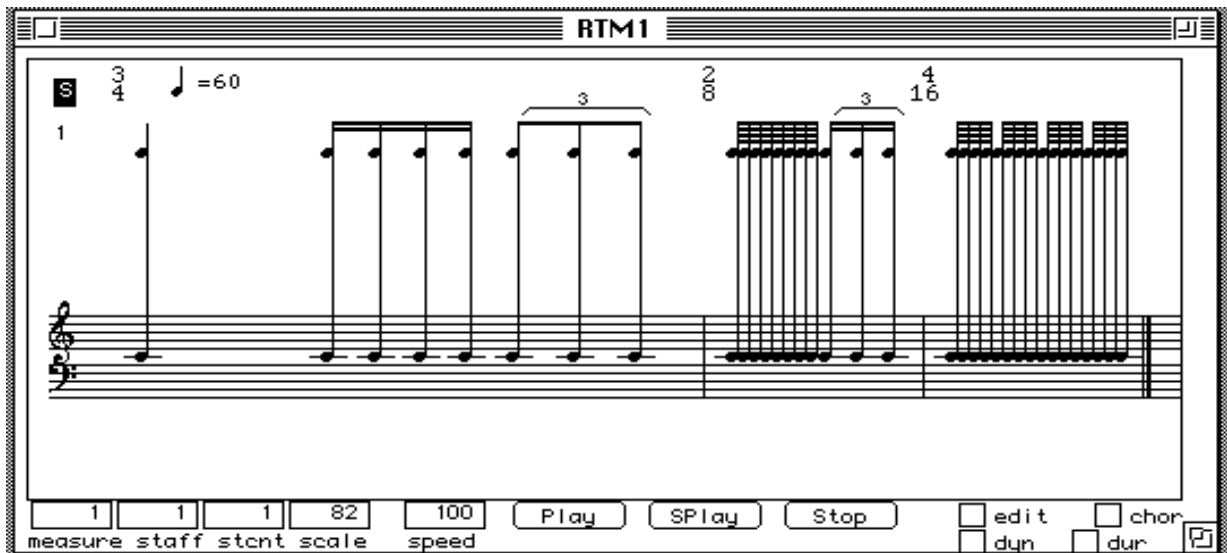
The *signs* input to the rtm module receives a list of measures, expressed as a list of lists. Each sublist indicates the time signature of a measure. For example: ((3 4) (2 8) (4 16)) indicates one measure in 3/4, one measure in 2/8 and one measure in 4/16.

The values input for *signs* must be consistent with those input for beats and chords. If there are inconsistencies between these three inputs (for example the subdivisions, number of notes, and the list of measures are incompatible), the rtm module tries to adjust the data. Warning: this can, in some cases, change the entered values.

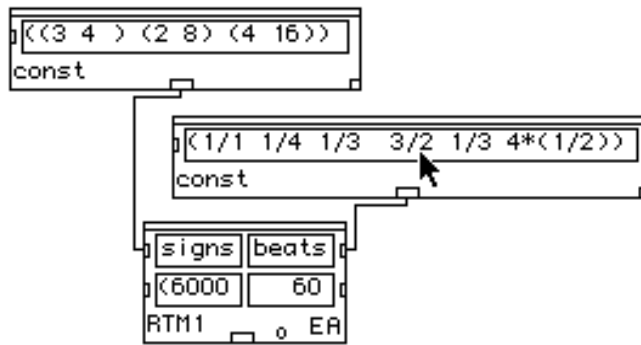


In the above configuration everything is consistent. The list coming to the beats input indicates how the beats in the measures are divided up.

- the measure in 3/4 contains one quarter note (1/1), four sixteenth notes (1/4) and three triplet eighth notes (1/3);
- the measure in 2/8 contains eight sixty fourth notes (1/8) and three triplet sixteenth notes (1/3); and
- the measure in 4/16 contains sixteen sixty-fourth notes (4\* (1/4)). (This notation is derived from that of the module **expand-list**, see its documentation.)



It is also possible to divide beats into unequal parts, such as (4 (1 2 1 3 2 1)); in other words we can divide four beats into six unequal parts. Try this example yourself! If we try to alter this, for example, by changing the list of beats:



We have changed the subdivisions of the measure in 2/8. If we now reevaluate the **rtm** module, the following message appears in the Listener window:

```
;Warning: measures (2 8) and beats do not agree. Measure(s) will be changed
; While executing: #standard-method patch-work::patch-value
(patch-work::c-patch-score-voice t)
PW-#patch-work::c-measure-line #xC4DA69,
```

This indicates that modifications were made to our data.

If we look at the edit window :

we see that the second measure has been changed and an additional measure in (1/16) added at the end.



## beats

The *beats* input to the *rtm* module is a list of subdivisions of the basic beat. For example:

(1/4 1/3 2/5).

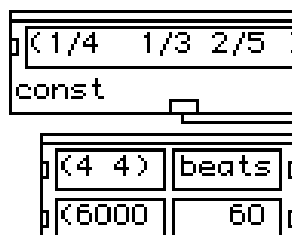
Subdivisions can be indicated in two different ways: as a fraction or explicitly using a nested list notation. The next sections describe each case in turn.

### Case 1: Fractional Representation of Subdivisions of a Beat

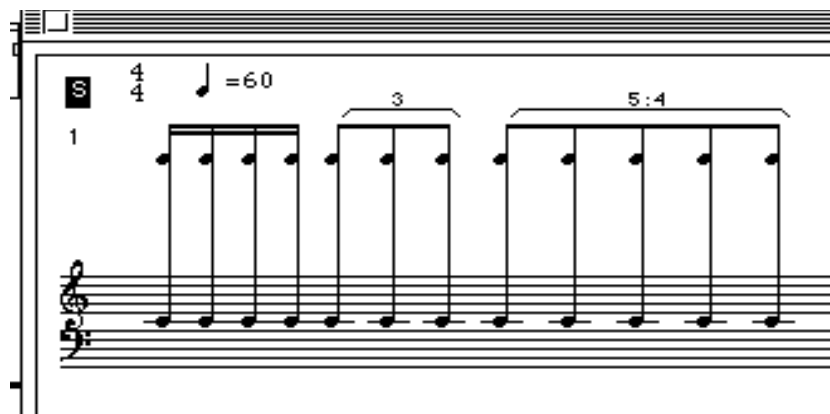
Subdivisions can be indicated as a fraction indicating either a subdivision of the beat or a substitution. Here are two examples of beat subdivision:

- (1/4) = one beat, divided into four parts
- (1/3) = one beat, divided into three parts
- Here are two examples of fractional subdivision:
- (2/3) = three figures in the place of two beats, or
- (2/5) = five figures in the place of two beats

In both cases the *rtm* module always fills the beat (or several beats in the case of a substitution) with the desired number of subdivisions. For example, if (1/4) is indicated one time for a measure in (4 4), *rtm* fills the time of one beat with four sixteenth notes. Please be ware that there must always be consistency (parity) between the data input for signs with that of beats and chords. For example: If a measure in (4 4) is indicated, the subdivisions or substitutions for each beat of that measure must also be indicated.



After evaluation, this yields:

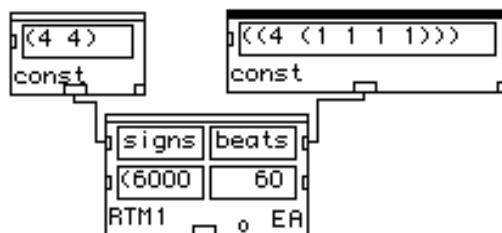


## Case 2: Explicit Representation of Subdivisions of a Beat

Subdivisions of a beat can be explained explicitly with a double list (indicating how the beats are to be divided). For example, in the list:

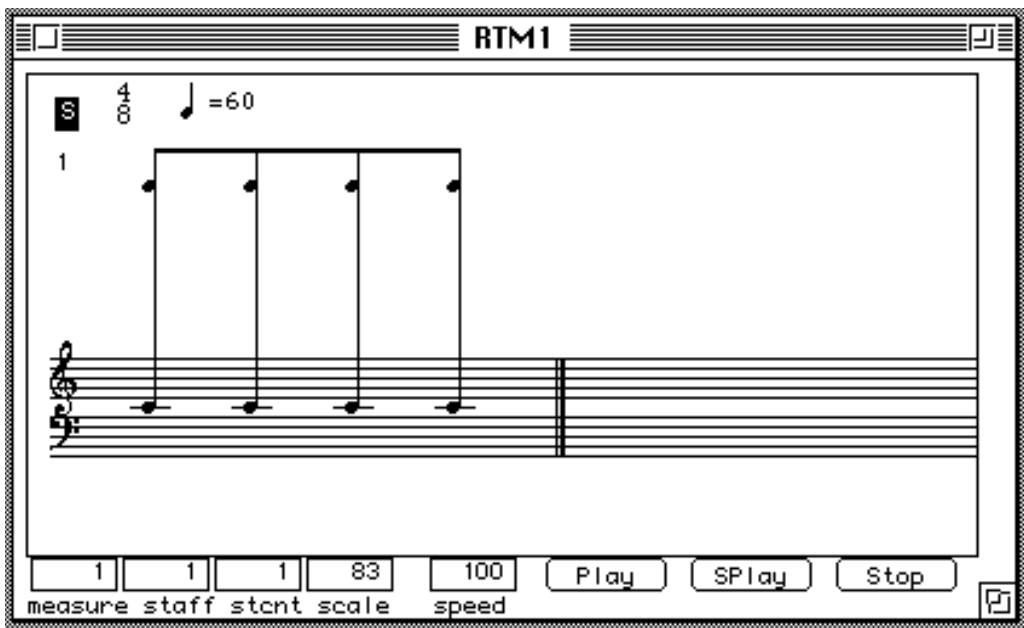
(4 ( 1 1 1 1 ))

the first element of this list indicates the number of beats effected and the second element (a sublist) indicates, proportionally, into how many parts those beats should be subdivided. For example: (4 ( 1 1 1 1 )) indicates that four beats are divided into four equal parts when evaluated



the following results:

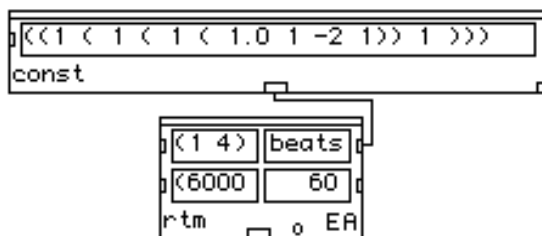
if we use the same input for the argument beats, but with a measure in (4 8):



Remember that subdivisions are always a function of the selected metric indication.

- Here is the adopted system for representing beats  
a positive integer number represents a subdivision.
- a negative integer number represents a rest proportional to the number
- a positive floating-point number means that this element is tied to the previous element

.Try editing : ((1 (1 (1 (1.0 1 -2 1)) 1)) )



After evaluation, we obtain :

RTM1

S  $\frac{1}{4}$  ♩ = 60

1

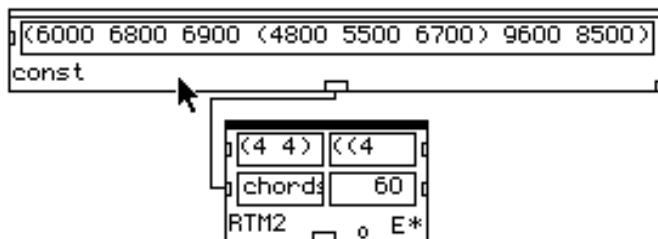
3 5

1 1 1 400 100 Play SPlay Stop

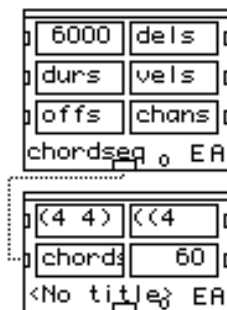
measure staff stent scale speed

## chords

The *chords* input to the **rtm** module can receive as input either a list of chords, a list of notes in midicents, or a list of chord objects (e.g. a list of outputs of several **make-chord** modules).



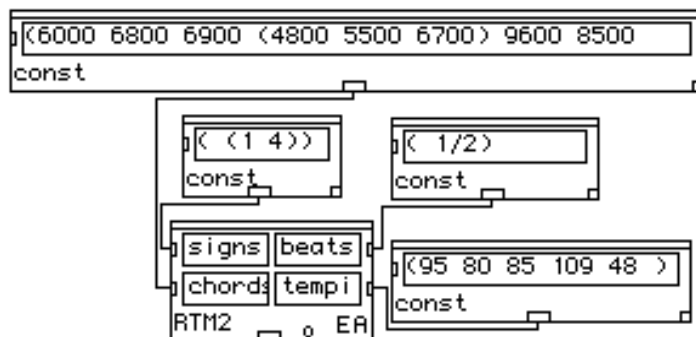
It also handles the output of a **chordseq** module:



This input flexibility allows a great deal of control over the characteristics of the chords (or notes) to be used. Remember that the *chords* input accepts two-level lists in which the sub-lists represent chords and values on the first level represent notes.

## *tempi*

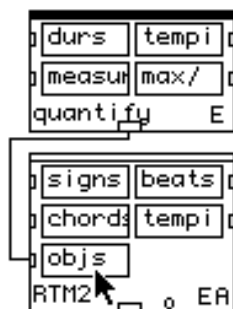
The *tempi* input to the **rtm** module is a list of metronome markings in the form of a simple list



After evaluation, the previous example produces the following image:

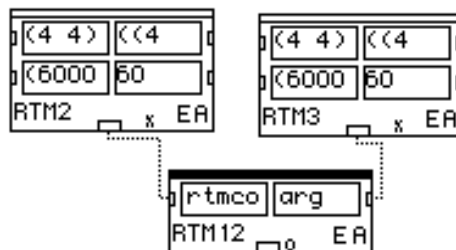
## ***objs***

The *objs* input to RTM is used when one wants to quantize real durations (in hundredths of a second) into musical rhythmic notation. In this case, connect the module **quantify** to this input:



Important: Once a **quantify** module is connected to this input, the data received by the inputs *signs*, *beats* and *tempi* of the **rtm** module is no longer used. The object provided by the **quantify** module contains all the necessary information.

## The *poly-rtm* Module: Editing Multiple Rhythmic Sequences



This module allows one to edit and play many rhythmic sequences simultaneously. It only plays sequences generated by the **rtm** module. If the out-box of a **poly-rtm** box is double-clicked, then all the measure-line objects are taken inside the **poly-rtm** box.

The module can be locked (to avoid new evaluations of the patch that is under **chord**) to keep its contents by clicking on the small *o* in the lower right of the module. An *o* indicates that the module is open.

You can enter an editor for the **poly-rtm** object either by selecting the module and typing the letter *o*, or by double-clicking on the module. See the image on the next page.

Type *H* with the editor window open for more information about keyboard commands.



**S**  $\frac{3}{4}$   $\text{J} = 92$   $\frac{1}{4}$   $\text{J} = 69$   $\frac{2}{4}$   $\text{J} = 15$

1

**S**  $\frac{3}{4}$   $\text{J} = 92$   $\frac{1}{4}$   $\text{J} = 69$   $\frac{2}{4}$   $\text{J} = 15$

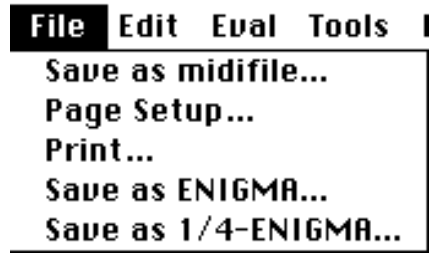
2

1 1 1 100 100 Play SPlay Stop

measure staff stent scale speed

## ***The File Menu for poly-rtm***

The File menu changes when you open a **poly-rtm** editor window, providing several new possibilities:



**Save as MIDI file...** Save the contents of the file in Standard MIDI File format; this lets it be read by sequencer programs

**Page Setup...** Specify printing parameters

**Print...** Print the contents of the editor window

**Save as ENIGMA...** Save the contents of the editor in equal-tempered ENIGMA format; this lets it be read by the Coda Finale notation program

**Save as 1/4-ENIGMA...** Save the contents of the editor in quartertone ENIGMA format; this lets it be read by the Coda Finale notation program

# 8 Bibliography

Amiot, G. Assayag, C. Malherbe, and A. Riotte, 1986, « Duration structure generation and recognition in musical writing », *Proceedings of the 1986 International Computer Music Conference*, International Computer Music Association, San Francisco.

Barrière, J.-B. 1990, « Devenir de l'écriture musicale assistée par ordinateur: formalisme, forme, aide à la composition », *Analyse Musicale*, 3e trimestre, pp. 52-67

Baisnée, P.-F., J.-B. Barrière, M. A. Dalbavie, J. Duthen, M. Lindberg, Y. Potard, and K. Saariaho, 1988, « Esquisse: a Compositional Environment », *Proceedings of 1988 International Computer Music Conference*, International Computer Music Association, San Francisco.

Boynton, L., J. Duthen, Y. Potard, and X. Rodet., 1986, « Adding a Graphical Interface to FORMES », *Proceedings of the 1986 International Computer Music Conference*, International Computer Music Association, San Francisco.

Duthen, J., and M. Laurson, 1990, « A compositional environment based on preFORM II, PatchWork, and Esquiss », *Proceedings of the 1990 International Computer Music Conference*, International Computer Music Association, San Francisco.

Laurson, M., and J. Duthen, 1989. "PatchWork: A Graphical Programming Environment in PreFORM." *Proceedings of the International Computer Music Conference*, International Computer Music Association, San Francisco.

Lerdahl, F., and Y. Potard, 1986, « La composition assistée par ordinateur », *Rapports de Recherche*, 41, Ircam.

Rodet X., and P. Cointe. , 1984., « Formes: Composition and Scheduling of Processes », *Computer Music Journal* 8(3). Reprinted in C. Roads, ed. 1989. *The Music Machine*, The MIT Press, Cambridge, Massachusetts, pp. 405-426.

## Lisp References

Apple, 1991, *Macintosh Common Lisp Reference*, Apple Computer, Cupertino.

Koschman, T, 1990, *The Common Lisp Companion*, John Wiley & Sons, New York.

Steele, G, 1990, *Common Lisp: The Language*, Digital Press, Bedford.

Touretzky, T, 1989, *Common Lisp*, Benjamin Cummings.

Winston, P., and B. Horn, 1988, *LISP*, Third edition, Addison-Wesley, Reading.

# Index

## A

A 86  
Abort 33  
absin 39  
absout 39  
Abstract 33  
Abstract option in the PWoper menu 39  
Agon C. 8  
Allegro Common Lisp 7  
Apple 33, 47  
Approximation 33, 46  
Approximation submenu 105  
Apps menu 30  
arithm-ser 77  
arpeggio mode 110  
Assayag G. 8

## B

Baisnée P.-F. 8  
Barrière J.-B. 8  
beats input to the rtm module 129  
Bonnet A. 8  
Boyton L. 8  
BPF 58  
bpf 68  
BPF Editor Window 93  
Break 33, 43  
breakpoint function 88

## C

Chabot X. 8  
CHANT 7  
Choose Staff submenu 113  
Chord 103  
chord 10, 88, 99  
Chord Editor Module 99  
Chord Editor Window 101  
Chord Object Fields 98  
Chord Sequence Object Fields 98  
chord sequence objects 111  
Chord View Submenu 103  
chords input to the rtm module 133  
chordseq 10, 68, 88, 111, 133  
chordseq Editor Window 111  
chordseq editor window 111  
chordseq, 71  
Chromatic 45  
Chromatic alterations 45, 112  
CLOS 7  
Coda Finale 7, 126, 138  
Combinatorial 56  
Common Lisp 13, 36

Common Lisp menu 29  
Connecting Modules 71  
Continue 33  
CRIME 7  
Csound 7  
Cut connection 20

## D

D 86  
d 84  
Dalbavie M.-A. 8  
Data 48  
Debugging Operations 33, 44  
Definition 32  
defunp 42  
Delete Config 33, 47  
Digidesign 33  
Doc on-line 14  
Documentation 32, 36, 81  
Documentation options 32  
Durieux F. 8  
Duthen J. 2, 8

## E

e 85  
Editors 10  
ENIGMA 126, 138  
Esquisse 7, 35  
Evaluating a module 77  
Evaluation 10, 33, 46  
Extensible modules 71

## F

File menu 31  
Fineberg J. 2, 8  
flip-mode 89  
Floating-point 80  
FMAKUNBOUND 84  
FORMES 7  
Fractional Representation of Subdivisions of a Beat 129  
Function 51

## G

g\* 65  
Gerzso A. 8  
Global 45  
Global options 33  
g-log 66

## H

h 83

Hash tables 80  
Help Files 82

## I

i 84  
Integers 80  
interpolation 81  
Interrupt operations 33, 43  
Invoking a Menu from a Module 70  
Invoking a module 66  
Invoking Editor Windows 68  
lovino F. 8

## K

Kernel 34  
kernel menu 48  
keywords 42

## L

Laurson M. 2, 8  
Lindberg M. 8  
Lisp 13, 81  
Lisp Data Types 80  
Lisp function... 33  
Lisp Programming 13  
Lisp References 139  
LISPcontains modules that perform standard Lisp functions. 53  
Listener window 28  
Lists 80  
Load Abstracts 33  
Load Abstracts... option 47  
Load Library... 33, 47  
Locking Modules 73  
lst-ed 68

## M

M 112  
make-chord 133  
Malt M. 2, 8  
measure sequence object 118  
Measure Sequence Object Fields 98  
measure-line 99, 128, 136  
Menu bars 29  
Menus 28  
Metronome markings 134  
MIDI all-notes-off 33, 47  
MIDI File 138  
Midi Manager 33, 47  
MIDI Operations 33, 47  
MIDI Reset 33, 47  
MIDI SetUp 33, 47  
MIDI synthesizers 7  
MIDI velocities 124  
midic 98  
Module Help 14  
Modules 9, 65  
Mosaïc 7

multi-bpf 10, 88, 89, 91  
multi-bpf Editor Keyboard Commands 97  
multi-bpf Editor Module 88  
multi-bpf Menu Bar 94  
Multiple instances 67  
multiseq 10, 88, 115  
Murail T. 8, 16  
Music 35  
Music menu 61

## N

Next BPF from lib 94  
Nicolas F. 8  
Non-OMS Applications 33  
Note Object Fields 98  
nth-overtones 15, 69

## O

o 85  
Objects 80  
objs 135  
objs input to RTM 135  
OMS 33, 47  
OMS Midi Setup 33  
On-line Documentation 81  
Opcode 33, 47  
Open patch 31  
Opening an Abstraction 69  
Operation that affects the PW program image 33  
Operations that create modules 33  
Optional Inputs 71

## P

Patch 9, 15  
PatchWork Image 15  
PatchWork patch window 29  
Play 46  
Play Option 33  
poly-rtm 10, 88, 136, 138  
Potard Y. 8  
PreFORM 7  
PW menu 29  
PW-debug-ON 33  
PW-debug-ON option 44  
PWoper menu 32, 36

## Q

quantify module 135  
quantize 135  
quarternote 126

## R

r 83  
Rationals 80  
Removing extensible module inputs 71  
Reset BPF lib 94  
Rhythm editor 30

Rhythm Editor Module 118  
Roads C. 2  
RTM 88  
rtm 10, 98  
RTM Editor Keyboard Commands 126  
RTM inputs  
126  
RTM Menu Bar 124  
rtm module 118  
Rueda C. 2, 8

**Y**  
Y 87

## **S**

Saariaho K. 8  
SampleCell 33  
Save Current Config 33, 47  
Save image option 47  
Save with MN 31  
Save with MN as... 31  
Saving and Opening Patches 26  
Scale 33, 105  
Selecting a Module 74  
show error box 33  
signs input to the rtm module 126  
Simultaneous breakpoint functions 10  
Staff 105  
Standard MIDI File 126  
Stretching and Shrinking Modules 73  
Strings 80  
Subdivisions of a beat 130

## **T**

t 81, 85  
tempi input to the rtm module 134  
text-win 68  
time-offset mode 110  
tlist 91  
Transforms submenu 113  
Tutorial 81, 85  
Type 81

## **U**

UserLib 35

## **V**

v 85  
vlist 91

## **W**

Waxman D. 8  
Window Comment 32  
Window Help 14  
Wins 35

## **X**

X 87  
x-append 71