Ircam
documentation

- Etudes et recherches
- Œuvres musicales
- Logiciels

PatchWork

Esquisse

First English Edition, February 1996

IRCAM Centre Georges Pompidou

This manual was written by Joshua Fineberg, the introduction was written by the author and Tristan Murail, in collaboration with Curtis Roads, under the editorial responsibility of Marc Battier - Marketing Office, Ircam.

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

The Esquisse library conceived and programmed by Jacques Duthen, Tristan Murail, Camilo Rueda.

2nd edition of the documentation, February 1994.
This documentation corresponds to version 2.0 or higher of PatchWork, and version 1.0 of the Equisse library.

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

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. (1) 44 78 49 62
Fax (1) 42 77 29 47
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

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

Le librairie Esquisse pour PatchWork est constituée de modules répondant à des besoins spécifiques de la composition assistée par ordinateur. Comme toutes les librairies PatchWork, les modules sont accesibles dans un menu spécifique de la fenêtre PW.

Le lecteur pourra consulter les exemples réalisés par plusieurs compositeurs, et tirés de leurs œuvres, électroniques ou instrumentales.Ils sont rassemblés dans le dossier `Documentation>Sample Patches>Esquisse` du dossier PatchWork.

Les modules de la librairie Esquisse peuvent être classés en deux catégories : Interval et Freq. harmony.

Interval propose des fonctions pour la manipulation d'intervalles de hauteur.

Freq. harmony propose des fonctions axées sur la la manipulation de fréquences, dans la perscpetive de lier timbre et harmonie. En ce sens, ces fonctions sont à rapprocher du mouvement musicale de ce qu'on appelle la « musique spectrale ».

Ce manuel présente et explique en détail chaque module de la librairie.

1. Introduction

Esquisse is a library of musical functions developed at IRCAM for specific needs of composers. It is different from modules within PatchWork itself in that the functions it contains are intrinsically musical, and often linked to specific techniques and aesthetics. The current Esquisse library is embedded in PatchWork; originally, Esquisse was conceived in 1988 by a team working within the frame of Musical Research at Ircam. Members of the team were : Pierre-François Baisnée, Jean-Baptiste Barrière, Marc-André Dalbavie, Magnus Lindberg, and Kaijia Saariaho; members from the scientific department were Jacques Duthen and Yves Potard. At this time, Esquisse had been written in Le_Lisp, with the help of. Esquisse was designed as a software layer built upon PreFORM, which was an object-oriented environment with a graphic interface built as an extension to Le_Lisp, and had been written by Lee Boyton at Ircam in 1986-87. At that time, PatchWork was an extra software layer built "above" PreFORM: it was witin PatchWork that the composer could define and use graphical patches for computer-aided composition. In 1990, Pierre-François Baisnée merged Esquisse with PatchWork in Common Lisp, which is an object-oriented flavor of the generic Lisp language. PreFORM was then abandoned and CLOS (Common Lisp Object System) was used instead. As of this writing, PatchWork is developed within the same environement, which is based on Macintosh Common Lisp.

This library should be seen as a work in progress. The functions provided are in no way meant to influence aesthetic choices; they are merely functions that have proven useful to various composers for the realization of both acoustic and electronic works. Examples are available in the Tutorials folder show many compositional aspects of these functions in specific musical contexts (either taken from pieces or invented for the purpose of demonstration).

The modules in Esquisse can be divided into two main parts: Intervals and Freq. Harmony.

Intervals contains functions for traditional types of intervallic manipulations of musical material. These functions allow serial, post-serial, and combinatorial operations to be performed quickly and simply, avoiding laborious manual reckoning. An operation such as producing a *chord multiplication* can be performed using a single module and taking less than a second. This ease and rapidity is valuable not so much to save time, but rather to allow a level of suppleness not otherwise possible; the material can be fine-tuned repeatedly without forcing the composer to spend hours recalculating chord multiplications after each refinement. Manipulations of these modules can be performed with notes, traditional intervals (expressed as a number of semitones), or in set-class notation; the units to be used can be selected by optional arguments explained in the documentation for each module.

Freq. Harmony contains functions linked to a compositional movement centered in France and generally refered to as Spectral Music. The composers in this movement derive their harmonies and timbres from calculations performed directly with frequencies (instead of notes or intervals). These frequencies can be either used directly, when applied to electronic music, or approximated to the nearest available note, when used with traditional instruments. Since these functions are not tempered, we recommend that they be carried out with an approximation smaller than the semitone (set the option under PWoper/Global Options/Approximation to 1/4 or 1/8 of a tone). Another application of these functions is to calculate precisely the results of electronic processes, for example Frequency Modulation. If you take a Yamaha DX7 or DX7II digital synthesizer and select a simple FM patch with one oscillator and one modulator in a certain ratio, you can use the module **fm-ratio** to

see the frequencies of the sidebands that will be generated by the synthesizer. Adjusting the *index* argument produces results equivalent to adjusting the amplitude parameter of the DX7's modulator. This structure once approximated into notes could also be used as an instrumental harmony.

The Utilities section contains function similar to those found elsewhere in PatchWork ; the means of control, however, are more intuitive, and more directly applicable to musical contexts.

The MIDI section is still in development, but will contain modules for controlling the manner in which musical material generated in PatchWork will be treated by MIDI synthesizers.

This *Reference Manual* presents the modules in Esquisse in as consistent a way as possible. Each module (where applicable) contains optional arguments that allow the user to determine the format of the output values and the units of input and output parameters. The text of the module-by-module documentation is available as on-line documentation. Input ands optional input arguments are listed by name followed by a description of the possible input types.

atom—number or symbol not in parentheses

list—one or more atoms in parentheses

structured list—a list of lists

menu—options available by scrolling the mouse over the input box

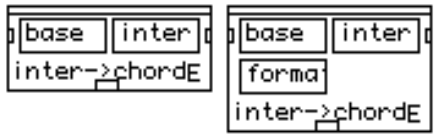
Finally, please note that the Esquisse library was written by Camilo Rueda, Jacques Duthen and Tristan Murail. It was revised by Tristan Murail and documented by Joshua Fineberg.

2. Intervals

Menu Intervals->Generation

inter->chord

Builds a chord from a list of intervals



Inputs

base (atom, list)

inter (list)

Optional inputs

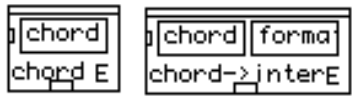
format (menu)

Constructs a chord starting with the note *base* and then adding the list of intervals *inter* (given in semitones; positive or negative) to that base. Microintervals can be represented as floating-point number parts of a semitone (e.g. a quartertone = 0.5). If *base* is a list, **inter->chord** constructs the same chord beginning on each successive note.

The optional argument *format* allows a choice of whether the box functions in real intervals (setting *inter*) or in *set notation* (setting *notes*). In set notation the first note is always 0; other notes are represented as a number of semitones above 0. For example, a major triad in interval notation is (4,7) and in set notation (0,4,7). Using *inter* returns the note *base* as well as the notes generated. *Notes* returns only the notes calculated, the base is included only when 0 is one of the requested intervals.

chord->inter

Converts a chord into a list of intervals



Inputs

chord (list)

Optional inputs

format (menu)

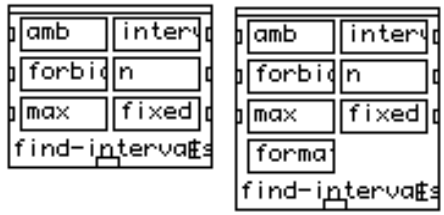
chord->inter returns a list containing the intervals (given in semitones; positive or negative) between the first note of *chords* and each successive note. Microintervals are represented as floating-point number parts of a semitone (e.g. a quarter-tone = 0.5).

If *chords* is a list of chords it will return a structured list containing the intervals of each successive chord.

The optional argument *format* allows a choice of whether the box returns real intervals (setting *inter*) or set notation (setting *notes*). *inter* returns only the intervals between notes. *notes* returns the intervals present as well as the first note, given as *0*. (See the documentation for the box **inter->chord** for further explanations concerning the format option)

find-intervals

Generates a list containing specified intervals



Inputs

amb	(atom)
intervals	(list)
forbid	(list)
n	(atom)
max	(atom)
fixed	(list)

Optional inputs

format	(menu)

find-intervals attempts to generate a list or lists containing the specified *intervals* (given in semitones; positive or negative) within the range *amb* (if negative intervals were used the range become between + and -*amb*). Microintervals may be represented as floating point number parts of a semitone (e.g. a quarter-tone = 0.5).

The number of intervals in each list found is determined by the argument *n*. If *n* is not at least twice the number of intervals in *intervals* no solution is possible. In addition to the requested *intervals* other intervals will be formed by this box. The argument *forbid* allows the exclusion of certain intervals from the resultant lists.

The argument *max* determines the maximum number of solutions returned (fewer may exist).

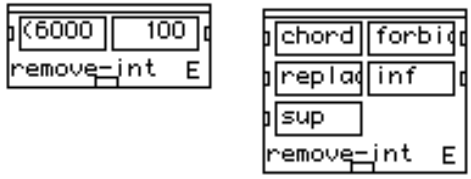
The argument *fixed* forces each list to contain that value or those values.

The optional argument *format* allows a choice of whether the box returns real intervals (setting *inter*) or set notation (setting *notes*). *Inter* will return only the intervals between notes, without the first note. *Notes* returns the intervals present as well as the first note, given as '0'.

Warning: a solution is not always possible; if none is found the value *nil* is returned.

remove-int

Removes or replaces intervals



Inputs

chord	(list, structured list)
forbid	(atom, list)

Optional inputs

replace	(atom, list)
inf	(atom)
sup	(atom)

remove-int removes the interval *forbid* (given in midicents) from the list *chord*. The upper note of the forbidden interval is deleted.

A list of chords can be entered in *chord,* in which case the interval is removed from each successive chord. A list of intervals for the argument *forbid* is also possible.

The optional argument *replace* takes the notes that would have been deleted and, instead, moves them to convert the forbidden interval into the replacement.

The optional arguments *inf* and *sup* allow the low, *inf*, and high, *sup*, limits in which changes can take place to be defined. These values are given in midicents, and refer only to the upper note in the interval pair.

Warnings: The lists returned by this box are ordered lowest note to highest note, regardless of the order in the entries. Entries are in true midicents, without any internal approximation, this may provoke certain problems which can be rectified through the use of the function **approx-m** prior to the entries.

transpoct

Changes the octave transposition of a chord

Inputs

chord	(list)
min	(atom)
max	(atom)

Optional inputs

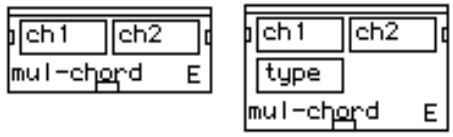
pivot	(atom)

transpoct transposes notes of a chord or list of chords *chord* by octaves such that all its notes will be contained within the range between *min* and *max*, given in midicents.

The optional argument *pivot* (a note, in midicents) forces all notes to be transposed so that they are within one octave of that note. *Pivot* must be within the specified range, or an error is produced.

mul-chord

Performs a chord multiplication



Inputs

ch1	(list)
ch2	(list)

Optional inputs

type	(menu)

mul-chord generates a list of chords in which the intervallic structure of *ch2* (a single chord in midicents) is reproduced beginning on each successive note of *ch1* (also a single chord in midicents).

The optional argument *type* allows the choice of whether the output is a list of chords (*seq*) or a single chord (*chord*) containing all the transpositions combined.

all-inversions

Lists all inversions of a chord

Inputs

chord (list)

direction (menu)

Optional inputs

format (menu)

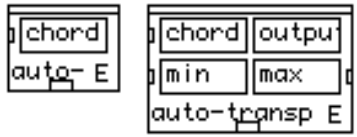
all-inversions generates a structured list of all possible inversions of *chord* (in midicents). Inversion here means the moving of the highest note down by octaves so as to make it the new bass note (when direction is '>'), or moving the lowest note up by octaves to make it the highest (when direction is '<'). The output is a list of chords containing the inversions, beginning on each successive note of *chord.* For example, the triad C-E-G would retun either E-G-C and G-C-E or G-C-E and E-G-C.

The optional argument *format* allows the choice of whether the original *chord* will be included, *inclu* or excluded, *exclu* from the output list.

Warning: *chord* can take only a single chord.

auto-transp

Lists transpositions of a chord



Inputs

chord (list)

Optional inputs

output (menu)

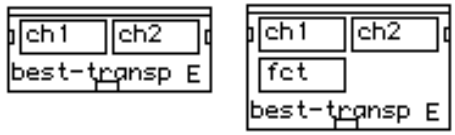
min (atom)

max (atom)

auto-transp outputs a list of all possible transpositions of *chord* (a single chord in midicents) which contain the first note of the chord. The optional argument *output* allows the choice of whether the original *chord* will be included, *inclu* or excluded, *exclu* from the output list. The optional arguments *max* and *min* (in midicents) will cause the notes to be transposed by octaves to fit within the specified range.

best-transp

Transposes one chord ***close to a second***



Inputs

ch1 (list)

ch2 (list)

Optional inputs

function (menu)

Transposes the chord *ch2* (a single chord in midicents) so that its intervallic distance to *ch1* (also a single chord in midicents) is as small as possible. Thus the distance between each note of *ch2* and each note of *ch1* becomes as small as possible. This is essentially the same as the box *best-inv* except the ordering of *ch2* is preserved. The optional argument *fct* allows the choice between two different algorithms for calculating this function, *sum* and *max*. The default is sum because *max* can produce quarter tones from semitone input. For best results one should experiment with both and chose according to context.

best-inv

Inverts and transposes one chord close to a second

Inputs

regch (list)

intch (list, struictured list)

Optional inputs

function (menu)

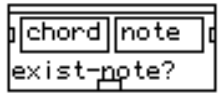
Extracts the intervallic content of the chord *intch* (in midicents) and through a global transposition of the chord followed by octave transpositions of individual notes produces a chord with the intervals of *intch* whose notes are as close as possible to those of the chord *regch* (also in midicents). This in essence distorts the chord *regch* by the smallest amount possible for it to take on the intervallic structure of *intch*. *regch* must be a single chord. *intch* may be a list of chords (also in midicents) from which the one that distorts least *regch* is used for the operation. The output will still be a single chord.

The optional argument *fct* allows the choice between two different algorithms for calculating this function, *sum* and *max*. The default is sum because *max* can produce quarter tones from semione input. For best results one should experiment with both and choose according to context.

Menu Intervals->Analysis

exist-note?

Finds whether a pitch is present within a chord



Inputs

chord (list, structured list)

note (atom, list)

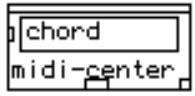
Outputs the first note in *chord* (a single chord in midicents), which is the same as *note* (regardless of octave). The value returned is the midicents value of that note at the octave it exists in the *chord*. If the note is not present the value *nil* is returned.

The argument *chord* may receive a list of chords; in which case the output is a list of lists containing the analyses of each successive chord.

The argument *note* may receive a list; in which case the output is a corresponding list of notes found and/or nils.

midi-center

Finds the central midi value of a chord



Inputs

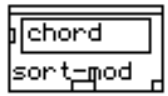
chord (list, structured list)

Calculates the value in midicents exactly halfway between the lowest and highest notes of the chord *chord*.

If *chord* is a list of chords, the output is a list of the central values.

sort-mod

Sorts a list of notes by interval



Inputs

chord (list, structured list)

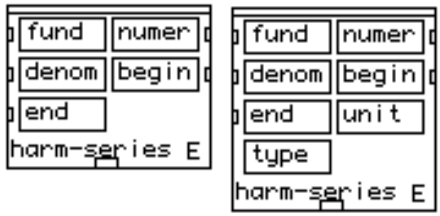
Returns a sorted list of lists in which each note of the chord *chord* (in midicents) is converted into a list containing the interval in midicents of that note from the C below it, followed by the midicents value of that C. A list of chords also may be entered for *chord* the resulting list will, however, have a higher level of structure and may not be acceptable as entry for some other boxes.

3. Freq Harmony

Menu Freq harmony->Harm Series

harm-series

Generates a harmonic series



Inputs

fund	(atom, list)
numer	(atom, list)
denom	(atom, list)
begin	(atom)
end	(atom)

Optional inputs

unit	(menu)
type	(menu)

Builds the harmonic and/or subharmonic series starting with the fundamental *fund*. The harmonic series is the series of positive integer multiples of the fundamental frequency. The subharmonic series is the series of positive integer divisions of the fundamental frequency.

The arguments *numer* and *denom* determine what sample (*numer*/ *denom*) of the partials is taken. (e.g. 1/1 = all; 1/2 = every other; 2/5 = the first two of each group of five)

The arguments *begin* and *end* determine the lowest and highest partials generated. The fundamental is represented by '1' or '-1' sub-harmonics are represented by negative numbers, overtones by positive. (e.g. partial number 7 is 7 times the fundamental frequency,partial -7 is the fundamental frequency divided by 7; thus to go from the seventh undertone to the seventh overtone *begin* would equal '-7' and *end* would equal '7')

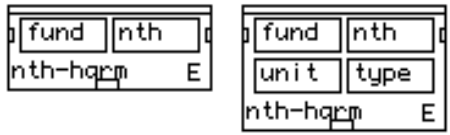
The optional argument *unit* determines whether the *fund* is entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the value will be converted to frequency inside the function and then the output is reconverted

to midicents. If *freq* is selected, the entry, calculation and output are all in Hertz.

When *fund* is a list, the optional argument *type* is used to determine the format of the output. The value *seq* returns a list of chords representing the partials requested for each successive fundamental. The value *chord* returns a single chord containing all the partials of all the fundamentals.

nth-harm

Generates a harmonic series



Inputs

fund	(atom, list)
nth	(atom, list)

Optional inputs

unit	(menu)
type	(menu)

Receives a fundamental *fund,* or list of fundamentals and returns the *nth* harmonic or sub-harmonic of each fundamental. The harmonic series is the series of positive integer multiples of the fundamental frequency. The sub-harmonic series is the series of positive integer divisions of the fundamental frequency. Partial numbers are determined by their relationship to the fundamental. (e.g. partial number 7 is 7 times the fundamental frequency,partial -7 is the fundamental frequency divided by 7)

If *nth* is a list, a corresponding series of partials will be returned for each *fund.* If *nth* contains non-integers the returned partials and/or sub-partials will be non-harmonic, and correspond to the fundamental frequency multiplied by *nth* (when *nth* is positive) or the fundamental frequency divided by the absolute value of nth (when *nth* is negative).

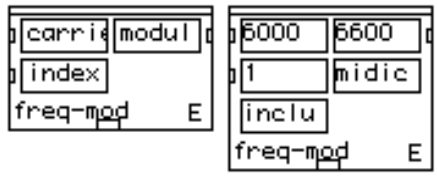
The optional argument *unit* determines whether the *fund* is entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected, the value will be converted to frequency inside the function. The output then is reconverted to midicents. If *freq* is selected the entry, calculation and output are all in Hertz.

When *fund* is a list, the optional argument *type* is used to determine the format of the output. The value *seq* returns a list of chords representing the partials requested for each successive fundamental. The value *chord* returns a single chord containing all the partials of all the fundamentals.

Menu Freq harmony->Modulations

freq-mod

Simulates frequency modulation



Inputs

carrier	(atom, list)
modul	(atom, list)
index	(atom, list)

Optional inputs

unit	(menu)
output	(menu)

Simulates the pitches generated by frequency modulation. The frequencies of the carrier *carrier* and the modulator *modul* are treated according to the following formula: carrier ± (i * mod.), i = 0,1,... *index*

If *carrier* is a list the output is a list of modulations around each successive carrier.

If *modul* is a list the carrier or carriers are each modulated by all the notes in *modul* as well as the partials of those notes up to the *index* specified.

If *index* is a list the formula is computed with 'i' equal to only the listed values. (e.g. *index* = (1 3), notes calculated are:

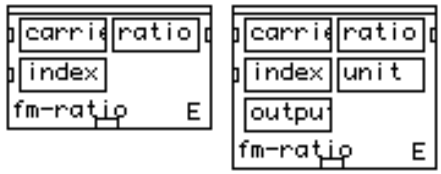
carr. + 1 * mod.; carr. - 1 * mod.;carr. + 3 * mod.; carr. - 3 * mod.)

The optional argument *unit* determines whether the <carrier> and <modul> are given in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the values will be converted to frequency inside the function and then the output is reconverted to midicents. If *freq* is selected the entry, calculation and output are all in Hertz.

The optional argument *output* determines whether the carriers are included (*inclu*) or excluded (*exclu*) from the output list or lists.

fm-ratio

Simulates frequency modulation



Inputs

carrier	(atom, list)
ratio	(atom, list)
index	(atom, list)

Optional inputs

unit	(menu)
output	(menu)

fm-ratio simulates the pitches generated by frequency modulation. The frequency of the *carrier* is modulated by a modulator whose frequency is equal to the frequency of the carrier multiplied by the *ratio*. Once the frequency of the modulator is determined, the calculations follow the same formula as the box **freq-mod**:

carrier ± (i * modulator), i = 0,1,...*index*

If *carrier* is a list the output is a list of modulations around each successive carrier.

If *ratio* is a list the carrier or carriers are each modulated by the frequencies that form all the requested ratios as well as the partials of those notes up to the *index* specified.

If *index* is a list the formula is computed with *i* equal to only the listed values. For *index* = (1 3), the notes calculated are:

carr. + 1 * modulator

carr. - 1 * modulator

carr. + 3 * modulator

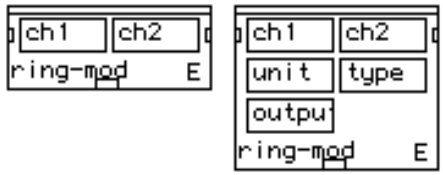
carr. - 3 * modulator

The optional argument *unit* determines whether the *carrier* is entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the value will be converted to frequency inside the function and then the output is reconverted to midicents. If *freq* is selected the entry, calculation and output are all in Hertz.

The optional argument *output* determines whether the carrier or carriers are included (*inclu*) or excluded (*exclu*) from the output list or lists.

ring-mod

Simulates ring modulation



Inputs

ch1	(atom, list, structured list)
ch2	(atom, list)

Optional inputs

unit	(menu)
type	(menu)
output	(menu)

ring-mod simulates the ring modulation of each note of *ch1* by all the notes of *ch2*. The frequency of each note of *ch2* is added to and subtracted from the frequency of each note of *ch1*; thus, all the possible additive and subtractive combinations are produced.

The optional argument *unit* determines whether *ch1* and *ch2* are entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected, the values will be converted to frequencies inside the function and then the output is reconverted to midicents. If *freq* is selected, the entries, calculations and output are all in Hertz.

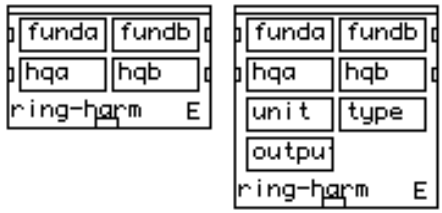
Note: Ring-modulation can produce negative frequencies; conversion to midicents will automatically reflect these notes back into the positive domain.

When *ch1* contains multiple notes, the optional argument *type* is used to determine the format of the output. The value *seq* returns a list of chords representing the modulation of each successive note of *ch1* by all the notes of *ch2*. The value *chord* returns a single chord containing all the notes of all the modulations.

The optional argument *output* determines whether the original notes of *ch1* and *ch2* are included (*inclu*) or excluded (*exclu*) from the output list or lists.

ring-harm

Simulates ring modulation of two harmonic series



Inputs

funda	(atom, list)
fundb	(atom, list)
hqa	(atom, list)
hqb	(atom, list)

Optional inputs

unit	(menu)
type	(menu)
output	(menu)

ring-harm simulates the ring-modulation between the harmonic series (see box *harm-series*) built on *funda* and the harmonic series on *fundb*. The arguments *hqa* and *hqb* determine the number of partials present for each fundamental. The frequencies of each partial of the harmonic series on *funda* is added to and subtracted from the frequency of each partial of the harmonic series on *fundb*; thus, all the possible additive and subtractive combinations are produced.

If the arguments *hqa* or *hqb* are a list, rather then including all the partials up to and including the number given: only the listed partials for both fundamentals will included in the calculations.

The optional argument *unit* determines whether *funda* and *fundb* are given in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the values will be converted to frequencies inside the function and then the output is reconverted to midicents. If *freq* is selected the entries, calculations and output are all in Hertz. (note: Ring-modulation can produce negative frequencies; conversion to midicents will automatically reflect these notes back into the positive domain.)

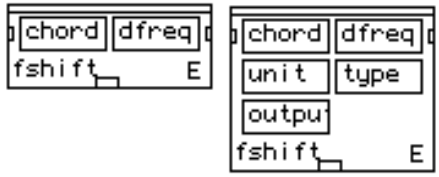
The optional argument *type* is used to determine the format of the output. The value *seq* returns a list of chords in which each successive chord represents the notes involving the next partial or partials. Thus the first chord contains: funda ± fundb; the second: 2*funda ± fundb, funda ± 2*fundb and 2*funda ± 2*fundb; etc. The value *chord* returns a single chord containing all the notes of all the combinations and differences.

The optional argument *output* determines whether the notes *funda* and *fundb* are included (*inclu*) or excluded (*exclu*) from the output list or lists.

Menu Freq harmony->Treatment

fshift

Simulates frequency shifting



Inputs

chord	(atom, list, structured list)
dfreq	(atom, list)

Optional inputs

unit	(menu)
type	(menu)
output	(menu)

fshift shifts the frequency of each note of *chord* by a frequency *dfreq* (positive or negative, but always in Hertz).

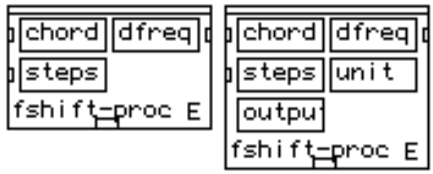
The optional argument *unit* determines whether *chord* is entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the values will be converted to frequencies inside the function and then the output is reconverted to midicents. If *freq* is selected the entry, calculations and output are all in Hertz.

If *chord* is a list of chords the optional argument *type* is used to determine whether the output will be a list of chords (*seq*), each one shifted by *dfreq* or a single chord combining the notes of all the shifted chords (*chord*). If *dfreq* is a list the same argument is used to choose between a list of chords shifted by each successive *dfreq* or a single chord combining the different distortions. If both *chord* and *dfreq* are lists the position *seq* will return a list of chords containing each chord shifted by each frequency; the position *chord* will return a list of chords containing each chord shifted by all the listed frequencies.

The optional argument *output* determines whether the original *chord* is included (*inclu*) or excluded (*exclu*) from the output list.

fshift-proc

Frequency shifts progressively



Inputs

chord	(atom, list, structured list)
dfreq	(atom, list)
steps	(atom)

Optional inputs

unit	(menu)
output	(menu)

Progressively shifts *chord* until the final chord which is shifted by *dfreq* (positive or negative, but always in Hertz). The argument *steps* determines the number of intermediate distortions to be produced between the unaltered *chord* and the chord shifted by *dfreq*.

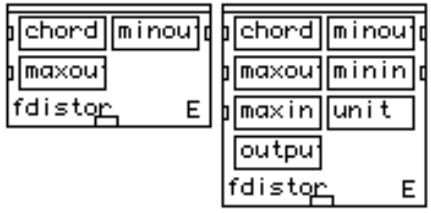
The argument *chord* may be a list, in which case the same process of shifting is carried out for each successive chord.*dfreq* and *steps* may not be lists.

The optional argument *unit* determines whether *chord* is entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the values will be converted to frequencies inside the function and then the output is reconverted to midicents. If *freq* is selected the entry, calculations and output are all in Hertz.

The optional argument *output* determines whether the non-shifted *chord* is included (*inclu*) or excluded (*exclu*) from the output list of chords.

fdistor

Simulates frequency distortion



Inputs

chord	(atom, list, structured list)
minout	(atom, list)
maxout	(atom, list)

Optional inputs

minin	(list)
maxin	(list)
unit	(menu)
output	(menu)

fdlstor distorts the frequencies of *chord* so that the lowest note is changed to *minout* and the highest note to *maxout*. Interior notes are rescaled so as to preserve the relative positions of their frequencies.

The optional inputs *minin* and *maxin* allow the scaling to be done relative to two selected reference notes rather than the highest and lowest notes of the chord. The note entered as *minin* will be moved to *minout*, and *maxin* to *maxout* the rest of the chord is then rescaled accordingly.

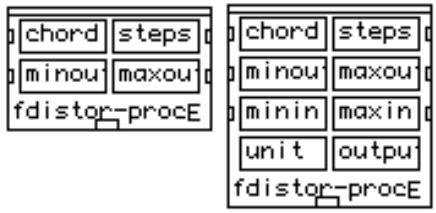
If *chord* is a list of chords, output will be a corresponding list of distorted chords.

The optional argument *unit* determines whether *chord* is entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the values will be converted to frequencies inside the function and then the output is reconverted to midicents. If *freq* is selected the entry, calculations and output are all in Hertz.

The optional argument *output* determines whether the non-distorted *chord* is included (*inclu*) or excluded (*exclu*) from the output list. If included the non-distorted notes will be mixed with the distorted into a single chord.

fdistor-proc

Frequency distorts progressively



Inputs

chord	(atom, list, structured list)
steps	(atom)
minout	(atom, list)
maxout	(atom, list)

Optional inputs

fminin	(list)
maxin	(list)
unit	(menu)
output	(menu)

fdlstor-proc progressively distorts *chord* until the distortion specified by *minout* and *maxout* is reached. The argument *steps* determines the number of intermediate distortions to be produced between the unaltered *chord* and the final distortion. (For explanation of frequency distortion, as well as the use of *minout*, *maxout*, *minin* and *maxin* see the box 'fdistor')

chord may not be a list of chords.

The optional argument *unit* determines whether *chord* is entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the values will be converted to frequencies inside the function and then the output is reconverted to midicents. If *freq* is selected the entry, calculations and output are all in Hertz.

The optional argument *output* determines whether the non-distorted *chord* is included (*inclu*) or excluded (*exclu*) from the output list of chords.

Menu Freq harmony->Analysis

harm-dist

Calculates the distance from a harmonic series

Inputs

chord (atom, list, structured list)

fund (atom, list)

Optional inputs

unit (menu)

harm-dist calculates the ratios between each note of *chord* and the closest partial of the harmonic series built on *fund*. (For explanations of harmonic series and partials, see the box **harm-series**; to decide on an approriate fundamental, the box **virt-fund** may be useful)

If *chord* is a list of chords the result consists of the analyses of each successive chord.

The optional argument *unit* determines whether *chord* and *fund* are entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the notes will be converted to frequencies inside the function before analysis.

closest-harm

Finds the closest harmonic partials

Inputs

chord (atom, list, structured list)

fund (atom, list)

Optional inputs

unit (menu)

type (menu)

closest-harm calculates the closest partial of the harmonic series built on *fund* to each note of *chord*. (For explanations of harmonic series and partials, see the box *harm-series*)

If *chord* is a list of chords the result will be the analyses of each successive chord.

The optional argument *unit* determines whether *chord* is entered in midicents, (*midic*), or in Hertz (*freq*). If *midic* is selected the values will be converted to frequencies inside the function and then the output (if appropriate) is reconverted to midicents. If *freq* is selected the entry, calculations and output (if appropriate) are all in Hertz.

The optional argument *type* determines whether the output is a list of partial rankings or the notes corresponding to those partials.

best-freq

Finds the best central frequency

Inputs

chord (atom, list, structured list)

Optional inputs

unit (menu)

best-freq returns the note that is the minimum possible distance from the frequencies of all the notes of *chord* , where the distance is measured is the minimum sum of the squares of the distances. This note can be thought of as a sort of "center of gravity" for *chord* (it is not usually a member of the chord).

If *chord* is a list of chords, the box returns a list of best frequencies.

The optional argument *unit* determines whether *chord* is entered in midicents, (*midic*), or in Hertz (*freq*). If

midic is selected the values will be converted to frequencies inside the function and then the output is reconverted to midicents. If *freq* is selected the entry, calculations and output are all in Hertz.

virt-fund

Calculates the virtual fundamental

Inputs

chord	(atom, list, structured list)
cents	(atom)

Optional inputs

unit	(menu)

Returns the highest fundamental for which the notes of *chord* could be thought of as harmonic partials. In general, the lower the virtual fundamental, the more dissonant the chord.

The argument *cents* determines the precision of the analysis. (a value of '0' would return the real fundamental; the larger the value the more approximate the result)

If *chord* is a list of chords, the box returns a list of virtual fundamentals.

The optional argument *unit* determines whether *chord* is entered and the result returned in midicents, (*midic*), or in Hertz (*freq*). The argument *cents* remains, however, unchanged.

4. Utilities

l-distor/2

Exponentially distorts a list

Inputs

newmin	(atom)
newmax	(atom)
liste	(list)

l-distort/2 distorts a list, *liste,* by a power function, thus if the list is linear the result follow the power function, if the list is non-linear the result will be a hybrid of the old liste and the power function.

The arguments *newmin* and *newmax* determine the scaling of the new list. (*newmin* will be the smallest value present, *newmax* the largest)

l-distor/3

Exponentially distorts a list

Inputs

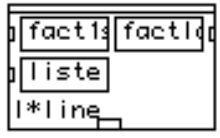
newmin	(atom)
newmax	(atom)
ref	(atom)
newref	(atom)
liste	(list)

Distorts a list, *liste*, by a power function, thus if the list is linear the result will follow the power function, if the list is non-linear the result will be a hybrid of the old liste and the power function. This box is identical to **l-distor/2** except that a reference point is controllable.

The arguments *newmin* and *newmax* determine the scaling of the new list. (*newmin* will be the smallest value present, *newmax* the largest) The values *ref* and *newref* are used to specify that the element of the original list with a value of *ref* will be moved to the value of *newref*. The curve is altered in order to accommodate the reference point.

l*line

Multiplies a list by a linear function



Inputs

fact1st	(atom)
factlast	(atom)
liste	(list)

l*line multiplies a list, *liste*, by a linear function. The first element is multiplied by *fact1st*, the last by *factlast* and all intermediate elements by linear interpolations between those values.

l*curb/2

Multiplies a list by a power function

Inputs

fact1st	(atom)
factlast	(atom)
liste	(list)

Multiplies a list, *liste*, by a power function. The first element is multiplied by *fact1st*, the last by *factlast* and all intermediate elements by interpolations along a power function between those values.

l*curb/3

Multiplies a list by a power function

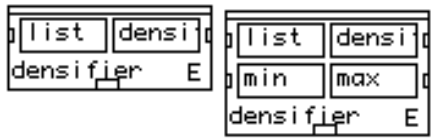
Inputs

fact1st	(atom)
factlast	(atom)
ref	(atom)
factref	(atom)
liste	(list)

Multiplies a list, *liste*, by a power function. This box is identical to *l*curb/2* except that a reference point is controllable. The first element is multiplied by *fact1st*, the last by *factlast* and the element of the original list with a value of *ref* will be multiplied by *factref*. All intermediate elements will be multiplied by interpolations along a power function between *fact1s>* and *factlast*. The power function, however, is altered to accommodate the reference point.

densifier

Increases the density of a list



Inputs

list	(list)
density	(atom)

Optional inputs

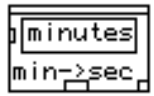
min	(atom)
max	(atom)

densifier increases the density of a *list* by adding equally spaced values between each of its elements. The number of added values is determined by the argument *density*, which is equal to the number of values added between each pair of elements.

The optional arguments *min* and *max* allow the densification to take place within only a portion of the list. Values will only be added between pairs of elements in which both members are at least *min* and no greater than *max;* all other parts of the list are returned unchanged.

min->sec

Converts minutes to seconds



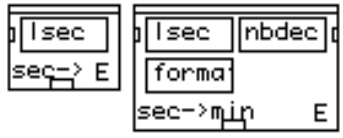
Inputs

minutes (list)

min->sec converts values in minutes into values in seconds. The value in minutes may be entered as a list in any of the following formats: (3 min), or (3 0); (3 min 30), or (3 30), or (3.5 min); (3 min 30.2), or (3 30.2). (the letters 'min' may be replaced by simply 'm' or any other non-numeric character or characters).

sec->min

Converts seconds to minutes



Inputs

lsec (atom, list)

Optional inputs

nbdec (atom)

format (menu)

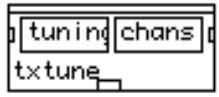
sec-min converts values in seconds (*lsec*)to values in minutes and seconds. The optional argument *nbdec* determines the number of decimals in the seconds column of the output.

The output is in the format *1 min 15* for an *lsec* equal to '75'. If the number of seconds is less than sixty the output will be in the form *0 min 32*. The optional argument *format*, if set to the position *abbrev*, will eliminate the minutes column if it has a value of '0'. (The first example would remain *1 min 15* while the second would become '32')

5. MIDI

txtune

Sends global tuning parameters to a Yamaha TX-816

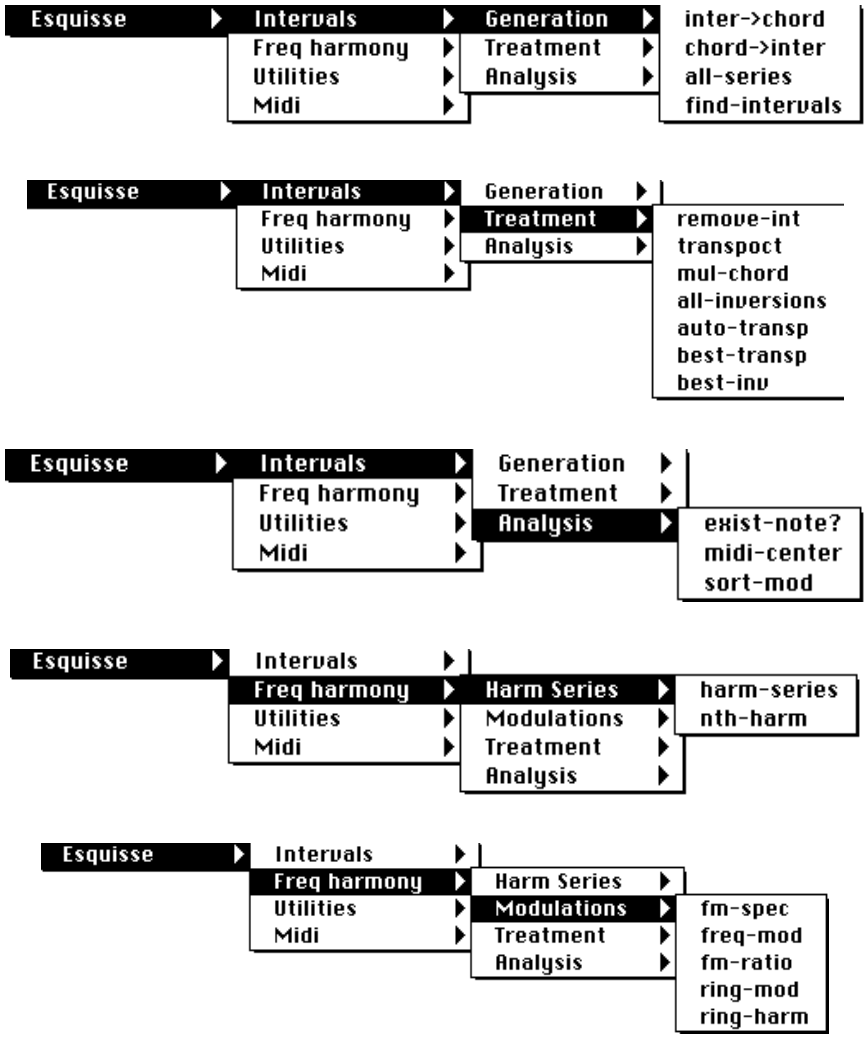


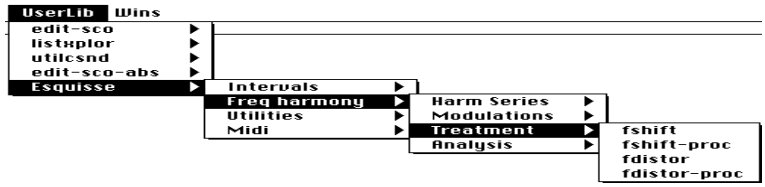
Inputs

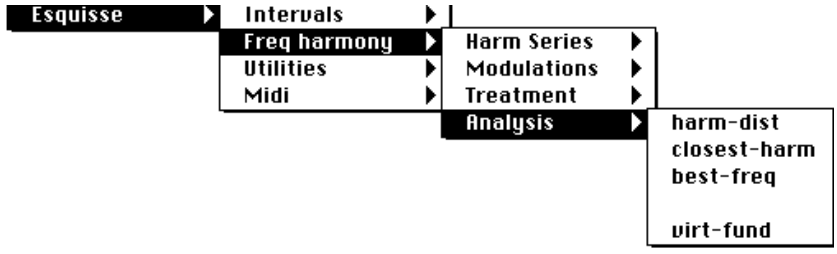
tuning	(atom, list)
chans	(atom, list)

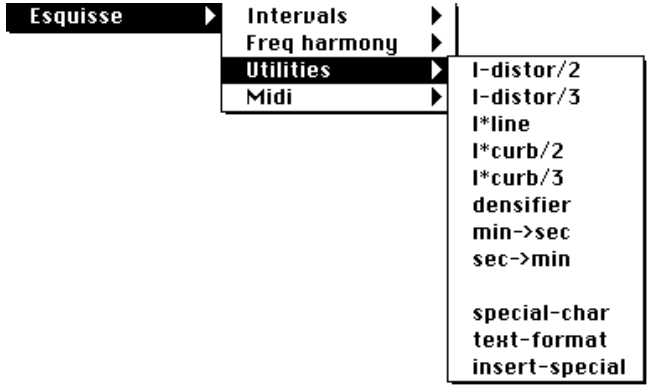
txtune sends global tuning parameters to a Yamaha TX-816 FM Tone Generator. The value *tuning* is sent to the MIDI channel specified, *chans*. If *chans* is a list, the tuning is sent to all listed channels.

Esquisse Menus




UserLib
Wins
edit-sco
listxplor
utilcsnd
edit-sco-abs
Esquisse
Intervals
Freq harmony
Utilities
Midi
Harm Series
Modulations
Treatment
Analysis
fshift
fshift-proc
fdistor
fdistor-proc


Esquisse
Intervals
Freq harmony
Utilities
Midi
Harm Series
Modulations
Treatment
Analysis
harm-dist
closest-harm
best-freq
virt-fund


Esquisse
Intervals
Freq harmony
Utilities
Midi
l-distor/2
l-distor/3
l*line
l*curb/2
l*curb/3
densifier
min->sec
sec->min
special-char
text-format
insert-special


Esquisse
Intervals
Freq harmony
Utilities
Midi
txtune

Index

T

V

Y