Ircam
documentation

- Research reports
- Musical works
- Software

PatchWork

PW-Max

An Interface between PatchWork and Max

First English Edition, March 1997

IRCAM Centre Georges Pompidou

This manual was written by Xavier Chabot, and was produced under the editorial responsibility of Marc Battier, Marketing Office, Ircam.

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

The library was written by Xavier Chabot.

First edition of the documentation, March 1997.
This documentation corresponds to version 2.5 or higher of PatchWork, and to version 1.0 of the library.

Ircam
1, place Igor-Stravinsky
F-75004 Paris
Tel. 01 44 78 49 62
Fax 01 44 78 15 40
E-mail ircam-doc@ircam.fr

IRCAM Users' group

The use of this software and its documentation is restricted to members of the Ircam software users' group. For any supplementary information, contact:

Département de la Valorisation
Ircam
Place Stravinsky, F-75004 Paris

Tel. 01 44 78 49 62
Fax 01 44 78 15 40
E-mail: bousac@ircam.fr

Send comments or suggestions to the editor:
E-mail: bam@ircam.fr
Mail: Marc Battier,
Ircam, Département de la Valorisation
Place Stravinsky, F-75004 Paris

http://www.ircam.fr/forumnet

Contents

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.

Résumé

La librairie PW-Max, écrite par Xavier Chabot, permet un liaison fonctionnelle entre PatchWork et différentes versions du programme Max (version pour Macintosh et, à l'époque de la réalisation de cette librairie, en 1994, avec la version Max pour la Station informatique de l'Ircam.

La librairie PW-Max est le plus souvent utilisée en conjonction avec la librairie SpData, qui a fait l'objet d'une distribution en mars 1997.

Le chapitre 1 introduit les concepts utilisés par la librairie.

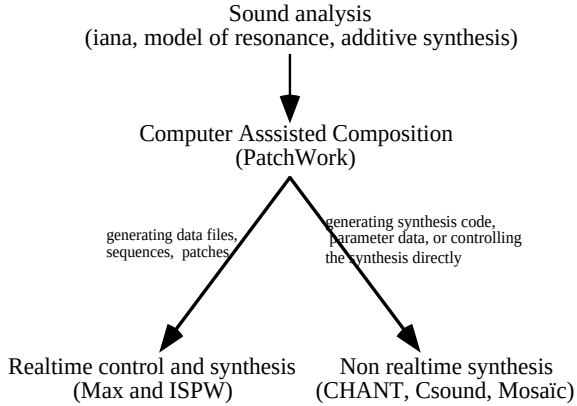
Plusieurs objets premettent de générer des données directement utilisables dans Max. Ces objets sont décrits au chapitre 2.

Ce manuel comprend une description de la liaison PatchWork-Max, et est illustré par des exemples regroupés dans la partie 3, intitulée "PW-Max tutorial patches".

1 Introduction

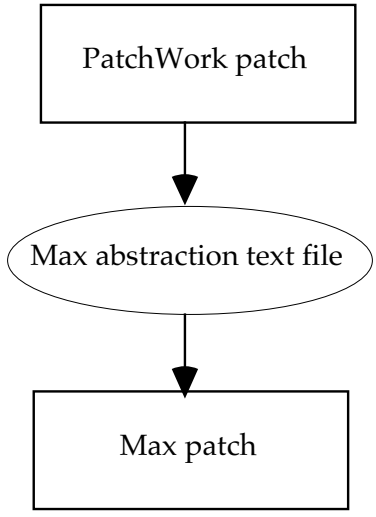
1.1 Synthesis control environments

Synthesis programs often require a sophisticated environment to compute and prepare their time evolving control parameters (reference Jean-Baptiste Barrière and Xavier Chabot: "Integration of Aid to composition and performance environments", Proceedings of the ICMC 1993, Tokyo). Such an environment must be capable of displaying and editing parameters; it must allow establishing inter-dependancy rules between control parameters, for exemple when using the CHANT synthesizer (Barrière, Laurson, Iovino: "A new CHANT synthesizer in C and its control environment in PatchWork", Proceedings of the ICMC 1991) and it must give access to the features of a computer assisted composition (CAO) environment to compute synthesis control parameters and their time evolution in relation with the control of other musical parameters. PatchWork aims to be such an environment. The PatchWork set of library packages, that can be obtained through the IRCAM Forum, includes, among other things, "SpData", sound analysis utilities (ref. Pottier), "csound-edit-sco", an interface with the synthesis program Csound from MIT, (ref Vercoe) "CHANT" (ref Iovino), a control interface for the CHANT synthesizer (ref. Rodet), and "PW-Max , an interface for the Max program (ref Puckette, Zicarelli) which is described in this document.



1.2 PW-Max general principles

The general principle is as follows: a PatchWork patch computes data and writes it as arguments for one of the Max data structure object (*qlist, explode, table, message box,* or a complex abstraction) which is itself embedded in a predefined generic Max patch for that object.



Such a connection between the two programs is a so-called "loose" connection since it uses a text file and even a network between the Macintosh and the NexT computer in the case of the PatchWork-Max/ISPW connection. Nevertheless, the PW-Max library demonstrates the need of a link between PatchWork, including dedicated editors, Lisp functions, various computation models, AI engines, etc. , and realtime synthesis engines.

A prototype of what should be a link between an CAC environment (represented by the language Scheme) and a realtime environment (Max on the ISPW) has been developped (Gerhard Eckel and Francisco Iovino, IRCAM internal seminar in April 1994).

The PW-Max library is most often used in conjunction with other PatchWork libraries, such as the "SpData" library. Such a combination allows one to work with various analysis-resynthesis models, with the the ability to process analysis data before resynthesis with the IRCAM workstation (ISPW). Examples of applications will be given in the tutorial section of this manual.

1.3 Starting up the PW-Max library

1.3.1 loading PW-Max

Like all PatchWork libraries, the PW-Max library folder must be in the "User-Library"; loading is performed through the "load library..." item of the PWoper menu by choosing the "pw-max.lib" file in the "PW-Max folder.

1.3.2 files and directories

- *README* explains the version history and the last correction and updates.
- *pw-max.lib* : loader for the pw-max library: it defines the pw-max directory structure and loads the code.
- *pw-max.-src:* includes lisp sources and *load-pw-max.lisp* the code loader.
- *pw-patches:* divided into:
  - *on-line-help*: help files called with the command "t" after selecting a module.
  - *tutorials*: extended exemples of use.
  - *MaxProgs*: used by the demos patches to write Max abstractions
- *pw-max.-abs*: contains the PatchWork abstraction files which are read and set in the userlib menu.

1.3.3 dependencies - PW-Utils library

By default the PW-Max library loads the "utils" PatchWork library which includes utilities for filenames and path processing. Please make sure that the PW-Utils library is present in your library folder.

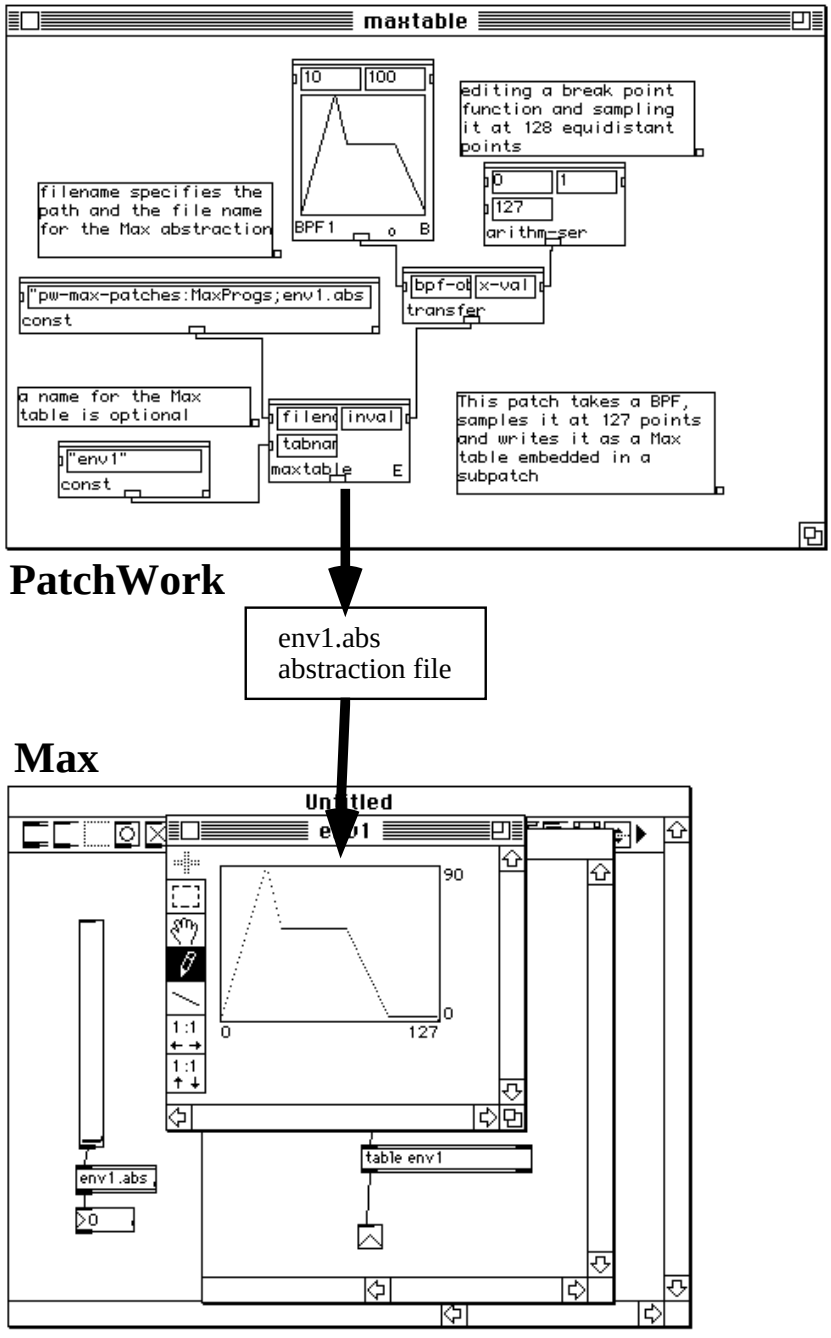
1.4 PW-Max: correspondence with Max

There are **PW-Max** modules corresponding to various Max objects:

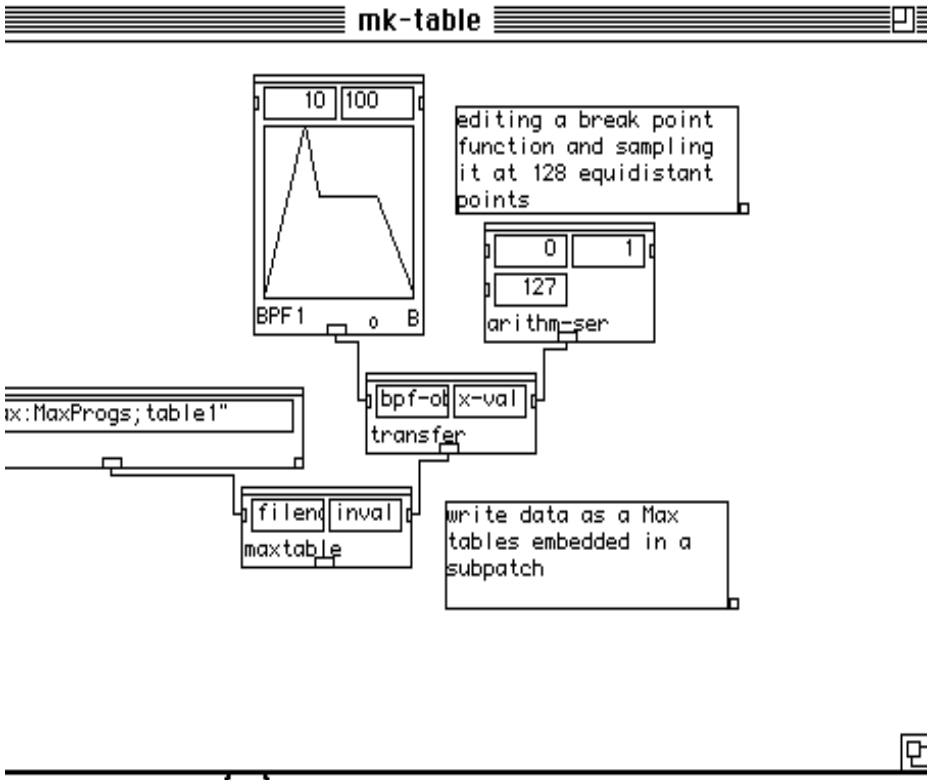
- The *maxtable* module produces data inside a Max *table* object.
- The *sndtable* produce wave tables for the *osc1~* and *tab1~* Max SIM objects.
- The *msgbox* module writes data and commands inside a Max *message box* .
- The *explode, qlist-obj,* and *qlist-par* modules writes sequence data (for instance produced by the PatchWork modules *RTM* or *chordseq*) in explode and qlist max modules
- The *maxbank* and *maxbank*2 modules create banks (i.e. an abstraction with chained modules cooperating for one task such as filtering) for Max.

1.4.1 Max tables

Max tables are written by the PatchWork "maxtable" module as a Max table object embedded in a Max sub-patch with an inlet for x-values triggering the ouput of the corresponding y-value trough the outlet.



The table data is input in the "maxtable" module as a list of integer numbers for y-values. The size of the table is the size of the list. However graphics may not correspond to the table data; one should not try to edit the table with the the Max table editor. The following PatchWork patch produces the above Max abstraction:



1.4.2 Producing tables and curves in PatchWork

The list representing the data could entered:

- by typing the list in a "const" module; one y-value must be entered for each x-value between 0 and size-1. This is useful for exemple to enter a table which has been found empirically.
- by entering points in the BPF (break point function) editor and sampling it at "size" points between 0 and size-1. This is useful to enter simple enveloppes shapes with linear segments (straight lines). The BPF module needs a "tranfer" module to read y-values and a "arithm-ser" module to produce x-values for the "transfer" module, or by sampling a curve computed by a PatchWork module.
- by entering x-y pairs in a "lagrange" module. This module tries to find a formula (a polynomial) for a smooth curve coming as close as possible to the specified points. The curve is then sampled with the "sample-fun" module in the same way than the BPF module. Note that the precision depends on the point sequence (thus if a x-y pair is entered in the lagrange module, the y-value will

not necessarily come out as an answer for the x-value; an approximation takes place). Shapes with angles have no solution and trigger an error.

• by creating a curve with a mathematical formula using the "make-fun-num" module and sampling it as above. Refer to the PatchWork reference manual, to the tutorial, and the one-line documentation for the syntax.

• by creating "series" of numbers and sampling them in lists. Series are abstract sequence of objects, possibly of infinite size. They can be viewed as "domains" that limit the values that an entity can take. For more information, refer to the documentation on the PatchWork library "series" by Camilo Rueda. This library is part of the PatchWork user-lib package and must be loaded in order to be used.

1.4.3 Max data

Max data is usually transmitted by messages consisting of a name followed by parameters. The exact syntax is defined by the target entity, max module or subpatch which has a "receive" max object for that name. A Max message can be sent in four ways: directly through a connection between two modules, entering the parameter value list in a "send" max object with that message name, a message in a message box, or a message in a "qlist" Max object.

Global and local; a message is normally global among all Max subpatches but the Max IRCAM version on the Musical Workstation has the following feature: a message is local to a processor unless present in a "send" module;

The PW-Max library has modules to write data as messages in message boxes and in qlists. Note that a qlist is more efficient for execution, but message boxes are easier to edit and their execution is sequential while qlists are executed asynchroneously and are more difficult to synchronize with everything else. Messages boxes and qlists are instantiated inside Max subpatches with a variable number of inlets and outlets.

Several data structures are implemented:

• a simple list of messages with parameters that will be stored in either a single message box (module "msg") or a qlist (module "qlist").

• Models of resonance (reference J.-B. Barrière, etc..) data. Messages for frequency, amplitude, and bandwidth for a number of filters specified in advance are stored in either qlist (module "modres-to-max2") or message boxes (module "modres-to-max"). The current version uses global message for efficiency and it expects the filter bank element "n" to have receive objects for "freqn" "ampn" and "bwn". See the section on Max banks bellow. A later version might use a unique message with an index selector for each bank.

• additive synthesis analysis data consist of time sequences of "frames". Each time frame is made of a set of partials: frequency, amplitude and phase. Frames are written in qlist with time delays.

• Pitch (more generally note) sequences as created by the "chordseq" and "RTM" objects; see bllow the section "max sequences".

Limitations; there are two limitations: first the processor ability to actually handle sending a bulk of messages, each of them triggering in turn other computations, and second the limit on the total size of a message

bulk; the computation limitation exist for both message bozes and qlists, but the message bulk maximum size is a restriction only for message bozes since they send everything at once.

1.4.4 Max sequences

Producing max sequences is a very important application of the PW-Max library. A sequence could be computed by PatchWork or produced by a sequencer, and written as a midifile (refer to the PatchWork library "midi-file" by Laurent Pottier). However a midifile is restricted to MIDI syntax. Typically a rhythm would be written as a sequence of delays and MIDI key events. In that case, only two parameters are allowed (key and velocity values) and they must be integers less that 128. The Max object "seq" is able to read a midifile sequence but it does not exist yet in the current IRCAM Max version 0.24 for the ISPW.

A more flexible way is to use the Max module "qlist". Qlist acts as a sequencer if connected to a simple delay mechanism. The reader is referred to the qlist documentation (reference Cort Lippe).

The qlist object needs a front-end, such as the "nqlist-del" abstraction in the SIMlib library (reference X.Chabot), to control sequencing.

In the case of nqlist-del, the basic syntax of a qlist line should be:

- to send a message right away:
<*message_name >p1 p2 p3...;*
- the message will be sent after the specified delay:
<delay> <message_name >p1 p2 p3...;
- to stop execution until the the qlist object receives a "next" message:
<*negative delay > <message_name >p1 p2 p3...;*

The module qlist-obj reads data out from the PatchWork modules "chordseq" or "RTM" while qlist-par takes a list of delays to schedule messages with their parameters. Both modules have a "message name" input to specify messages in qlist. If the list is too short (it normally contain only one message) the last message name will be repeated. The two modules also have optional inputs for any number of parameters that can be integer, float or string. In fact these modules are able to schedule anything and not only sonic events similar to notes.

1.4.5 Creating banks for Max

The "maxbank" is able to produce banks of Max modules with the following characteristics:

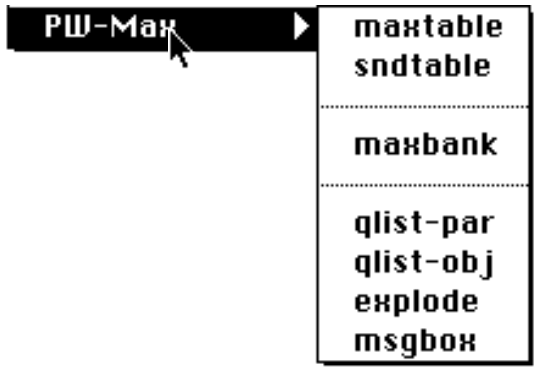
- all modules are identical
- they are in parallel, that is with a common signma input
- their outputs are summed to a common output
- they are inputs for global controls such as scaling etc...
- the modules are indexed (with # abstraction parameters) and indices are used to name local controls (i.e. to select partial data).

Current realizations feature banks of fixed size for:

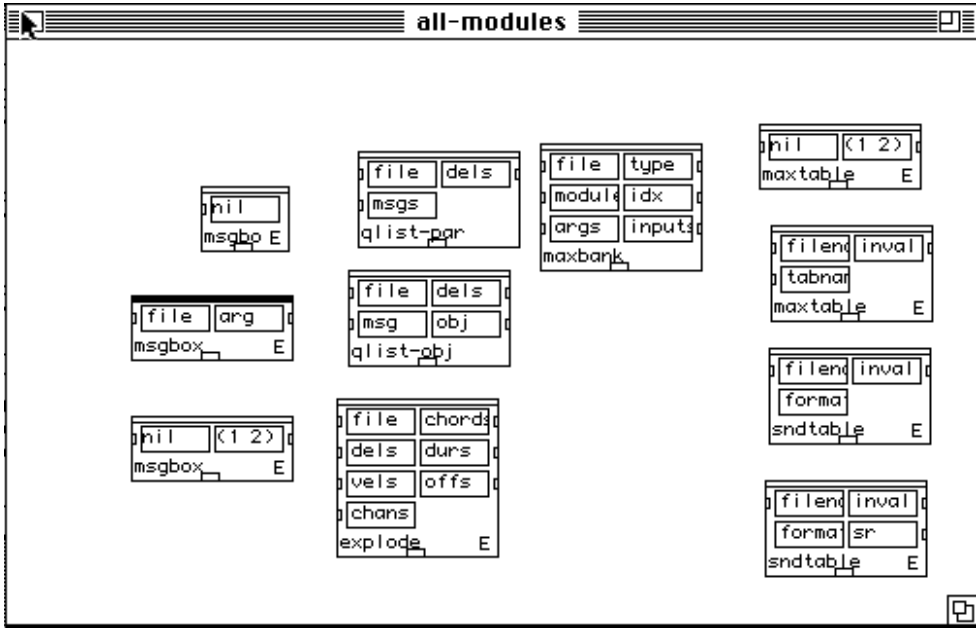
- a 2nd order band-pass filter bank for model of resonance and spectral shape construction.
- additive synthesis oscillator bank with linear interpolation.
- bank of "FFT-peek" modules able to peek at specified frequencies in a FFT tabel.

2 PW-Max objects reference manual

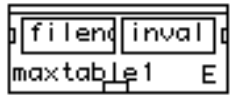
The PW-Max menu found in the Userlib menu should look like that:



The PW-max modules are is the figure below:



maxtable *create a max table*



Syntax

```
pw-max:maxtable filename inval  &optional tabname
```

Inputs

filename string

inval fixs (list of integers)

tabname string

Output

```
string (the file name)
```

Description

```
[function]
```

The maxtable module takes a list of values and creates a Max table module.

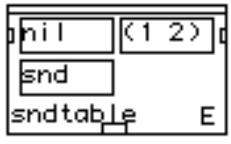
The table is embedded in a simple abstraction patch with one inlet and one outlet. The input values are scanned in order to produce the right table parameters:

size is the number of input values, range is the maximum value, and a flag is set if they are negative values. See the <inspect table> dialog for each version of Max.The filename input names the Max abstraction file.

The inval input must a list of integers (floats will be truncated).

The optional tabnale input allows naming the Max table for global access in a Max patch.

sndtable *writes values in a soundfile*



Syntax

```
pw-max:sndtable filename inval format  &optional sr
```

Inputs

filename string

inval list of integers

format menu: snd, aiff, sd2, carl

sr sampling rate

Output

```
maxtable object
```

Description

sndtable takes a list of values and writes a file in the chosen soundfile format. This module is intended to create wave tables for oscillator, whave shaping, windowing for fourrier transforms, etc...

It is not intended for sound synthesis, although there is no limitation on the table size, but it should be 512 for the current implementation of oscillators (with the <osc1~> module in MAX-ISPW.

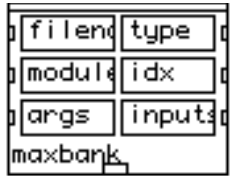
<filename> is a string which is the name of the soundfile.

<inval> must be a list of integer (-32768 <= i <= 32767) values. In this version, there is no check on the values, but an error is triggered by the write function if one value is out of bounds.

Use the module <g-scaling-max> to scale the values. The table size is simply the length of the value list. <format> let you chose the soundfile format; however, the current version implements only the snd (next) soundfile format.

<SR> the optional SR input let you specify the sampling rate which must be set in the sound file header for the application programs to handle the data properly.

maxbank *creates banks for Max*



Syntax

```
pw-max:maxbank filename type module idx args inputs
```

Inputs

filename string: the name for the Max abstraction

type string: the type of bank

module string: the name of the module used in the bank

idx list of integers: the list of module indexes

args list: arguments for the module

inputs list: input specification

Output

```
maxbank object
```

Description

makes a bank and writes to file as a Max abstraction. Depending on the keywords specified in <inputs> (explained below), one of two predefined Max patch is used. See the paper documentation or open with one of the examples in <pw-max-patches:Maxpatch:> to understand the exact outlook of these two types of banks. Each of the two types of banks is made of one module repeated a number of times as specified with the <index> input and all copies being chained together. The module used in the bank

must comply with the following restrictions:

The number of outputs must be equal to the inputs. Each module's output is connected to the corresponding input of the next module. The last module's left output is connected to the only abstraction's outlet.

The number and the type of the bank abstraction's inlets are specified by the <inputs> inlet of the module, see below.

<filename> is the name of the Max abstraction which will be created.

<type> is a string used in the comment printed in the max abstraction file, denotating the type of bank i.e. oscillator bank, filter bank, etc...

<module> is a string naming the Max module on which the bank is contructed.

<idx> is a list if integers used to index the modules. It is usually sequential, i.e. (1 2 3 .. 25).

For each module in the bank the index is the first parameter writen #1 in the abstraction.The exact use of the index is defined in the abstraction itself (for example it can be used in <route>,

<sel> modules, or simply to construct global names such as 1-amp, 1-freq).

<args>: specifies the arguments for the banks modules. If nil, the only argument is the index.

If non nil, it must be a list of items; each module in the bank will have the same arguments in order. Giving $1 instead of an item, allows specifying parameters when calling the bank astraction.

<input> specifies both the number of inputs for the bank's modules and the way the first module should be connected to the bank abstraction's inlets.

<none> means no inlet for that input .

<sig> means that a signal inlet is to be connected to the corresponding input of the first module

<ctl> means that a control inlet is to be connected to the corresponding input of the first module

<route> has a special meaning; if it is not present in the <inputs> list, a simple template of Max abstraction is used where module are simply chained, one to the next. If it is present, a second template is used instead, where the inlet denotated by <route> is connected to a <route> (several if idx > 10) filled with the values of the <idx> input and with each ouptut for the route module connected to the corresponding input of the corresponding module.

For example (none sig ctl) means that the modules are expected to have a total of three inputs (and outputs) but the bank abstraction has only two inlets: a signal inlet to be connected to the 2nd module's input and a control inlet to be connected to the 3rd input

3 PW-Max tutorial patches

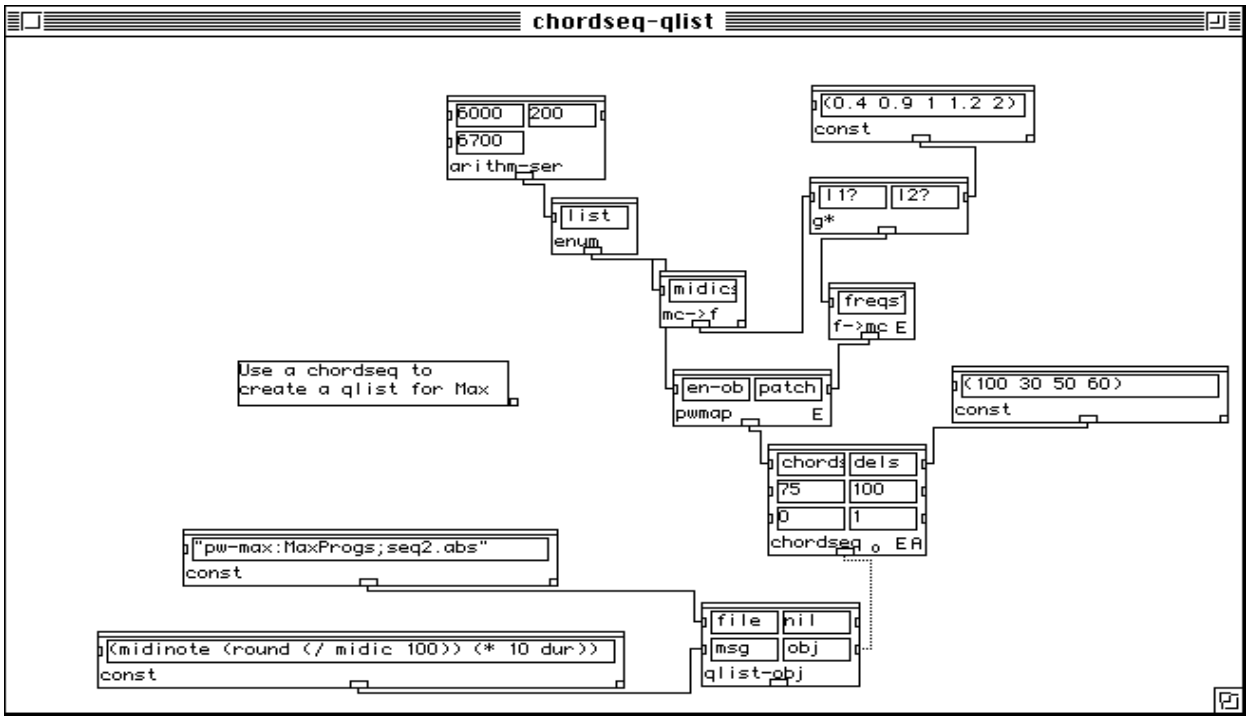


Figure 1 Use of a chordseq module to write a qlist for Max.

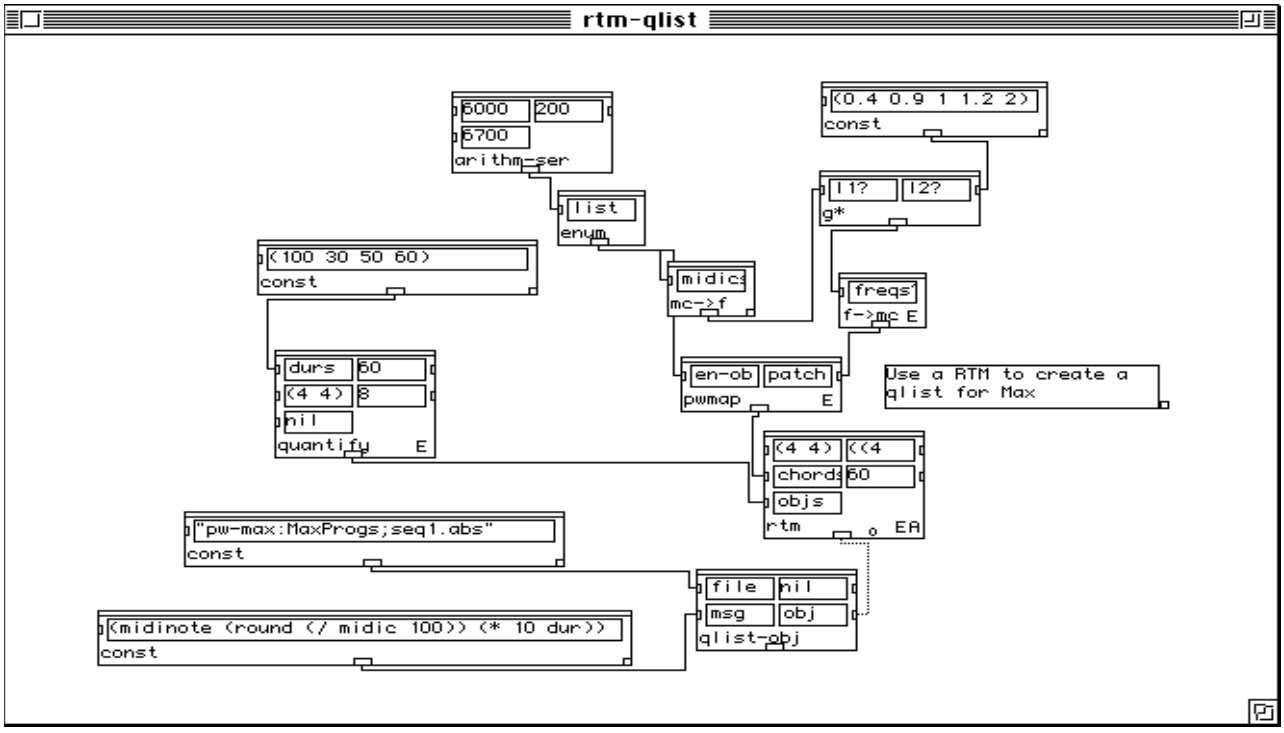


Figure 2 — Use of a RTM module to write a qlist for Max.

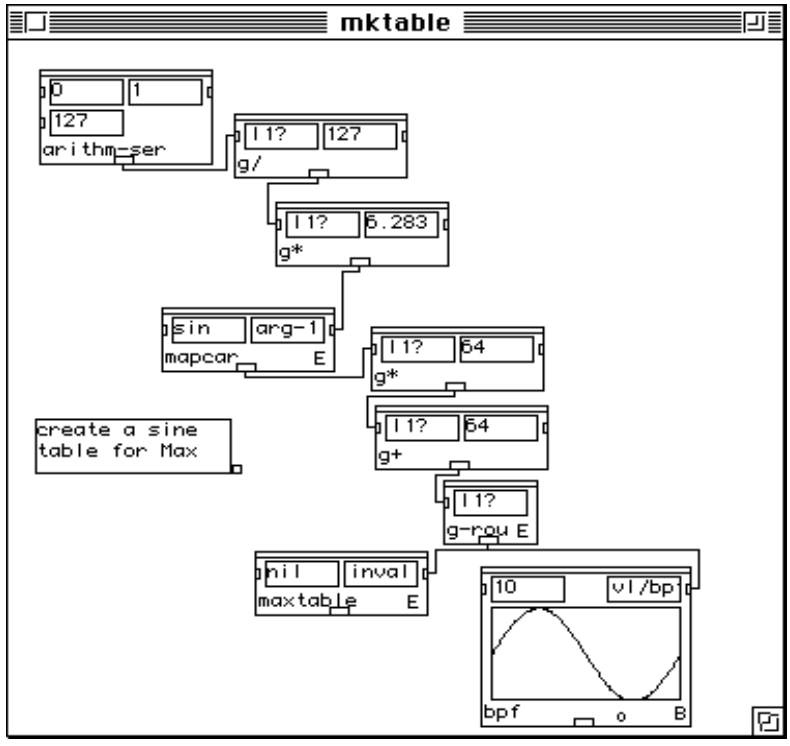


Figure 3 The making of a Max table.

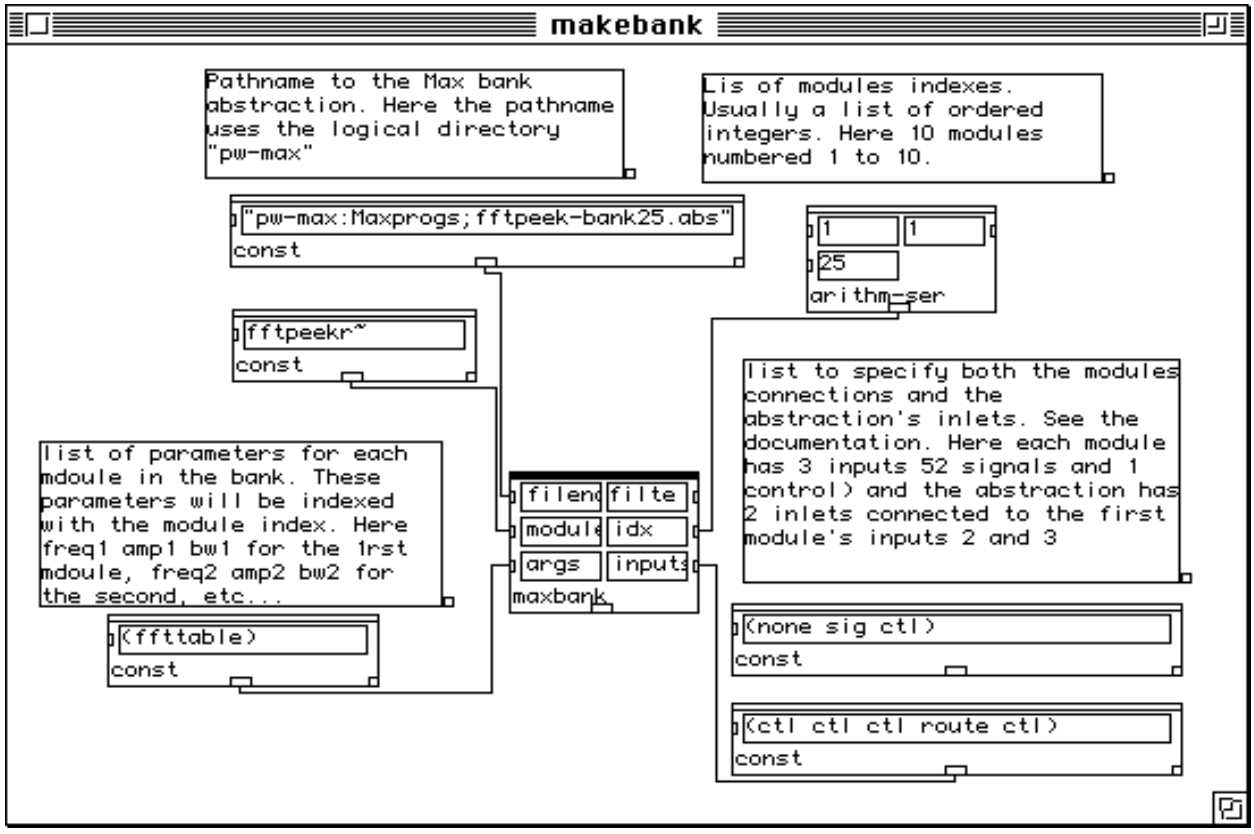


Figure 4 — The making of a databank to control 25 SIM filters.

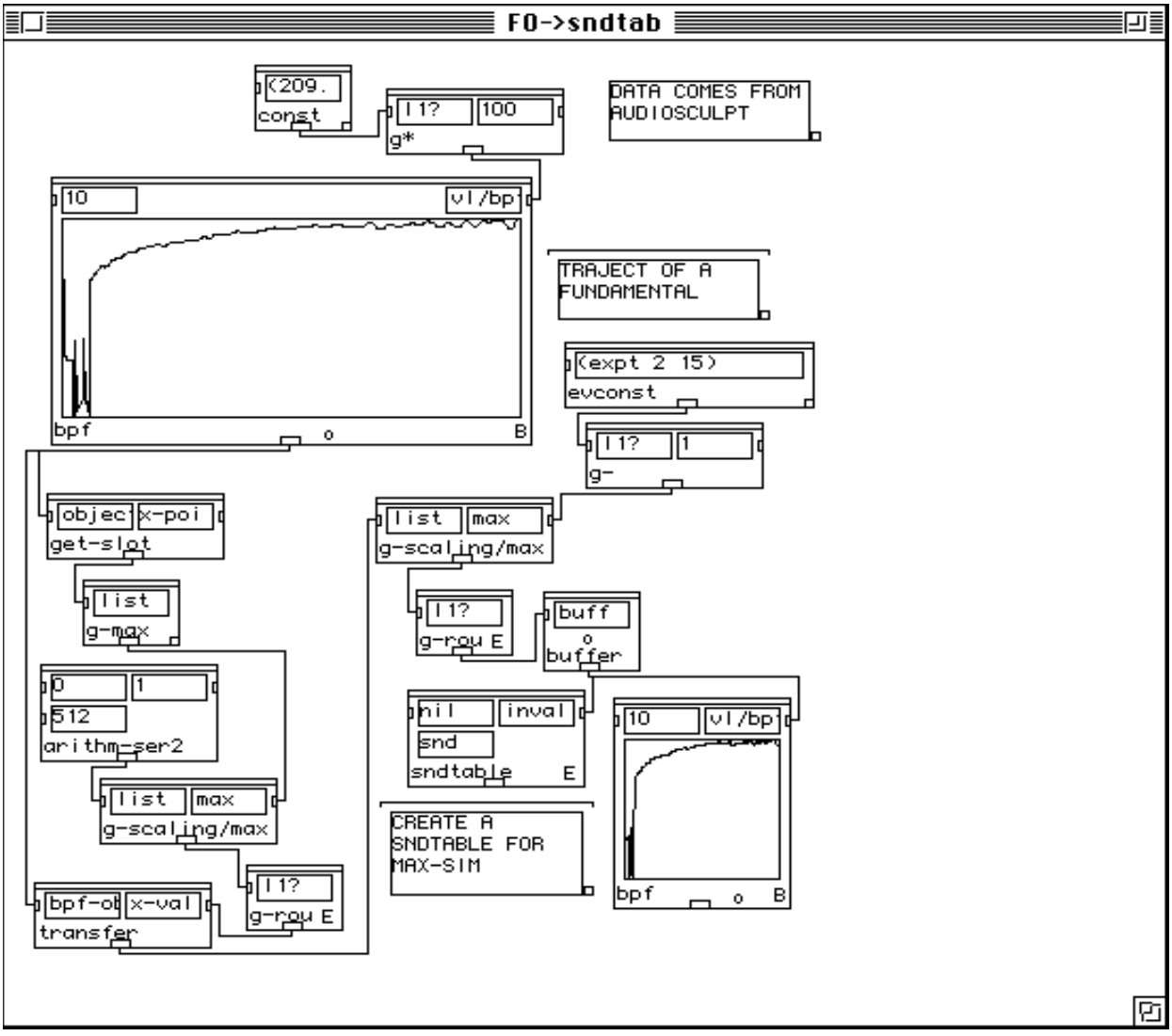


Figure 5

The making of a "SND" soundfile of 512 samples, to be read by Max/SIM as a SND table.

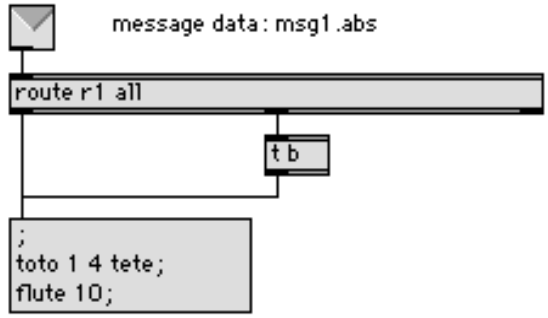


Figure 6 Sending message with parameters

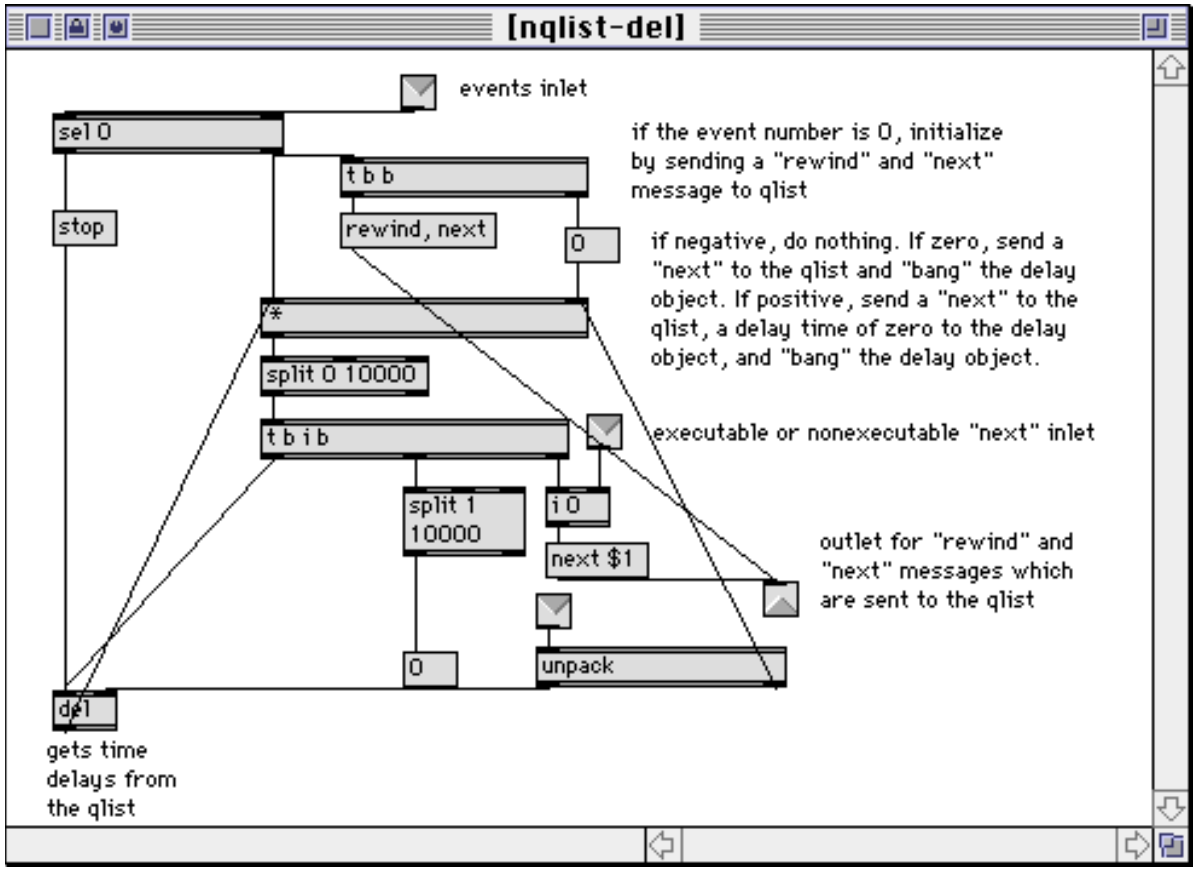


Figure 7 Abstraction "nqlist-del"

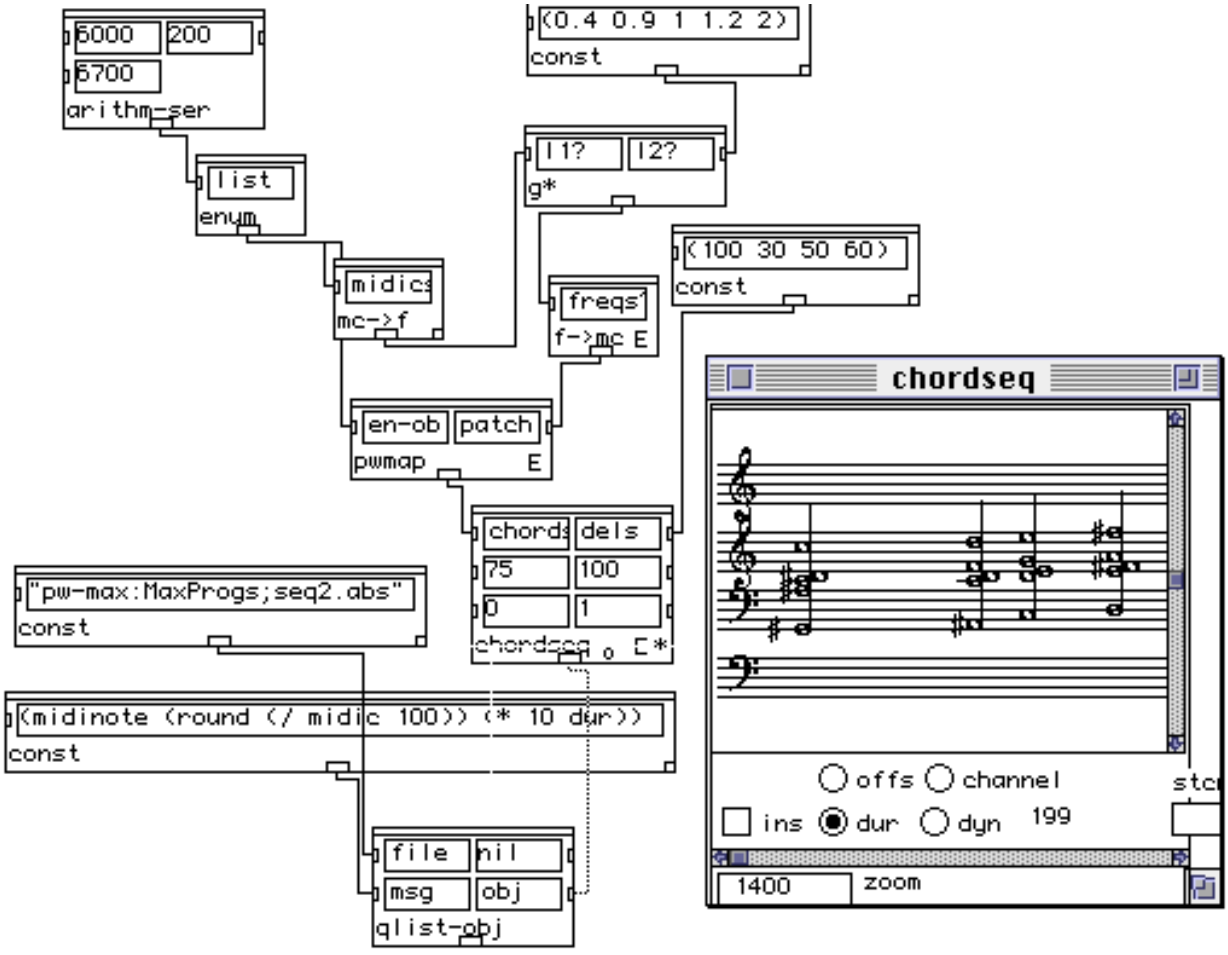


Figure 8 (a) A sequence in a chordseq

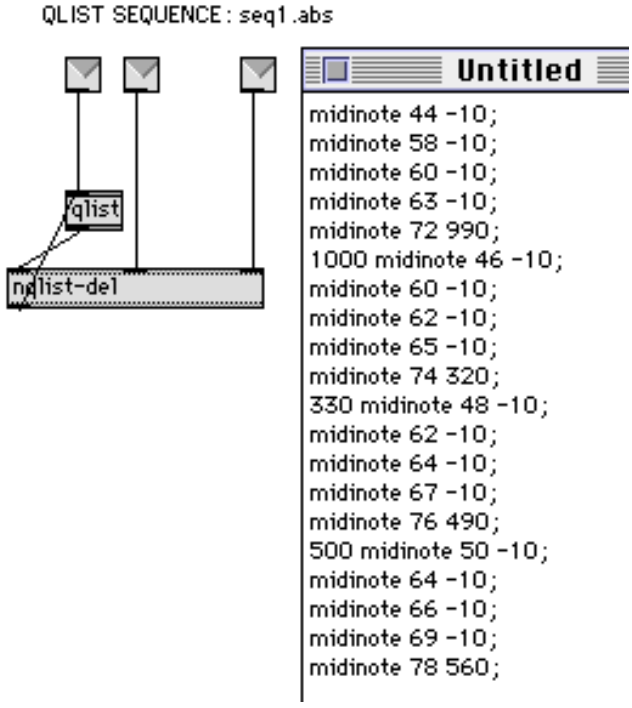


Figure 9 (b) The same sequence in a qlist

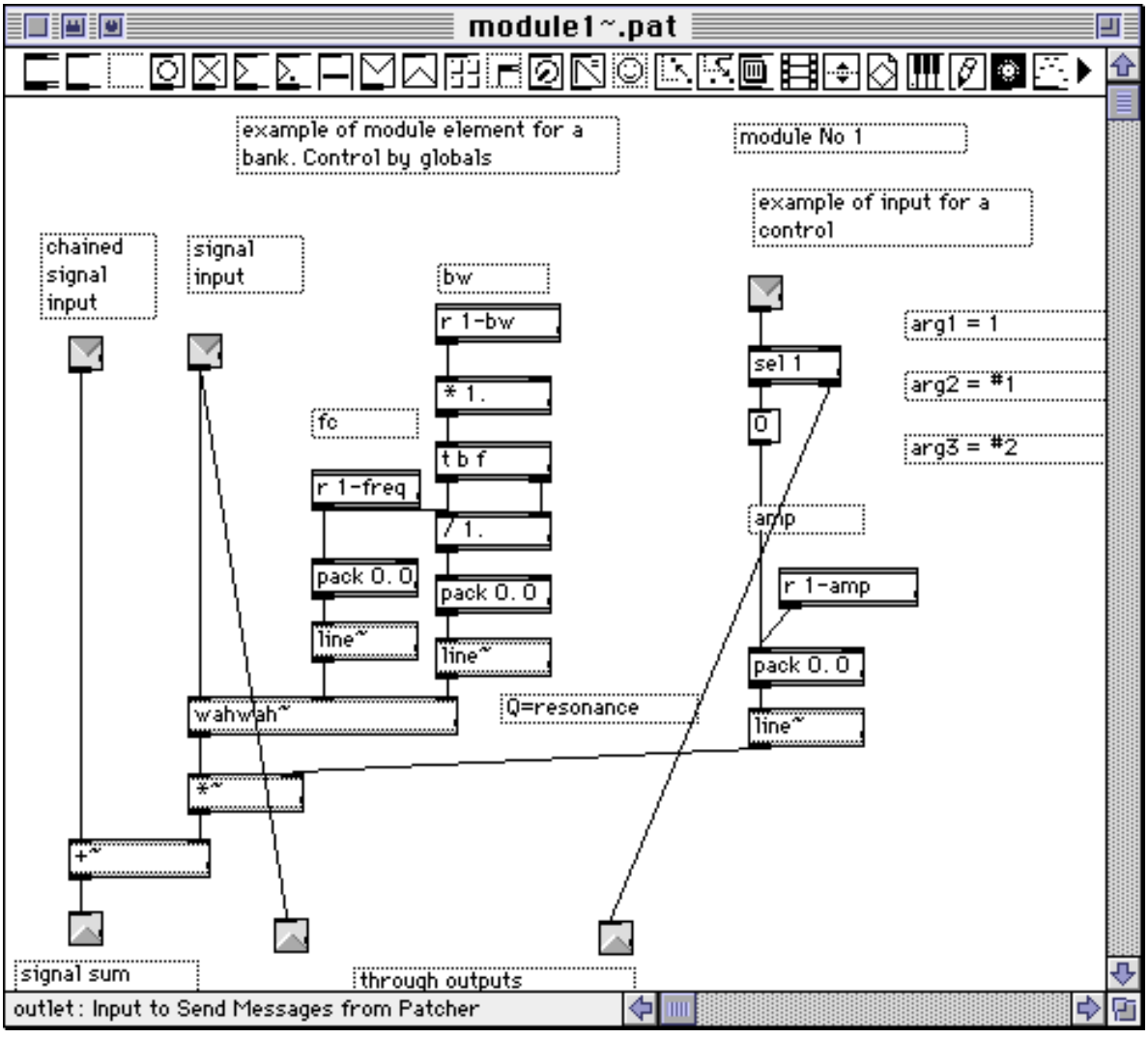


Figure 10 — The module in Max with name "module1~.pat" with three arguments

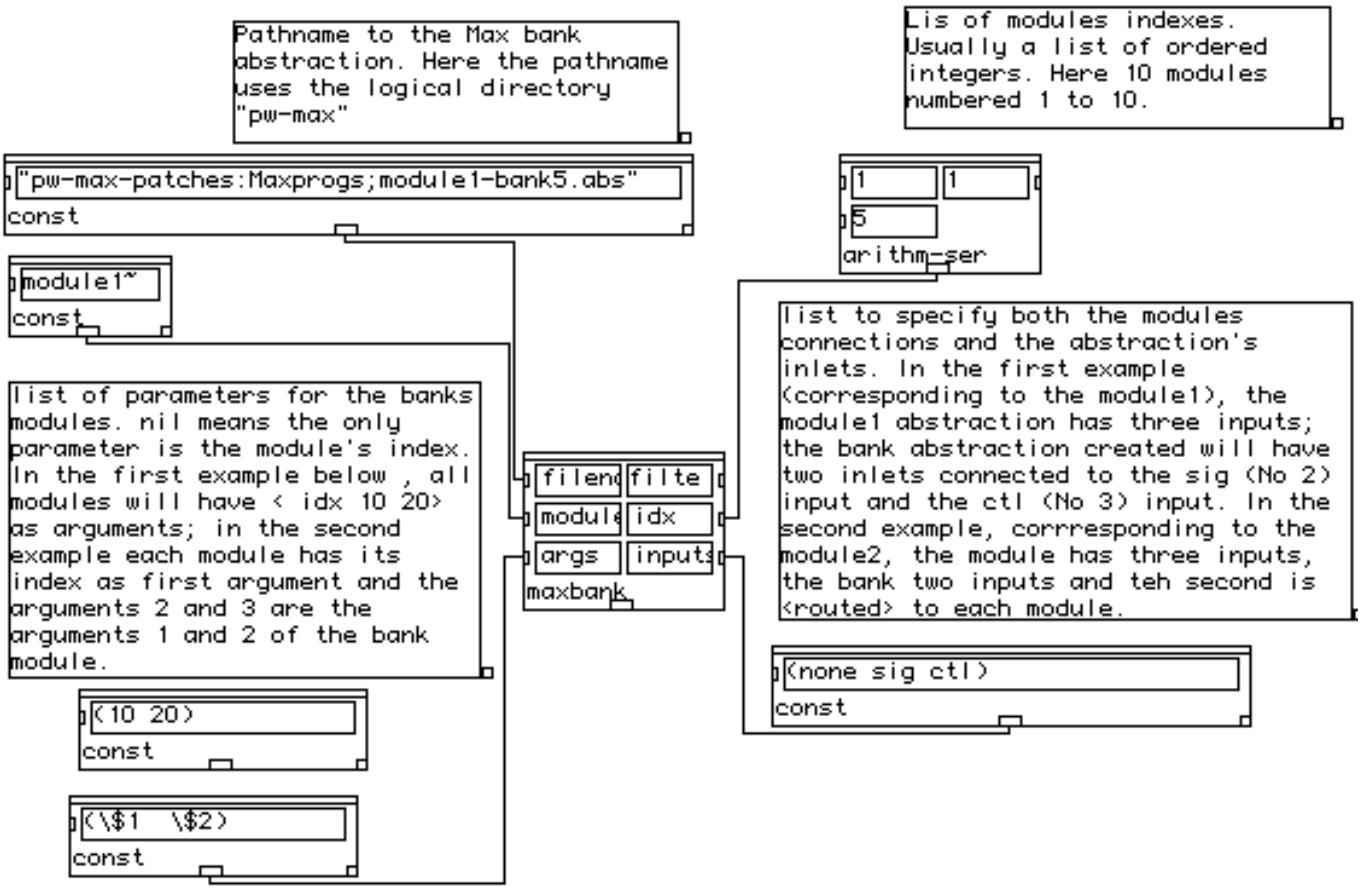


Figure 11 (a) Patch for assemblage of a bank of modules in parallel

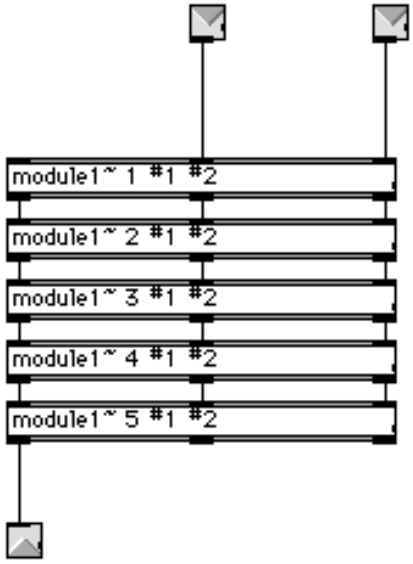


Figure 12 (b) Assemblage of a bank of modules in parallel in Max

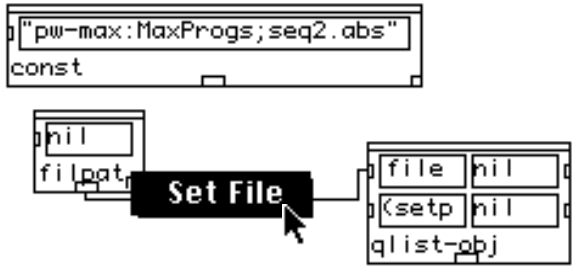


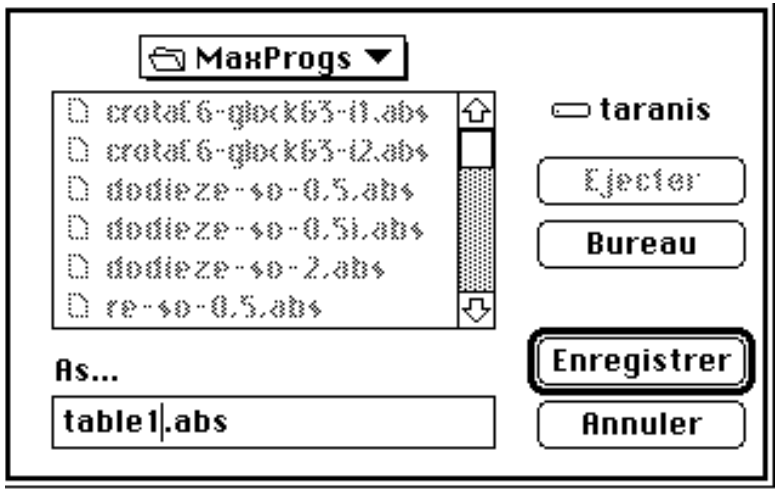
Figure 13 A module with file input (module "const" or "filpat")

4 Appendix: Max patches encoding

4.1 File names inputs

For all modules, the first input is the name for the Max file to be written; it has the behavior described below. The file name input allows the user to enter a file name either interactively through a standard Macintosh dialog or by typing and storing a path string in a "const" PatchWork module. When a module has a filename input, it responds in the following way:

• if the input is nil (nothing connected), the evaluation of the module triggers a standard Macintosh dialog to appear, allowing one to specify a file name. If the "cancel" button is depressed, the evaluation of the module is also cancelled. The file name and the current directory are memorized and the next time the module is evaluated, the dialog will open again on the previous directory and with the previous file name already written in the name box.



• if a PatchWork "const" module is connected, it is expected to contain a path string with the Macintosh Common Lisp syntax (beware of ";" and ":", refer to Macintosh Common Lisp documentation): "logical directory:directory;....;file name" or "harddisk name;absolute path;....;filename". Below is the list of the most commonly used logical directories.

• root	current boot disk
•pw-lib	the *user-lib* folder in which are all currently distributed PatchWork libraries.
• pw-max	the *PW-Max* library folder.

- spdata — the SpData library folder which is defined if the SpData library is loaded; this is normally the case.

Let us give a few examples:

- a file named "datafile" placed in a folder "myfolder" at the top level of the boot disk can be accessed through the path: "root:myfolder;datafile"
- the same file in the folder "myfolder" at the top level of an external disk named "mybackup" is accessed with the path: "mybackup;myfolder;datafile"
- a Max abstraction "myabstarction.abs" in the "MaxProgs" folder of the PW-Max library is accessed by: "pw-max:MaxProgs;myabstraction.abs"
- a model of resonance data file in the modres-data folder in the CHANT library folder is accessed in the following way; note than the logical directory "chant" is normally not defined, unless the CHANT library is loaded; it must be accessed through the "pw-lib" logical directory: "pw-lib:chant;data-folder;modres;cymbalum.ll"

4.2 Max Patches Encoding

When the button "write as text" is selected, Max patches are saved to files as text command lines for the embedded Max interpreter.

4.2.1 command lines

The most common command line is in the form

```
#P <module name> <four numbers for place and size> <data for this module .....> ;
```

For example a simple comment box with text might be:

```
#P comment 40 20 300 4  this is a comment;
```

4.2.2 patches

A patch (or subpatch window is instantiated by the commands:

```
max v2;
#N vpatcher 200 80 950 730 -1;
...patch text .....
#P pop;
```

The arguments for vpatcher are graphic parameters (place and size) and CPU number for the SIM. -1 means no CPU. On the Macintosh Opcode Max the last parameter will trigger the warning "newex extra parameter" which does not matter. However it is not possible to produce patches for various SIM CPUs from the Macintosh.

4.2.3 connection

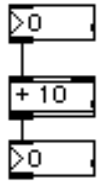
Connections in Max patches are done with the command

```
#P connect <source module number > <source outlet> <taget module number> <target inlet>;
```

Modules (i.e. instantiated by #P) are numbered starting from 0 being the last #P command which appears in the script and n-1 beeing the first (nth module) which appears in the script. Do not forget to count comment boxes even if they are never connected to anything. Thus the following text:

```
max v2;
#N vpatcher 53 127 453 427;
#P newex 126 110 35 196617 + 10;
#P number 126 79 35 9 0 0 0 3;
#P number 129 142 35 9 0 0 0 3;
#P connect 2 0 0 0;
#P connect 1 0 2 0;
#P pop;
```

will produce the following patch



0 is the bottom number box, 1 is the upper number box, and 2 is the + module. Thus the outlet 0 (the only one) of the upper number box is connected to the inlet 0 (the left one) of the + module and the outlet 0 of the + moduleis connected to the inlet 0 of the bottom number box.

Note that the order in the script does not correspond necessarily to the up/down order in the graphic patch.

Index

U

V

Z