



# **Music 5 Mac 1.0**

by Simone Bettini

C.S.C.: Centre of computational sonology,  
Padova University.

From a PC version by Daniel Arfib, Marseille University.

# **Table of contents:**

## **1 Introduction**

**1.1 About this manual**

**1.2 The Music 5 History**

**1.3 About Music 5 Mac**

## **2. M5Mac User guide**

**2.1 Overview**

**2.2 The modules.**

**2.3 The windows.**

**2.4 The menus**

**2.5 Examples**

**2.5.1 Drawing an instrument:**

**2.5.2 Converting the graphical description of an instrument in a fragment of script.**

**2.5.3 Writing a script.**

**2.5.4 Playing a script (from a window and from a file).**

**2.5.5 Extracting the graphical description of an instrument from a script.**

**2.5.6 Converting sounds**

## **3. Music V Handbook**

**3.1 Overview**

**3.2 Structure of a score**

**3.3 Instruction syntax**

3.3.1 Instructions in detail

3.3.2 Function generators

**3.4 Instrument definition**

**3.5 The Modules**

3.5.1 Inputs and outputs (first button)

3.5.2 The oscillators (second button)

3.5.3 Table readers, random generators, file readers (third button)

3.5.4 The arithmetical functions (fourth button)

3.5.5 Filters, delays, distortions (fifth button)

3.5.6 Reverberators (sixth button)

## **4. Acknowledgements and references.**

**4.1 Release notes**

**4.2 Acknowledgements**

**4.3 References**

## **1 Introduction**

### **1.1 About this manual**

This manual isn't a Music V manual, but simply a user guide for Music 5 Mac. If you don't know anything about music V you'd better to search some book explaining how it works[1]. A basic knowledge of Macintosh user interface and its terminology [2] and of sound synthesis principles is assumed.

Here is what you'll find in this manual:

- Chapter 1 (this one) contains this paragraph, an historical introduction to Music V and an introduction to Music 5 Mac.
- Chapter 2 (User guide) explains Music 5 Mac user interface and the operations that it performs.
- Chapter 3 is a Music V handbook, explaining syntax, commands and modules of this Music V version, with several examples and schemes.
- Chapter 4 contains acknowledgements and some references.

### **1.2 The Music 5 History**

In the late sixties a young researcher of Bell laboratories named Max Mathews had the idea to simulate the functions of an analog synthesizer using a program on a computer. Even if nowadays it seems obvious, at that time it was a completely innovative concept. We must remember that in these years computers were very rare and usually they were present only in research laboratories. Max's idea allowed to solve many critical problems about synthesizers project: with a "software" synthesizer it is possible to try new configurations using several "components" without physically mounting them, all components of the same type are "identical", and they can't break.

Because of technical limitations the sound couldn't be generated in "real time", so he decided to make its program to generate the sound in "deferred time". The problems that caused this choice were two: at that time computers were much slower than today, and generating even the simplest sound in real time needed a computational effort very near to the machine capabilities; moreover the complexity of an instrument and the number of notes to be played at the same time cause the computation time to enlarge in an unforeseeable way.

On the other side this concept gives to the composer the freedom to write the score in any order he likes, just like a cartoonist can create the scenes and the single frames of a cartoon in any order, as the computer will sort and synchronize them automatically.

The realization of Max's idea originated a music language and a series of compilers named Music I, Music II, Music III, Music IV and Music V, where the number denotes various versions.

Using this language the composer can write a score containing not only the notes to be played but also descriptions of the instruments on which they will be played.

The orchestra description specifies each instrument in the orchestra in terms of the type of each module (oscillator, reverberator, adder, filter...) in it and how the unit generator are interconnected or related.

Many interconnections are possible. For example the outputs of two oscillators can be added to produce a more complex tone, or one oscillator can control the frequency of a second oscillator to produce a vibrato.

Inherent in the description of each instrument are the input parameters needed to run it. For example if the instrument is to play notes of differing pitches, one input parameter must specify pitch. If vibrato is to have a controllable rate, a parameter must specify this rate. These parameters must eventually be supplied by the score.

Different instruments must be uniquely designated. This is done simply by numbering them. Thus the program will have instruments 1, 2, 3, and so forth, and the score will request a note to be played by an instrument number.

### **1.3 About Music 5 Mac**

The original idea was to make a port of a PC Music V compiler written in Marseille university by Daniel Arfib and to give it a complete macintosh user interface.

For this reason I decided to merge such compiler with a text editor, in order to edit the scripts to "play", and with a graphic instrument editor inspired to the diagrams I had seen used in Padova to make easier the construction of instruments.

Afterwards some sound conversion routines were added in order to simplify the use of the file reading capability of this Music V version.

## 2. M5Mac User guide

### 2.1 Overview

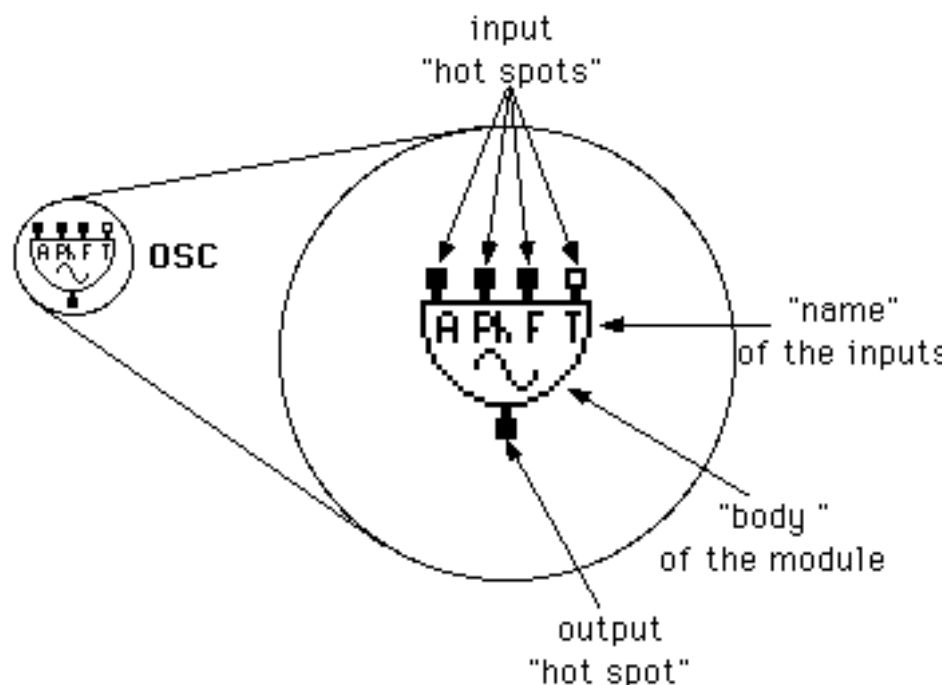
At this time the application performs the following functions:

- a script can be played both if present in an active edit window of the application, and from a non open file.
- it is possible to edit a script, and to transform the "alphabetical" description of its instruments in the correspondent "graphical" representation.
- it is possible to graphically create and edit an instrument and obtain its "alphabetical" description to use in a script.

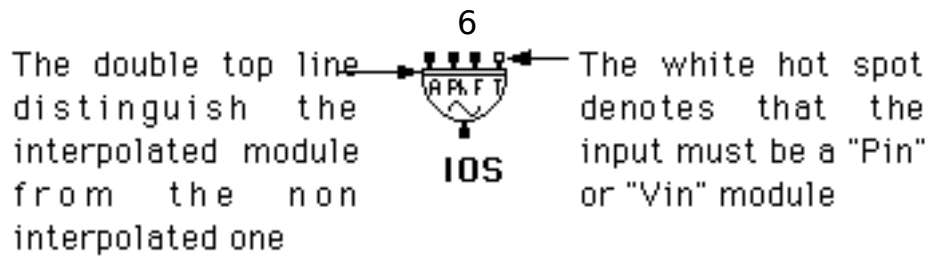
In this chapter we will first describe modules structure, Music 5 windows and menus, then we will see some examples of program use.

### 2.2 The modules.

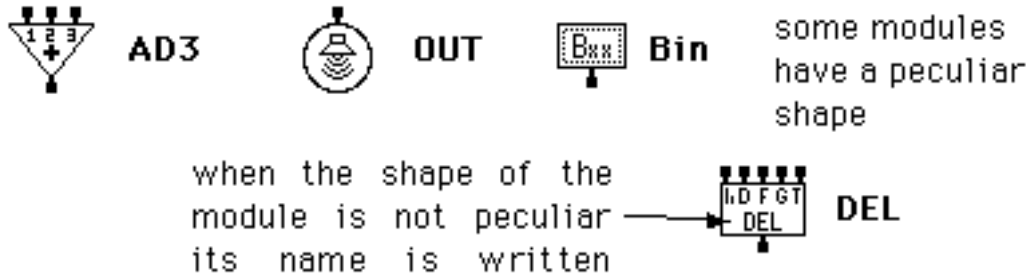
The structure of a module can be divided in three parts: the inputs, the body, the output.



**Structure of a module**



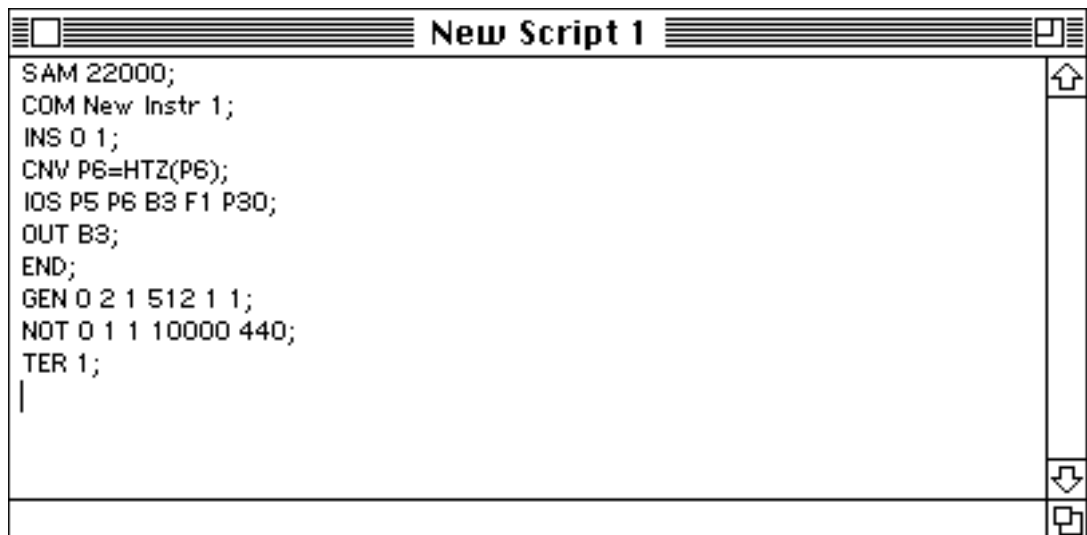
Some modules have a their own shape that distinguish them from others, anyway when the shape of a module don't characterise it, its name is written inside of it.



### 2.3 The windows.

M5mac manages two types of windows:

- a simple text window, with usual capabilities of insertion, cut, copy, paste, an additional function of find and replace and the possibility to choose text's font and size.

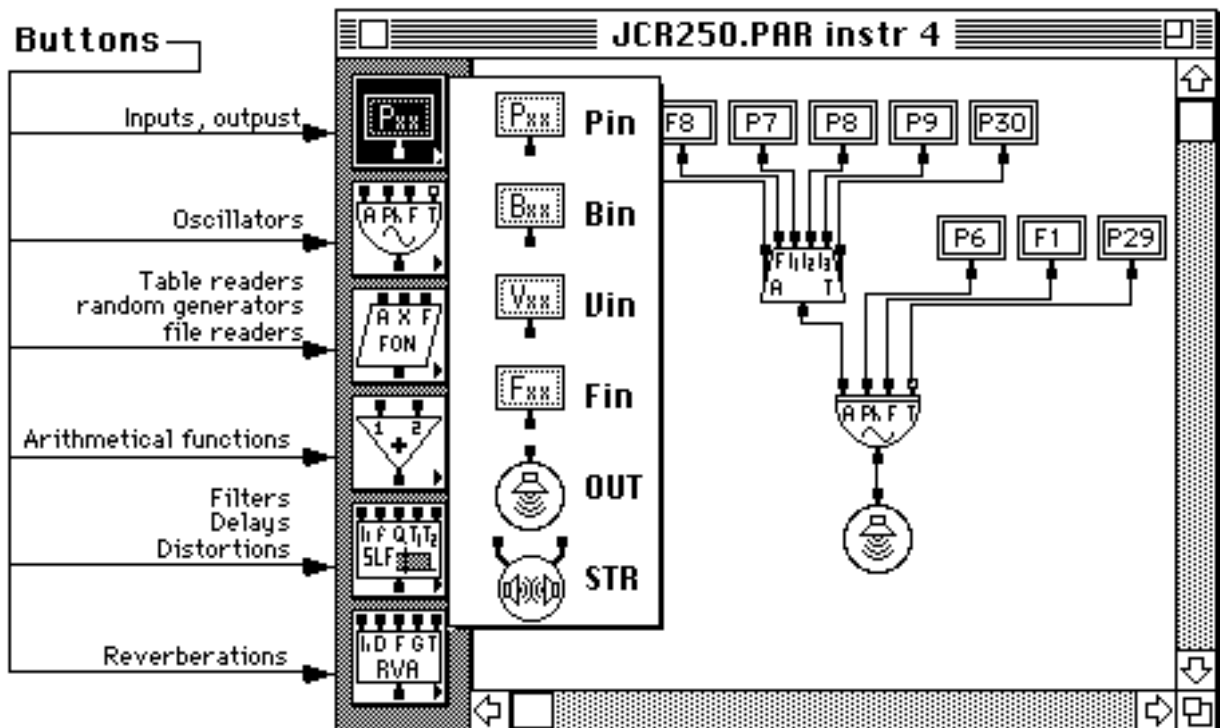


Script Window

- an instrument window on which are present 6 “buttons” from which is possible to choose the modules to use in order to build an instrument.

The modules can be “picked” from the correspondent button, or from the pop-up-menu associated, “put” in the window and joined simply clicking and dragging from a “hot spot” of a module to a “hot spot” of another module.

Besides it is possible to select one or more modules (clicking or shift-clicking) in order to move, delete, copy, cut, paste them.

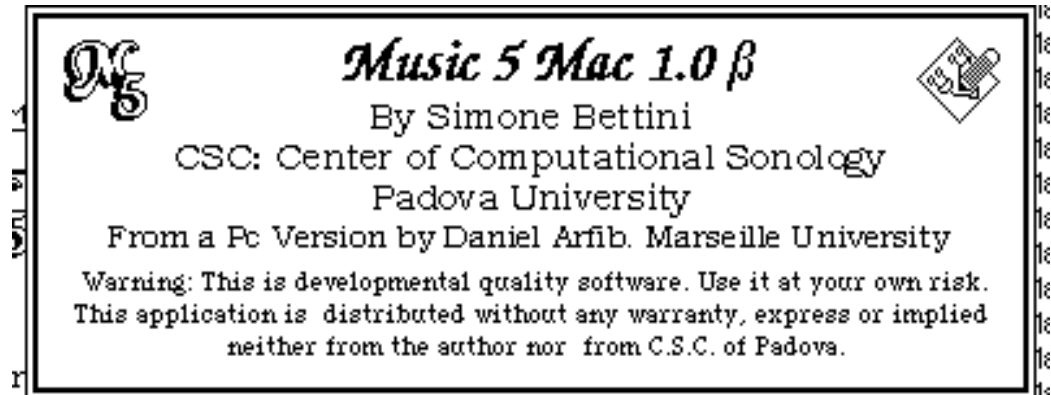


**Instrument Window**

## 2.4 The menus

- **Apple Menu**

Under the Apple menu is present the standard “About M5mac...” item that will make come up a simple alert with information about author and CSC of Padova.



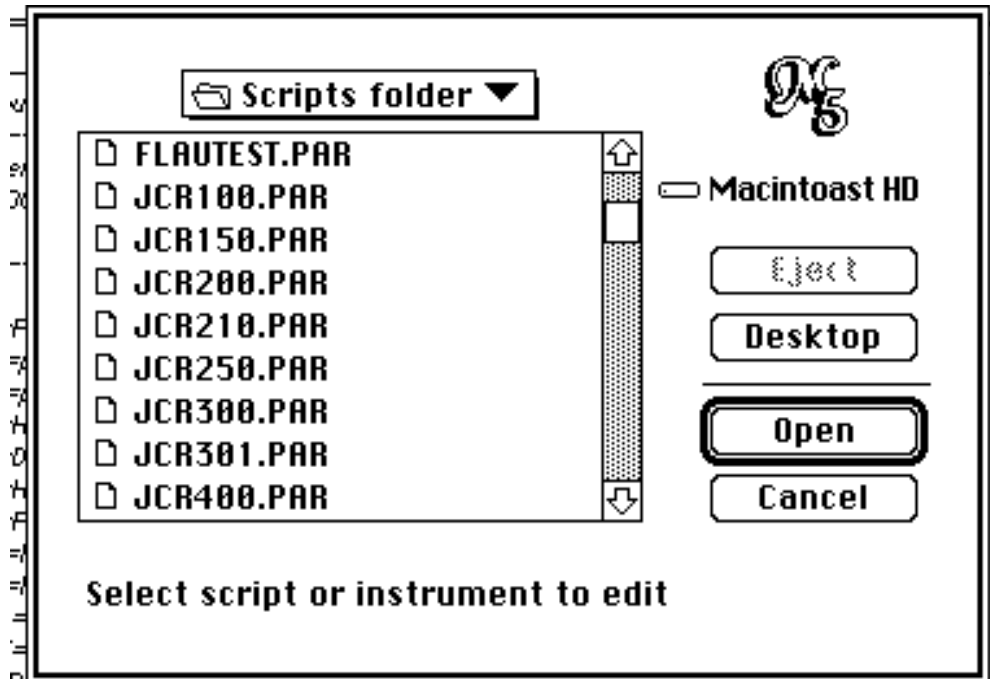
### About M5Mac ... alert

#### • File Menu

File	Edit	Special
New Script	⌘N	
New Instr	⌘I	
Open...	⌘O	
Close	⌘W	
-----		
Save	⌘S	
Save as...		
Revert...		
-----		
Page Setup...		
Print...	⌘P	
-----		
Quit	⌘Q	

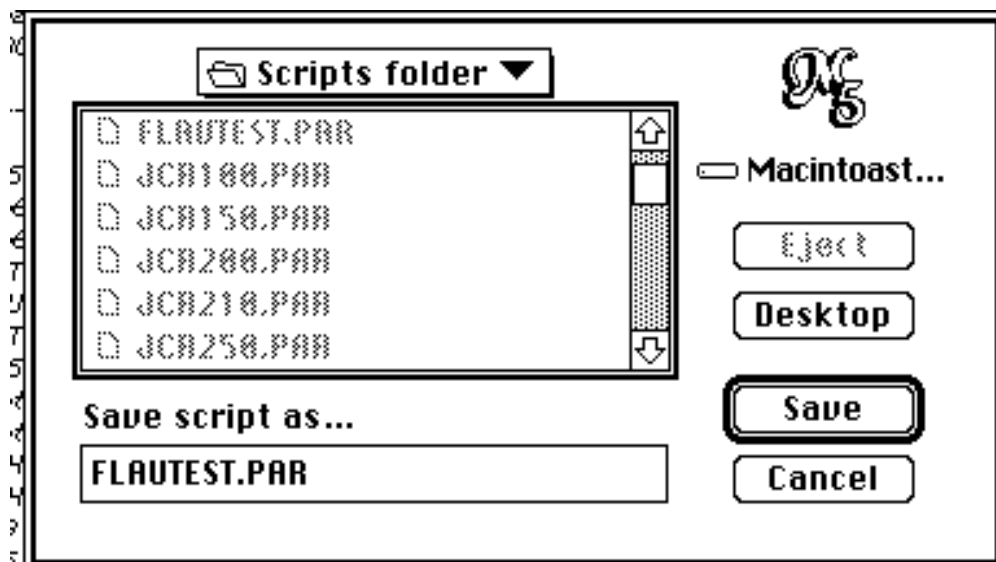
- New Script : it creates and shows a new untitled empty text edit window.
- New Instrument : it creates and shows a new untitled empty Instrument window.
- Open... : it shows a Standard File open dialog from which is possible to open both a script (text) and an instrument document.





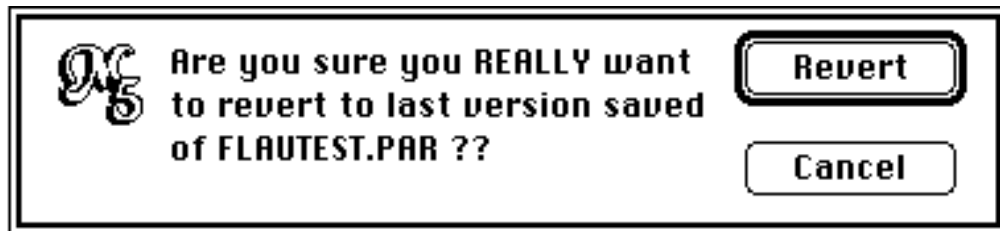
**Open... dialog**

- Close : it closes the actually active script or instrument window
- Save : it saves the actual state of the document relative to the active window (script or instrument) if it exists, prompts for a new document name if not.
- Save as... : prompts for a new document name where to save the actual state of the active window (script or instrument) .



**Save as ... dialog-**

- Revert : allows to revert to last saved version of the document relative to the active window (script or instrument).



### Revert alert

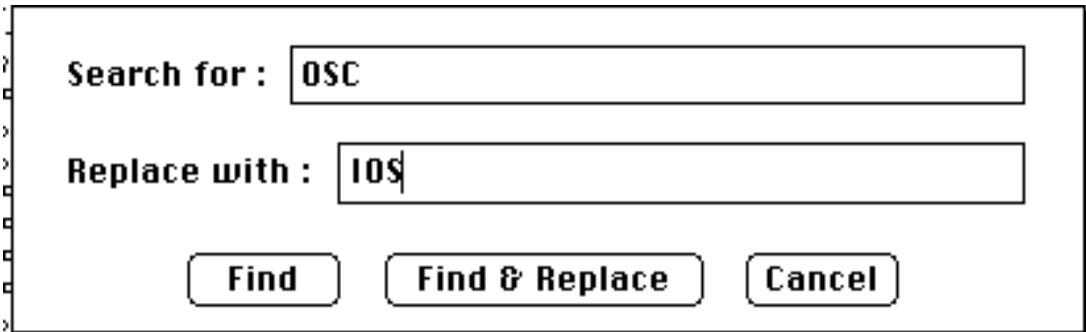
- Page setup : allows to choose the size of the paper that will be used to print the contents of the active window (script or instrument), its changes are reflected in the size of the instrument “sheet” if the actual window is an instrument one. if the size chosen is smaller than the previous and some modules “falls” outside of the sheet, they will be automatically repositioned inside it.
- Print... : it allows to print the content of the document relative to the active window.
- Quit : to exit the application.

### • Edit Menu

Edit	Special
Undo not implemented	⌘Z
Cut	⌘H
Copy	⌘C
Paste	⌘V
Clear	⌘K
Select All	⌘A
Duplicate	⌘D
Find...	⌘F
Find Again	⌘G
Replace	
Replace & Find again	
Replace all	
Mod info	⌘M

- Undo : this item hasn't still been implemented
- Cut : to cut the actually selected text or modules (accordingly the window kind)
- Copy : to copy the actually selected text or modules

- Paste : to paste the previously copied/cutted text or modules
- Clear : to delete the actually selected text or modules
- Select All : to select the entire content of the actual window
- Duplicate : to duplicate the actually selected text or modules
- Find... : brings up a dialog in which is possible to type the wanted string and, in case, the replacement one.

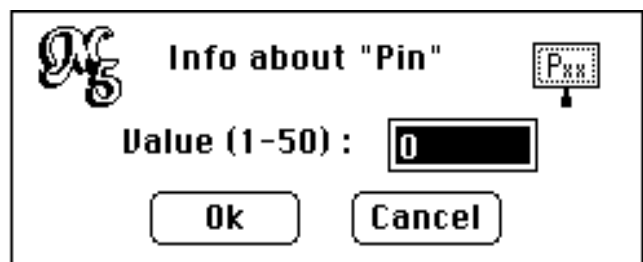


**Find... dialog**

- Find again
- Replace
- Replace and find again
- Replace all

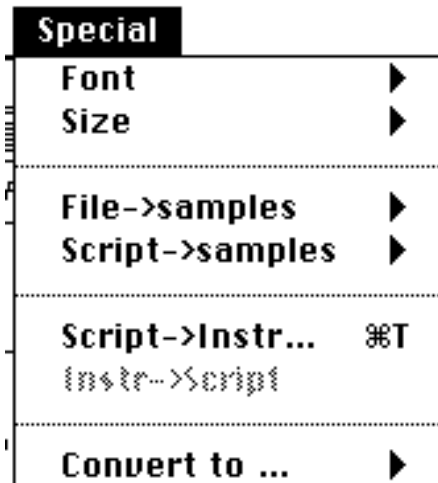
they perform the relative operation on the current active SCRIPT window.

- Mod Info... : if the active window is an Instrument one and a IN (Pxx, Bxx, Vxx, Fxx) module is selected it brings up a dialog with which is possible to associate the input module with its number (eg. P11, B2, V4, F3)

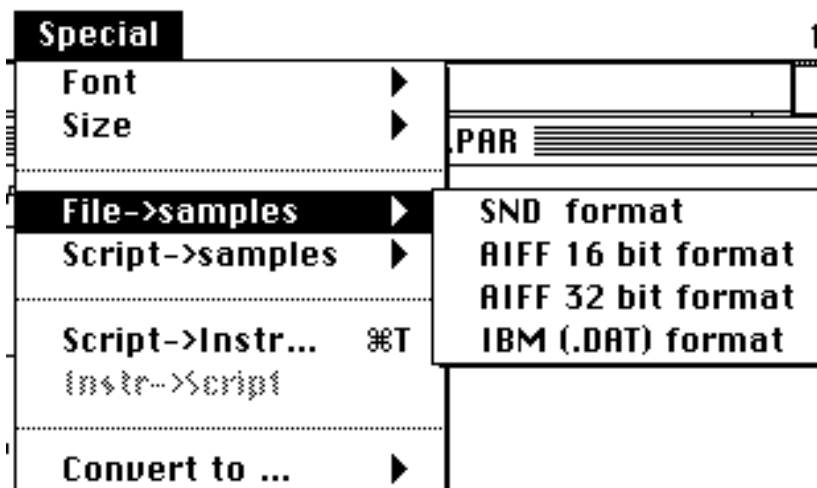


**Mod info... dialog**

- **Special Menu**

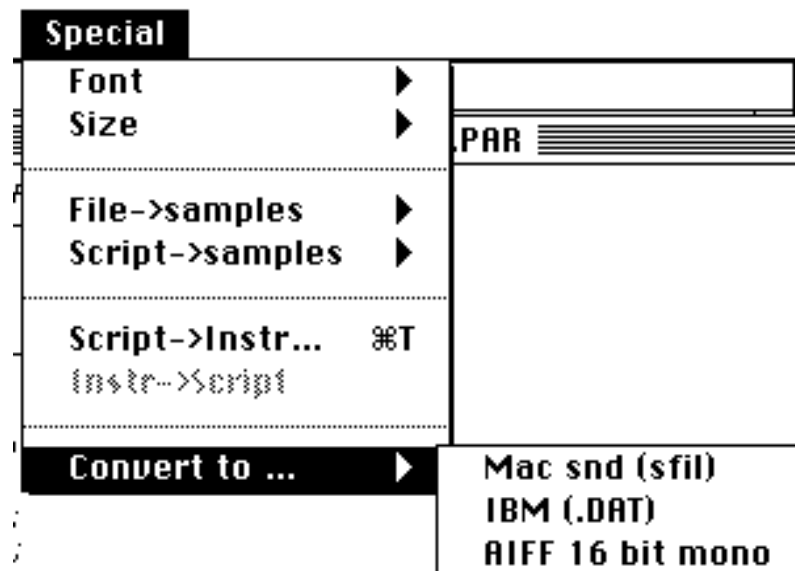


- Font : a hierarchical menu from which to choose the desired font
- Size : a hierarchical menu from which to choose the font's size
- File->Samples : this item allows to make the application generate the samples relative to a file containing a score in the desired audio format: 4 formats are available : Macintosh sound (sfil, snd resource), 16 bit AIFF, 32 bit AIFF, IBM (.DAT, that is a file of integer)
- Script->Samples : this item performs the same operation of the preceding, but using the script contained in the active script window.



**File(and Script)->samples submenu**

- Script->Instrument : allows to obtain the graphic version of any of the instruments described in the active script window : it brings up a small dialog with a pop up menu containing a list of all instruments of the script. Choosing the desired one and clicking on the arrow makes the application to open a new instrument window with its graphic representation.
- Instrument -> Script : performs the opposite operation: starting from a instrument drawn in a instr window it opens a new script window containing a fragment of script with its "alphabetical" description.
- Convert to ... Selecting a format from the submenu of this item (Mac snd, IBM file, AIFF 16 bit mono) appears a Standard File open dialog from which is possible to open a sound of and to convert it in the selected format.



**Conversion formats submenu.**

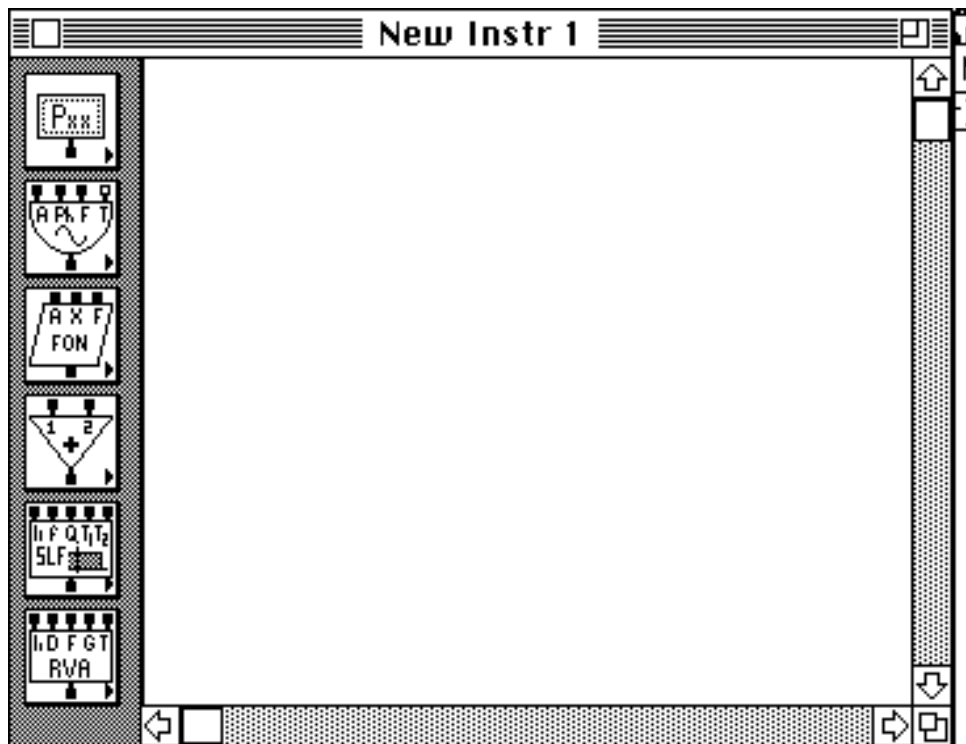
## 2.5 Examples

in this section will be presented some simple examples of the fundamental operation performed by M5Mac:

- Drawing an instrument.
- Converting the graphical description of an instrument in a fragment of script.
- Writing a script.
- Playing a script (from a window and a file).
- Extracting the graphical description of an instrument from a script.
- Converting sounds.

### 2.5.1 Drawing an instrument:

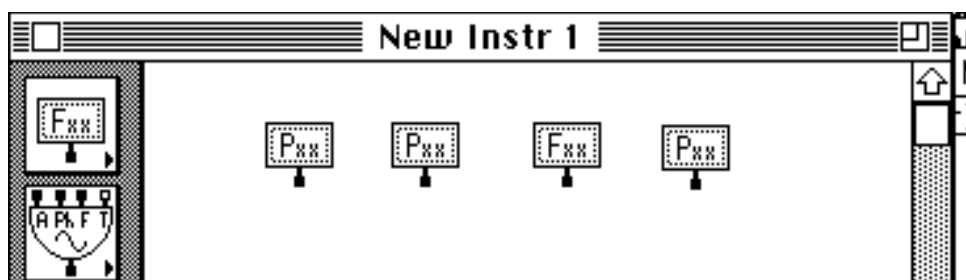
In order to draw an instrument you need an Instr Window, so choose “New Instr” from the file Menu.



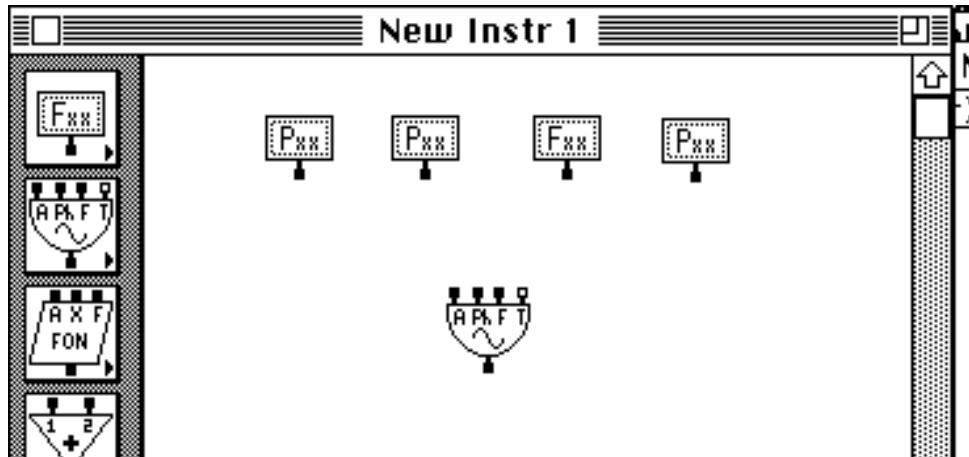
You now can start drawing your instrument. In this example we will draw the simplest one: an oscillator.

First take the input mods from the first button at the top of your Instr window: we will need 3 P inputs and 1 F input.

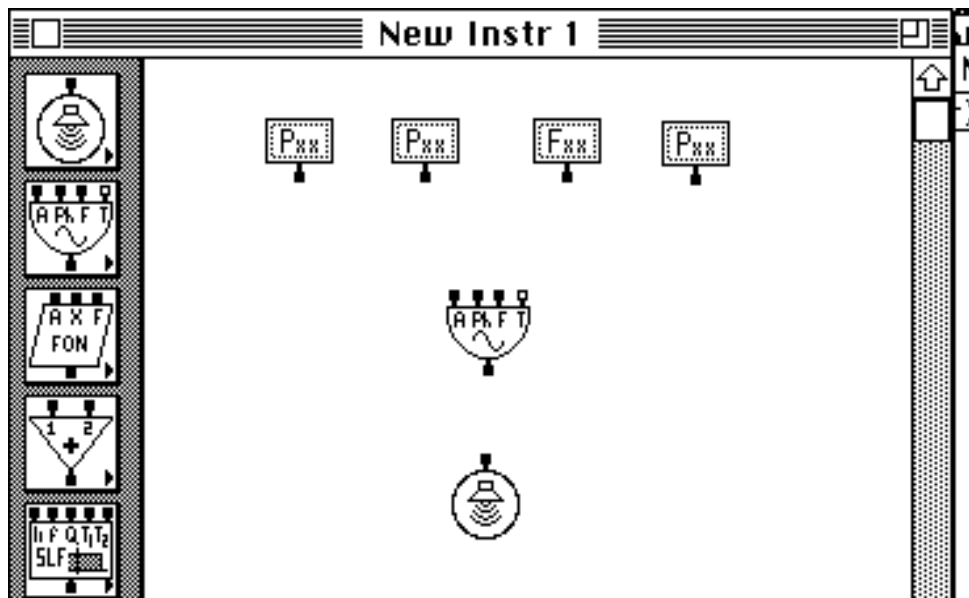
Arrange them at the top of the window as shown.



Now take the OSC module and put it under the inputs.



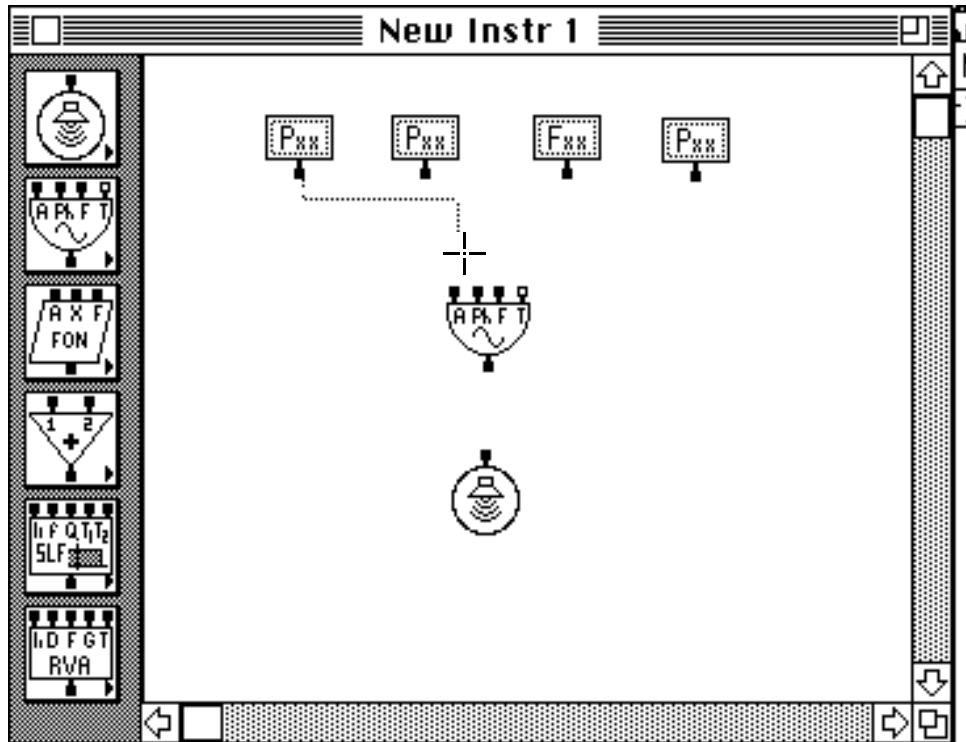
Finally take the OUT module and put it under the OSC.



Now we have to join the outputs and the inputs of our modules: let's start from top to bottom (just to choose an order).

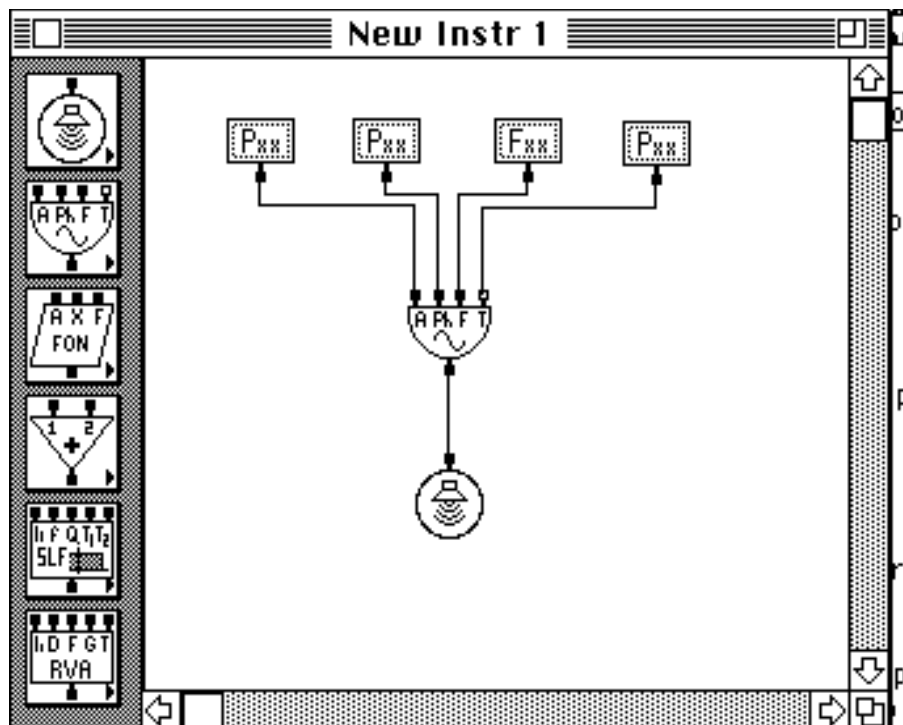
Click on the hot spot of the first Pin module and drag the split line to the first input of the OSC (A).





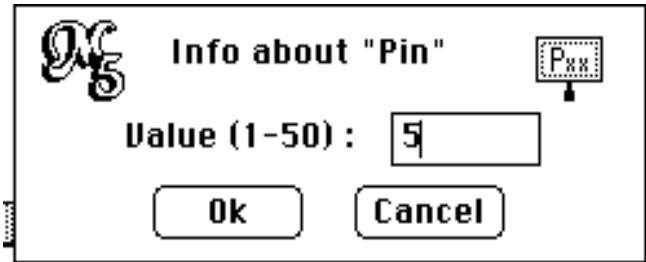
Now do the same for all others inputs, being careful to join the Fin with f input of OSC.

To finish join OSC output with OUT input.

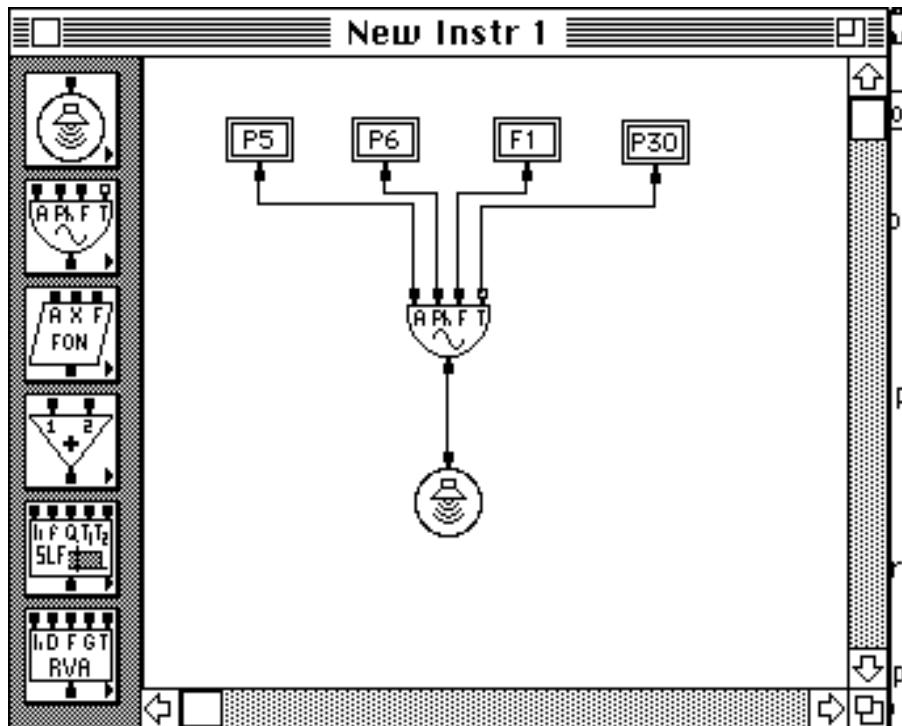


Now we have only to assign to the inputs their values .

We will assign to the inputs values (from left to right) respectively 5,6, and 30 to **Pins** and 1 to **Fin**. Select the leftmost Pin and choose "Mod Info" from Edit menu: a dialog relative to that module will come up. Type in "5" and click OK.



Repeat the same operation for all others input modules and assign them their values. Here is the result. You can see the assigned values inside the inputs modules.



That's all! It really takes more to say than to do.

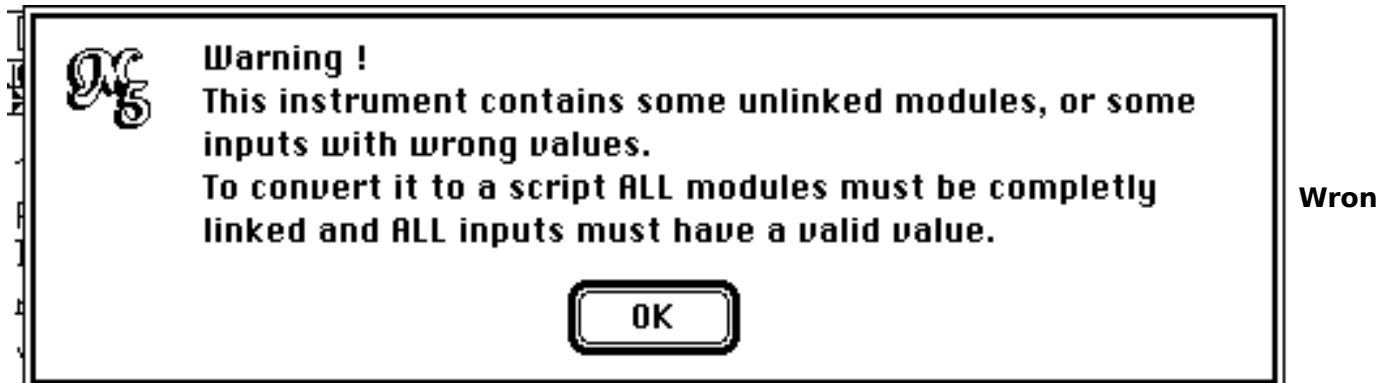
You now can save your first music 5 instrument (we will use it in the next examples).

## 2.5.2 Converting the graphical description of an instrument in a fragment of script.

There are two important condition to respect in order to convert an Instrument in a fragment of script :

- All modules must be completely linked (all hot spots must be joined each others)
- All input modules (Pxx, Bxx, Vxx, Fxx) values must have been initialised (see previous example)

Let's go on with the previous example: if you followed it exactly then all is ready, otherwise don't worry: the program will advise you that something is wrong.

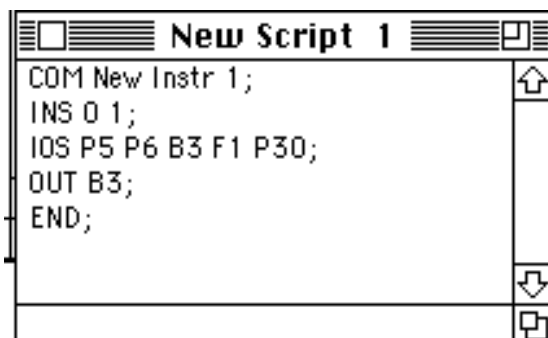


**g instrument alert**

Select the window of your oscillator (the one of previous example).

Now select "Instr->script" from the windows menu.

Wait a moment ... et Voila': the script window that is appeared contains the "music 5 language" description of our oscillator.

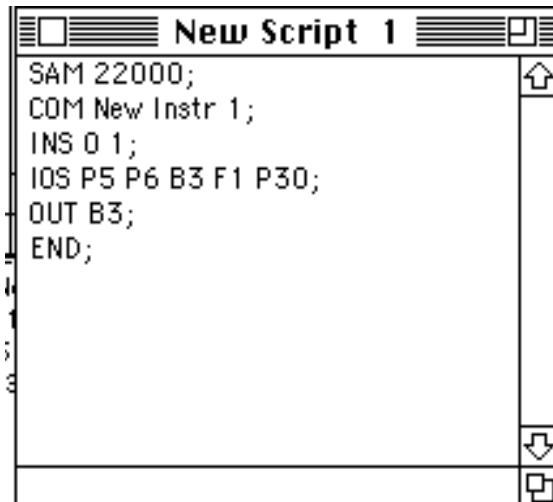


**"Translated" instrument**

If you decide to go on with the next example don't close it : we will use it.

### 2.5.3 Writing a script.

In this example we will simply complete the script obtained in the previous example: First let's indicate the sampling rate: let's write "SAM 22000;" before the description of the instrument.

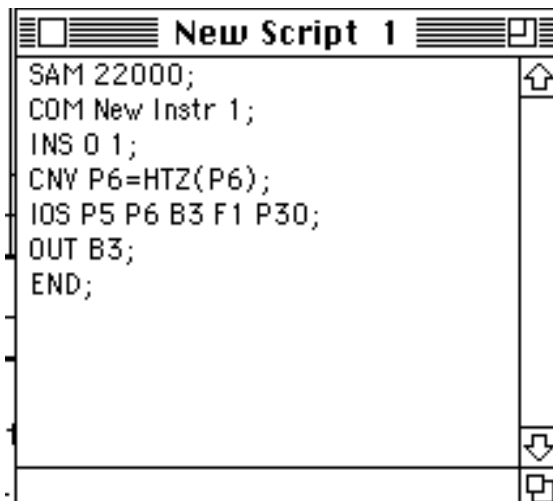


```

SAM 22000;
COM New Instr 1;
INS 0 1;
IOS P5 P6 B3 F1 P30;
OUT B3;
END;

```

We have to complete the description of the instrument with a conversion command: let's write "CNV P6=HTZ(P6);" after the INS statement (for details about CNVs see next chapter)



```

SAM 22000;
COM New Instr 1;
INS 0 1;
CNV P6=HTZ(P6);
IOS P5 P6 B3 F1 P30;
OUT B3;
END;

```

Now let's define the function used from the instrument: let's write "GEN 0 2 1 512 1 1;" after the description of the instrument (for details about GENs see next chapter)

```

SAM 22000;
COM New Instr 1;
INS 0 1;
CNV P6=HTZ(P6);
IOS P5 P6 B3 F1 P30;
OUT B3;
END;
GEN 0 2 1 512 1 1;

```

Finally we have only to make the instrument play a note and to terminate the score: let's add "NOT 0 1 1 10000 440;" and "TER 1;" at the end of the script and save it.

```

SAM 22000;
COM New Instr 1;
INS 0 1;
CNV P6=HTZ(P6);
IOS P5 P6 B3 F1 P30;
OUT B3;
END;
GEN 0 2 1 512 1 1;
NOT 0 1 1 10000 440;
TER 1;

```

The script now should look like this:

```

SAM 22000;
COM New Instr 1;
INS 0 1;
CNV P6=HTZ(P6);
IOS P5 P6 B3 F1 P30;
OUT B3;
END;
GEN 0 2 1 512 1 1;
NOT 0 1 1 10000 440;
TER 1;

```

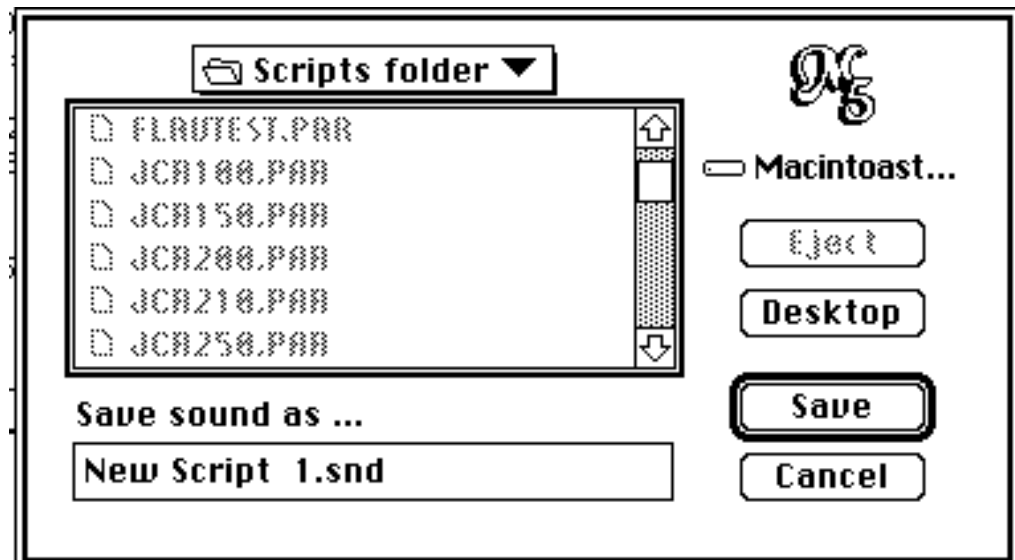
If you want to go on with the next example don't close the script window: we will use it.

#### 2.5.4 Playing a script (from a window and from a file).

Now we will try to get some sound from our hard work:

with the window relative to the script of our oscillator active (if you didn't follow preceding examples simply copy the script at the end of it or open a .PAR file) choose "script->Samples" from windows menu and choose "SND format" from its submenu.

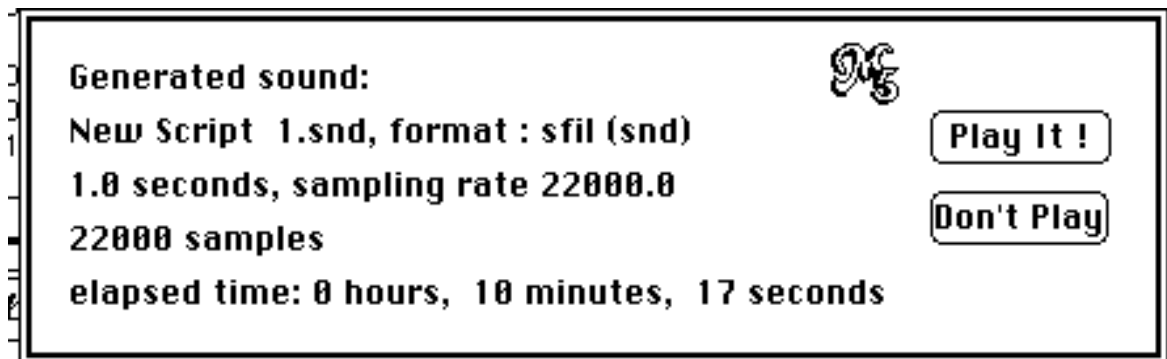
You will be asked to choose the name of the samples file: type in the name you like and click on "Save".



Now a progress dialog should appear, and the bar should flash twice (pass1 and pass2 execution) and next slowly fill (pass3 execution).



At the end a report dialog appears with a "Play It" button.



### Report dialog

Push it and hear !!

Switching to the finder (or quitting if you don't use multifinder or System 7.xx) you will find a sfil file, the one with the loudspeaker icon in System 7.xx [ , with the name you had choose. If you are running System 7.xx, you can just double-click it to listen it again, otherwise you will have to use an appropriate application (resEdit for example).

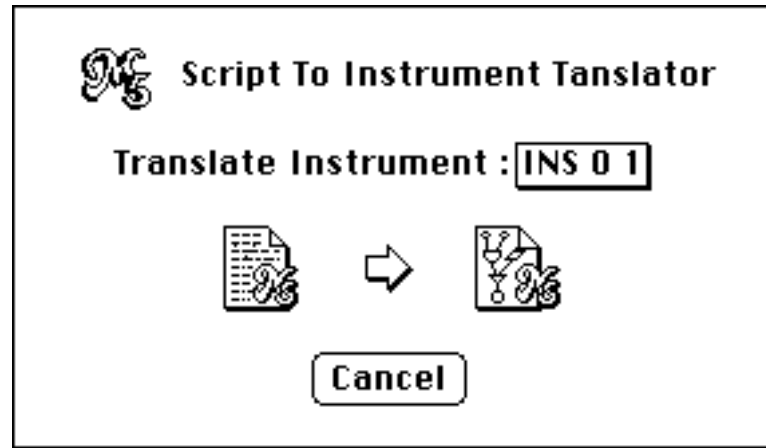
If you now want to try "File->Samples" item in windows menu you can save your script as "myFirstScript.PAR" (for example), close your window and choose the item. You will be prompted to choose a file. Choose the one you just saved and click OK. The rest should go on like in "Script->samples" case.

### 2.5.5 Extracting the graphical description of an instrument from a script.

If you have closed your oscillator window, re-open the file (or open another script).

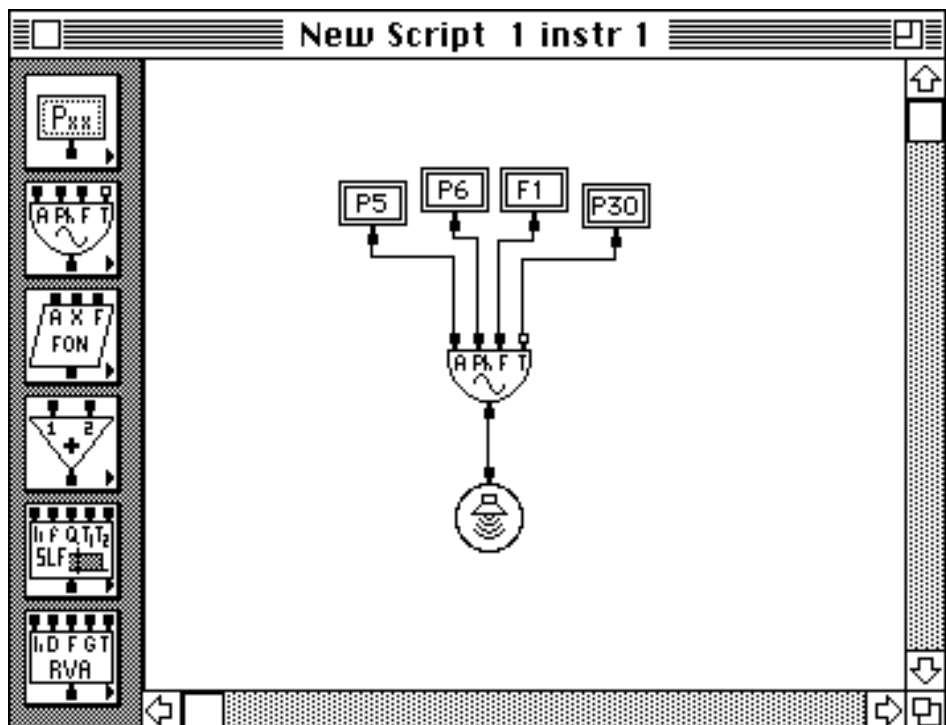
With the script window active select "Script->Instr" from windows menu: will appear a dialog with a pop up menu with written something like "INS 0 1".

If the script contains more instrument a small down arrow will be present on the right of the pop up menu.



Choose the instrument you want to translate and click on the arrow icon in the middle of the dialog.

A new Instr window with the graphic representation of the INS you choose will appear (if you did use the oscillator script of the preceding examples you should see an instrument nearly identical to the one you had drawn).



## 2.5.6 Converting sounds



This utility have been initially added in order to allow users to convert the sounds of different formats to the AIFF 16 bit mono format used for input in the scores (FIC, LUM, GEN...) and have been later extended to the conversion to the others.

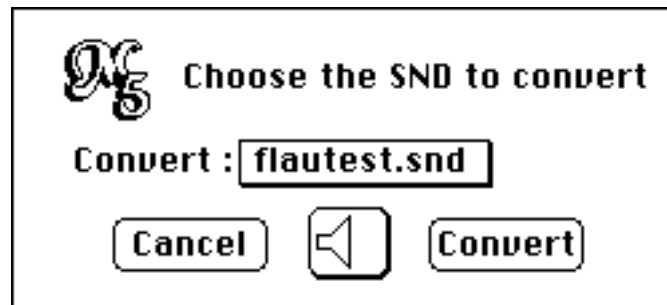
This procedure will convert snd files, files of integer (ibm .DAT) and AIFF 16 bit (both mono and stereo), and will extract snd resources from resource files.

The output format can be : Mac sound docs (sfil: the ones with the loudspeaker icon), files of integer (ibm .DAT) and AIFF 16 bit mono.

It is possible to choose the destination format from the “Convert to...” menu submenu.

A Standard file dialog allows to choose a file of one of the following types: Mac sound (sfil), a resource file, AIFF or IBM data (.DAT).

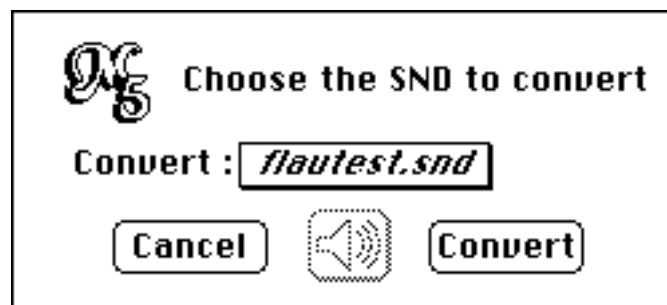
- If the file is a Mac sound file or a resource file a dialog like this appears:



the submenu contains a list of all the sounds in case present in the file.

With the central button is possible to hear the sound.

If an item's style is italic, it means that the sound is too large to be played, and if selected the “play” button becomes dimmed:



It is anyway possible to convert it.

Selecting convert button a dialog asking the name for the new file will appear.

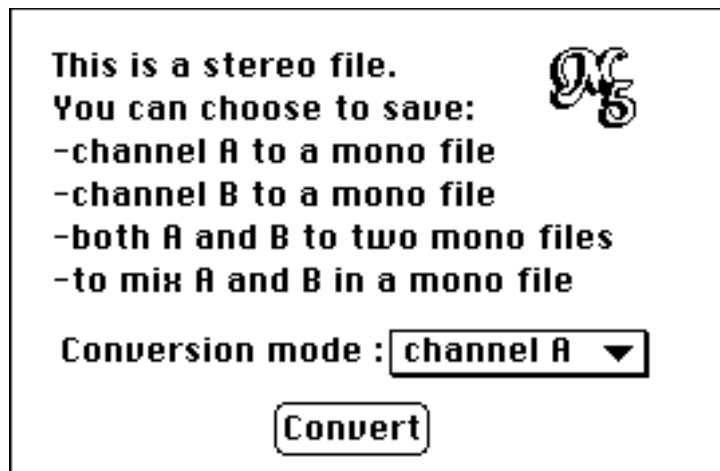
- If the “source” file is a data (ibm .DAT) file, as those file haven't any header, the user will be asked if the selected file is stereo or mono:





**mono/stereo selection dialog**

If the stereo button is selected the following dialog appears:



now it is possible to choose the output format desired and save the converted file.

•If the “source” file is an AIFF file the application will check if the file is stereo and, in case, will bring up the previous dialog.

### 3. Music V Handbook

#### 3.1 Overview

Music V is a program for sound synthesis.

The composer supplies it a score (script) containing both description of notes and description of instruments (simulation models).

Its execution is divided in three parts or passes: during pass1 the program checks and translates the score, during pass 2 the instructions are chronologically ordered, in pass 3 samples are generated.

#### 3.2 Structure of a score

We can divide a music 5 score in 2 parts : instructions and instrument definition.

Instructions tell to music 5 to play a note, to generate a function, to set a the value of a variable, to open a file...

In instrument definition we put the instrument's model expression, written in Music V "language".

#### 3.3 Instruction syntax

All Music V instructions start with a operative code (OPCode) of three letters and end with a semicolon (";"). An instruction can lay on more lines, and more instructions, divided by a semicolon, can stay on the same line.

In instructions the OPCode is followed from operands that are numbers divided from a space.

Each m5 instruction is considered to be divided in fields, called P-fields, associated to increasing numbers. The first field is the OPCode field.

Example:

P-fields :	P1	P2	P3	P4	P5	P6
	NOT	0	1	2.3	4	123
	SV2	0	20	23	22	440

### 3.3.1 Instructions in detail

#### NOT

This is the fundamental sound generating instruction.

Syntax :

```
NOT startTime insNum duration P5 P6 ... P30;
```

NOT makes *insNum* instrument to play a note with P5...P30 parameters, starting at *startTime* time for a *duration* time. It isn't necessary to define the value of all P parameters. Undefined ones are assumed to be zero.

#### INS and END:

A INS instruction opens Instrument *insNumber* definition, and a END one closes it:

```
INS defTime insNumber;
mod 1 ...
mod 2 ...
...
END;
```

#### SV1, SV2, SV3, SIA:

Those instructions allows to define the value of the program's "user variables" V1... Vn, actually 50 (stored in an array), respectively in pass1 (for use by PLF), in pass2 (for use by PLS), in pass3 (for use by instruments), and in all passes.

Syntax:

```
SVx defTime firstVar Val0 Val1...Valk;
```

where *defTime* is the "time" at which the variables are defined, *firstVar* is the first variable (in the array) to be defined, *Val0 Val1...Valk* are the values associated respectively to *firstVal, firstVal+1...firstVal+k* variables.

**Note:** Variables 1...10 are predefined as storage for general purpose values: number of channels (8), sampling rate (4)... and for program use.

#### SEC and TER:

SEC and TER allows us, respectively to separate a score in more SECTions of a certain duration and to TERminate the last (or only) section at a certain time.

Whereas the use of TER is always necessary to close the execution of the score, the use of SEC can be sometimes optional: the amount of memory used from music 5 to store the definition of instruments and the NOT commands is fixed, and thus sometimes it is necessary to “divide” the score in more sections in order to not exceed the total available memory, moreover we can sometimes prefer divide it in sections for “logical” reasons.

Syntax:

```
SEC endTime;
TER endTime;
```

**Note:** the TER and SEC endTime are ALWAYS respected. If last (chronologically) note ends BEFORE endTime, then the remaining time is “filled” with silence. If it ends AFTER endTime, then it is truncated at endTime time.

### **PLF, PLS:**

These are pass1 and pass2 subroutines, originally definable from the user. Their purpose is to modify the original score in some way: PLF, acting in pass1, can be used to generate notes, chords and Music V instructions; PLS, acting in pass2, can be used to modify the instructions parameters.

As originally music 5 was a fortran application (later version, as the one I adapted, where in Pascal) the user, who had the sources, could write himself his personal PLF and PLS subroutines, add them to the code and recompile all the program.

I hadn't the time to study how to make my application accept external code resources, like some others applications do, so the only way is still to modify the original code and to recompile all the program.

### **CHN, SAM:**

They are “shortcuts”:

CHN *numChannels* with a numChannels value of 1 or 2 is used to set the number of channels instead of using “SIA 0 8 numChannels;” instruction.

SAM *samplingRate* is used to set the sampling rate instead of using “SIA 0 4 samplingRate ;” instruction.

**COM:**

It is possible to insert comments in the score preceding them with OPCode “COM” and closing them with “;”.

Example:

```
COM this is a comment and will be ignored during compilation ;
```

**MET, ME2:**

Those instructions allow to modify the “measure” (METronome) during the execution: the default measure is 60 (60 for second).

Syntax:

```
MET t0 v0 t1 v1...tk vk;
```

It means that from t0 time to t1 the measure will be v0, from t1 to t2 v1 and so on. The difference between MET and ME2 is that MET acts on start and end time of notes, while ME2 uses for the duration of notes the measure in force at the beginning of the note.

**FIC:**

This instruction allows us to open a samples file (FICier is the French for file) and to associate it to a number to use in instrument modules (LUM, LUS ...) and in generation of tabulated function (GEN 21).

The format of the file must be AIFF, 16 bits, mono.

Syntax:

```
FIC 0 fileNum filePath;
```

FileNum is the number to which the file will be associated. FilePath is the path to the file to be opened. In this version of music 5 if the application don't find the file where specified, the “composer” will be asked to locate the file to process. Any AIFF 16 bit mono file will be at this point accepted in place of the given one. This should allow to write “special effects” scores (reverberations, echoes, filtering ...) and to apply them to any file.

In the following table are summarised the instructions of music5:

NOT $t$ $NIns$ $dur$ $P5$ $P6$ $P7$ ... $P30$ ;	Plays a note on $NIns$ instrument at $t$ time for $dur$ seconds. $P5$ ... $P30$ values meaning depends on instrument definition
INS 0 $Nins$ ;	Opens $NIns$ instrument definition
END;	Close an instrument definition
GEN $t$ $gType$ $fNum$ $fLen$ $P6,P7$ ...;	Generates $fNum$ function on $fLen$ samples with $gType$ generator at $t$ time (see forward)
SV1 $t$ $n$ $Vn$ $Vn+1$ ... $Vn+k$ ;	Sets the value of pass1 variables from $n$ to $n+k$ respectively to $Vn$ ... $Vn+k$
SV2 $t$ $n$ $Vn$ $Vn+1$ ... $Vn+k$ ;	as $Sv1$ , but in pass 2
SV3 $t$ $n$ $Vn$ $Vn+1$ ... $Vn+k$ ;	as $Sv1$ , but in pass 3
SIA $t$ $n$ $Vn$ $Vn+1$ ... $Vn+k$ ;	as $Sv1$ , but in all passes (control variables)
SEC $t$ ;	Closes a section, whose duration is $t$ .
TER $t$ ;	Closes last section, whose duration is $t$ .
PLF 0 $plfNum$ $P4$ ... $Pk$ ;	Activate $plfNum$ pass1 subroutine, $P4$ ... $Pk$ are parameters specific to the subroutine .
PLS 0 $plsNum$ $P4$ ... $Pk$ ;	Activate $plsNum$ pass2 subroutine, $P4$ ... $Pk$ are parameters specific to the subroutine .
CHN $n$ ;	Sets the number of channels to $n$ (1 or 2) it's equivalent to SIA 0 8 $n$
SAM $n$ ;	Sets the sampling rate to $n$ (equivalent to SIA 0 4 $n$ )
COM this is a comment ;	Allows to insert a comment
MET $t0$ $v0$ $t1$ $v1$ ...; ME2 $t0$ $v0$ $t1$ $v1$ ...;	Allows to modify the "speed" of the execution, making pass2 to recalculate the notes timing .
FIC 0 $fileNum$ $FileName$ ;	Allows to open $FileName$ file and to associate it to $fileNum$ number



### 3.3.2 Function generators

In this chapter are explained in detail the function generators implemented in this version of Music V.

A function generator is an instruction that calculates and store the values of a function in an array. We can say that it “samples” the function in a certain number of points.

There are three important points to remember:

- the length of a function (the number of points on which) is sampled can change (it is given from the `funLength` parameter), a function can have up to 4096 points of length;
- the function is really sampled on `funLength+1` points, as the array index starts from 0, and it is usually important that 0 and `funLength+1` value are identical in order to avoid “clicks” when cycling through the function table (function are cyclically read in order to generate sound from, for example, OSC modules).
- Sometimes it is better to have a normalised function, sometimes not. In this version of music 5 generators 1 and 4 aren’t normalised while 2 can be optionally normalised.

The general syntax of a generator is :

```
GEN t genType funNum funLength P6,P7,P8...
-t : “start” time (moment in which the function is defined).
-genType : type of generator (explained below).
-funNum : number of function generated.
-funLength : length (number of “samples”) of generated function .
-P6,P7,P8... : generator’s parameters (according to generator type).
```

Now we will see each of implemented generators.

#### GEN 1: split line generator

```
GEN t 1 funNum funLength y1 x1 y2 x2... ;
```

The couples  $X_n, Y_n$  represent abscissas and ordinates of the split line segments. First point must have zero abscissa and last one `funLength`.

**GEN 2: sinusoid summation generator**

GEN t 2 funNum funLength S1 S2...Sn C0 C1...Cj N;

The n values S1...Sn represent the amplitudes of elements of the following summation:

$$\text{sum1} = S1*\sin(x)+S2*\sin(2*x)+\dots+S_n*\sin(n*x).$$

The remaining j+1 values (C0...Cj) represent the amplitudes of elements of the following summation:

$$\text{sum2} = C0+C1*\cos(x)+C2*\cos(2x)+\dots+C_j*\cos(j*x).$$

The value of the function generated for the x abscissa is sum1(x)+sum2(x).

N gives both the number of sinus components and the normalisation condition: if it is negative the function won't be normalised, if positive its ordinates will be scaled to fit between -1 and 1;

**GEN 3 : constant delta split line generator.**

GEN t 3 funNum funLength Y1 Y2 ... Yn;

This generator is the same that GEN 1 except that abscissas grow constantly of a step :  $\text{deltaX} = \text{funLength}/n$ . Thus only ordinates values Y1...Yn are indicated.

**GEN 4 : stumps of exponential generator.**

GEN t 4 funNum funLength x1 y1 x2 y2... ;

The values (Xi,Yi) have the same meaning that in GEN 1 but the points are joined with stumps of exponential.

**GEN 5 : fragments of sine generator.**

GEN t 5 funNum funLength F1 P1 I1 J1 F2 P2 I2 J2...

Each set of four numbers Fi Pi li Ji defines the following function :

$$\text{fragment } j = \sin( F_i*x+P_i) \quad [ l_i < x < J_i]$$

**GEN 6 : Envelop generator.**

GEN t 6 funNum funLength E1 Y1 Y2 E2;

This generator generates an envelop to be used with ENV e IEN modules:  
 The first quarter of the function generated grows exponentially from  $2^{-E1}$  to Y1; the second decreases constantly from Y1 to Y2; the third quarter decreases exponentially down to  $2^{-E2}$

**GEN 7 : decreasing exponential generator.**

GEN t 7 funNum funLength V;

The generated function changes according to V value:

- if  $V < 0$  then function decrease from 1 down to  $2^{-V}$
- if  $V = 0$  then function =  $e^{[\log(.008) * (1 - \cos(2\pi * (x/\text{funLength} - .5)))]}$
- if  $V > 0$  then function increase from  $2^{-V}$  up to 1

**GEN 8 : Bell curves generator.**

GEN t 8 funNum funLength E n;

This generator generates “bell” curves with n “bells” and a minimum value of  $0.008^E$ .

The function formula is

$$F(x) = 0.008[E * (1 - \cos 2\pi p x) / 2]$$

where p is proportional to n.

**GEN 10 : polynomial generator.(normalised)**

GEN t 10 funNum funLength d0 d1 d2 ... dn;

The function generated is  $P(x) = d0 + d1 * x + d2 * x^2 + \dots + dn * x^n$ ;

The function is scaled to fit, vertically, between -1 and +1, and results centred on  $\text{funLength}/2$ .

**GEN 11 : Chebychef sums generator.**

GEN t 11 funNum funLength h0 h1 h2 ... hn Vindex;

The generated function is  $P(x) = h_0 * T_0(x) + h_1 * T_1(x) + \dots + h_n T_n(x)$  where  $T_i(x)$  is a Chebychef polynomial;

**GEN 12 : Chebychef sums generator with automatical phase alternation and continuous component annulment;**

GEN t 12 funNum funLength h0 h1 h2 ... hn Vindex;

The generated function is  $P(x) = h_0 * T_0(x) - h_1 * T_1(x) + h_2 T_n(x) - h_3 * T_1(x) \dots$  where  $T_i(x)$  is a Chebychef polynomial;

**GEN 13 : Chebychef sums generator with automatical phase alternation, continuous component annulment and null output with null input;**

GEN t 13 funNum funLength h0 h1 h2 ... hn Vindex;

This function differs from GEN 12 in the fact that it eliminates the continuous component that can appear in the origin using GEN 12, generating a “click”.

**GEN 14 and 15: Chebychef sums generator starting from a precalculated function.**

GEN t 14 funNum funLength startFun HarmonicNum Vindex

GEN t 15 funNum funLength startFun HarmonicNum Vindex

They allow us to use a predefined function as startpoint for the distortion calculation. GEN 15 differs from 14 in that it eliminates the continue component in the origin. The stored function and the generated one MUST be of the same length.

**GEN 21 : sample files reader generator.**

GEN t 12 funNum funLength Nf Nb Pb;

Nf : number of file (previously opened with FIC instruction)  
 Nb: Number of block (64 samples)  
 Pb: Position inside block (0-64)

This generator reads from Nf AIFF mono 16 bit file. This generator reads funLength samples starting from a file position given from the formula  $\text{position} = \text{Nb} * 64 + \text{Pb}$ . This strange convention came from original version of music 5, written in Turbo pascal, that reads from “untyped files” blocks of 128 bytes (64 integer).

**GEN 30 : sum of functions.**

**GEN 31 : multiplication of functions.**

GEN t 3x funNum funLength f1 f2;

The function is the result of the sum, or the multiplication, of f1 and f2.

### 3.4 Instrument definition

The definition of an instrument is always opened from a “INS t Nins;” instruction, and is closed from a “END;” instruction.

In the “body” of the definition can be present two types of instruction : modules definition and conversion instructions.

The various modules who perform operations necessary to generate the sound are joined from “B” blocks: a B-block is a samples block (256 samples in this version of music 5) where a module can store the result of its calculations and from which another block can take the values to use as inputs. The use of blocks speeds up the programs that processes a block at a time instead of a single sample.

The modules can take their inputs both from B-blocks and from P-fields of NOT instructions, and from Variables of Pass 3 (SV3) and from tabulated Functions (GEN).

For example:

INS 0 1;	COM beginning of instrument 1 definition;
OSC P5 V2 B3 F1 P7;	COM module OSC : reads F1 function (see forward);
OUT B3 B1;	COM OUT module : takes block 3 and puts it out on 1;
END;	COM END of instrument 1;
SV3 0 2 10;	COM put 10 in variable 2 (speed of function reading)
GEN 0 2 512 1 1;	COM tabulate $1 * \sin(x)$ function on 512 points.

NOT 0 1 1 10;                   COM play instrument 1 at 10 amplitude from 0 for 1 second;  
 TER 1;                            COM end of the execution;

NOTE : B1 and B2 blocks are reserved for use by OUT and STR;

The meaning and the function of various modules will be explained in next chapter. The other type of instruction that can be present in an instrument definition is the conversion instruction: "CNV".

It is used in order to convert the value of a P-field allowing the composer to indicate, for example, the amplitude of a function in dB instead of in absolute abscissa, the frequency in Hz instead of in ticks, or to consider the duration as a time.

Actually several conversions are implemented:

- the four operations : + - \* /  
 example: CNV P5=P5/10; (P5=800 -> P5=80)
- conversion of frequency: HTZ();  
 example: CNV P5=Hz(P5). Allows us to consider the P5 parameter as a frequency
- conversion of duration: DUR()  
 example: CNV P5=DUR(P5); Allows to consider p5 as a time.
- the mathematical functions: SIN(),COS(),LOG(),EXP(),SQR().
- the envelop conversion: CEN()  
 allows to calculate the phase increments used from the ENV module (see next paragraph) starting from three times in three successive parameters.

The conversion function can have as arguments P-fields, variables of pass 2 (Gxx), integer or floating numbers, of copies of P values (W parameters: copies of P-fields before conversion).

### 3.5 The Modules

In this chapter we will see in detail modules of music 5 and some examples of their usage in building instruments. The modules have been divided in 6 groups,

according to their function, and each group have been associated to a button in the instrument window, in order to make easier finding them.

The division is respected in this chapter and all modules of a group are described in the same paragraph.

### 3.5.1 Inputs and outputs (first button)



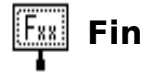
**Pin**



**Bin**



**Vin**



**Fin**

Pin, Bin, Vin, Fin as a metter of fact aren't music V modules, we could call them pseudo-modules. They have been introduced in the graphical environment in order to allow to represent the P,B,V,F values that appear in the "alphabetical" representation of the modules. In order to associate a value to one of those modules (and to the corresponding field in the script) it is necessary to select the module in the instrument window, to choose "Mod info" item from the Edit menu, and to write its value in the field of the "Mod info" dialog that appears.



**OUT I1 Out**

I1: input signal;

Out: Output Block (B1 by default);

OUT is the monophonic output module. It usually precede the END instruction (except in pseudo instruments). I1 is usually a block (Bxx), Out is by default B1 and can be omitted.

If the ouput have been set to stereo using the CHN 2 instruction or in the score an STR, CHA or CHB instruction is used, the output file will be stereo and OUT will put the same value on both channels.



**STR I1 I2 Out**

I1: input signal for channel A;

I2: input signal for channel B;

Out: Output Block (B1 by default);

STR is the stereo output module. It usually precedes the END instruction (except in pseudo instruments). I1 and I2 are usually blocks (Bxx), Out is by default B1 and can be omitted.

If the output have been set to mono using the CHN 1 instruction the samples of channel A and B will be mixed on the output mono channel (the output file will be half size, the computation time will be less, but not half)

### 3.5.2 The oscillators (second button)



**OSC A Ph Out F T**



**IOS A Ph Out F T**

- A: Amplitude
- Ph: Phase increment
- Out: Output Block
- F: Function used
- T: Phase storage (P or V parameter)

Oscillator (OSC not interpolated and IOS interpolated)

OSC (and IOS) read the tabulated function F at a speed (frequency) depending on Ph parameter (step in reading the table), multiply the value by A (amplitude) and puts the result in the Out output Block. The program stores the value of the “phase” (position inside the function table) in P or V field (Parameter or Variable) optionally indicated in T (if not specified the program will assign a P storage for it).

IOS is the interpolated version of OSC that is: OSC takes the “nearest” sample of F truncating it whereas IOS calculates the exact value interpolating precedent and following samples.

On one side OSC is faster, but may cause distortion, on the other IOS is more exact, but much slower.

Example:

```
COM -----;
COM sinusoid, amplitude 8000;
COM frequency 220 Hz;
```



```

COM duration 3 seconds;
COM -----;
INS 0 1;
CNV P6 = HTZ(P6);
OSC P5 P6 B3 F1 P30;
OUT B3 B1;
END;
COM -----;
GEN 0 2 1 512 1 1; COM sinusoid;
COM -----;
NOT 0 1 3 8000 220;
TER 3;

```



**OS1 A Ph Out F T**



**IO1 A Ph Out F T**

- A: Amplitude (P or V parameter)
- Ph: Phase increment (P or V parameter)
- Out: Output Block
- F: Function used
- T: Phase storage (P or V parameter)

OS1 and IO1 are identical to OSC and IOS except that A and Ph parameters must be a P parameter or a V variable. The algorithm they use is adapted to this specification and thus is faster. If a B block is passed as a A or Ph parameter, only the first value of the block will be taken.



**OS2 A Ph Out F S T**



**IO2 A Ph Out F S T**

- A: Amplitude (P or V parameter)
- Ph: Phase increment (P or V parameter)

Out: Output Block  
 F: Function used  
 S: Extra input  
 T: Phase storage (P or V parameter)

OS2 (IO2) is equivalent to an OSC (IOS) module followed by a AD2 adder who sums the OSC (IOS) output and the S input.

Example:

```

INS O 1;
CNV P6 = HTZ(P6);
CNV P7 = HTZ(P7);
CNV P8 = HTZ(P8);
OS2 P7 P8 B3 F1 P6 P30;
OSC P5 B3 B4 F2 P29;
OUT B4 B1;
END;
COM -----;
GEN 0 2 1 512 1 1;
GEN 0 2 2 512 10 9 8 7 6 5 4 7;
COM -----;
NOT 0 1 3 8000 220 22 6;
TER 3;

```

vibrato on a 220 Hz note: deep 22Hz (10%), speed of vibrato 6 Hz.



**OS3 A Ph Out F Ph1 T**



**IO3 A Ph Out F Ph1 T**

A: Amplitude (P or V parameter)  
 Ph: Phase increment (P or V parameter)  
 Out: Output Block  
 F: Function used  
 Ph1: Phase extra  
 T: Phase storage (P or V parameter)

Instead of scanning the F table only with a Ph step OS3 and IO3 use an additional phase given from Ph1.

It can be used to produce a FM synthesis in a simple and neat way (Ph can be the carrier and Ph1 the modulant or vice-versa) or when a very exact dephasing are needed.

Examples:

- Exact phasing:

```

COM -----;
INS 0 1;
IOS P7 P10 B3 F1 P30;
IO3 P5 B3 B4 F2 P6 P29;
OUT B4 B1;
END;
COM -----;
NOT 0 1 3 8000 100 0;
NOT 0 1 3 8000 100 .5;
NOT 0 1 3 8000 100 1;
TER 3;

```

The first oscillator defines a phase decreasing from P7 value down to 0. Additioning more sounds we obtain the equivalent of a variable delay with very exact dephasing.

- FM instrument;

```

COM -----;
INS 0 1;
CNV P6= HTZ(P6); COM carrier ;
CNV P7= HTZ(P7); COM modulant ;
IOS P8 P10 B3 F2 P30;
IOS B3 P7 B4 F1 P29;
IO3 P5 P6 B5 F1 B4 P29;
OUT B5 B1;
END;
COM -----;
GEN 0 2 1 512 1 1; COM sinus;
GEN 0 1 2 100 0 0 1 10 0 100;
COM -----;
NOT 0 1 3 8000 220 110 2;
TER 3;

```

Function F2 is a modulation envelop, F1 is a sinusoid.



**ENV A F Out I1 I2 I3 T**



**IEN A F Out I1 I2 I3 T**

- A: Amplitude
- F: Function used
- Out: Output block
- I1: increment of phase on the first quarter of F (attack)
- I2: increment of phase on the second quarter of F (sustain)
- I3: increment of phase on the third quarter of F (release)
- T: Phase storage (P or V parameter)

ENV and IEN are envelop modules (not interpolated and interpolated).  
 ENV (IEN) scan once the F function, using the Ph1, Ph2, Ph3 step for each quarter of it.

Example:

```
SAM 22254;
COM -----;
INS 0 1;
CNV P6 = HTZ(P6);
CNV P7 = DUR(P7)/4;
CNV P8 = DUR(P8)/4;
CNV P9 = DUR(P9)/4;
ENV P5 F1 B3 P7 P8 P9 P30;
OSC B3 P6 B3 F2 P29;
OUT B3;
END;
COM -----;
GEN 0 1 1 1000 0 0 1 100 .8 250 .6 500 0 750 0 1000;
GEN 0 2 2 1000 1 .9 .7 .3 4;
COM -----;
NOT 0 1 2 10000 261 .2 1 .8;
TER 5;
```

Note: the sum of times must be equal to note duration. This can be obtained automatically preceding other conversions with  $CNV P8=P4-P7-P9$ .

### 3.5.3 Table readers, random generators, file readers (third button)



**FON A X Out F**



**IFO A X Out F**

- A: Amplitude
- X: point inside the function (between 0 and 1)
- Out: Output Block
- F: Function used

FON and IFO read the tabulated function F in the X point and return the result in Out according to the formula:  $Out = A * F(X)$ .

The F abscissa is normalised between 0 and 1 because of the different length of functions.

Example:

```
INS 0 1;
CNV P10 = DUR(P4);
IOS P6 P10 B3 F1 P30;
FON P5 B3 B4 F2;
OUT B4 B1;
END;
COM -----;
GEN 0 1 1 512 0 0 1 512;
GEN 0 1 2 512 220 0 .1;
COM -----;
NOT 0 1 5 8000 1;
TER 1;
```



**RAH A f Out T1 T2 K**

- A: Amplitude
- f: pseudo-frequency ( frequency of change )
- Out: output Block

T1: temporary storage  
 T2: phase storage  
 K: correlation factor

RAH is a random and hold filter: it generates a random value between -1 and +1 and maintains it until a new value is generated.



**RAN A f Out T1 T2 Ph K**

A: Amplitude  
 f: pseudo-frequency ( frequency of change )  
 Out: output Block  
 T1: Value of the pseudo-phase  
 T2: temporary storage  
 Ph: phase storage  
 K: correlation factor

RAN is a random filter. The difference between RAH and RAN is that RAH output is constant until it generate a new random value, while RAN interpolates linearly two succesives values.

Example:

```
INS 0 1;
CNV P6 = HTZ(P6);
RAN P5 P6 B3 F1 P28 P29 P30 ;
OUT B3 B1 ;
END;NOT 0 1 5 8000 220;
TER 2;
```

In this example RAN is used as sound generator.



**LUM A V Out I1 I2 I3**

A: Amplitude  
 V: Reading velocity

Out: output Block  
 Nf: Number of file (assigned from FIC instruction)  
 Nb: Number of block (64 samples) of file;  
 Pb: Position inside block (0-64)

LUM allows to read samples contained in a AIFF 16 bit mono file both forward ( $V>0$ ) and backward ( $V<0$ ) starting from a sample whose position is given from the formula position =  $Nb * 64 + Pb$  (see FIC description in paragraph 3.3).

Example:

```
SAM 25600;
FIC 0 1 My_AIFF_file_name;
INS 0 1;
LUM P5 P6 B3 P7 P8 P9;
OUT B3;
END;
NOT 0 1 3 .5 2 1 100 0;
TER 5;
```

The file My\_AIFF\_file\_name is associated to number 1, and read by the LUM instruction at double speed and half amplitude, starting from the sample at position:  $P8*64+P9=64000$  (0.25 sec).



**LDI A Out I1 I2 I3**

A: Amplitude  
 Out: output Block for channel A  
 Nf: Number of file (assigned from FIC instruction)  
 Nb: Number of block (64 samples) of file;  
 Pb: Position inside block (0-64)

LDI is equivalent to LUM A 1 Out Nn Nb Pb: it reads the given file forward with a velocity of 1. It is about twice faster than the equivalent LUM instruction.

### 3.5.4 The arithmetical functions (fourth button)



**MLT I1 I2 Out**

I1: first input  
 I2: second input  
 Out: Output Block ( $I1 \cdot I2$ )

MLT multiplies I1 I2 and puts the result in Out



**DIV I1 I2 Out**

I1: first input  
 I2: second input  
 Out: Output Block ( $I1 / I2$ )

DIV divides I1 by I2 and puts the result in Out. If I2 is zero, then it returns zero (to avoid a divide by zero error)



**SUB I1 I2 Out**

I1: first input  
 I2: second input  
 Out: Output Block ( $I1 - I2$ )

SUB subtracts I2 from I1 and puts the result in Out.



**AD2 I1 I2 Out**



**AD3 I1 I2 I3 Out**





**AD4 I1 I2 I3 I4 Out**

- I1: first input
- I2: second input
- I3: third input
- I4: fourth input
- Out: Output Block (I1+I2 [+...])

ADx simply adds its inputs and puts the result in Out input .

### 3.5.5 Filters, delays, distortions (fifth button)



**SLF I1 Out f Q T1 T2**



**SHF I1 Out f Q T1 T2**



**SBF I1 Out f Q T1 T2**



**SNF I1 Out f Q T1 T2**

- I1: Input signal
- Out: Output block
- f: central frequency/sampling rate
- Q: overtension factor
- T1: Temporal parameter
- T2: Temporal parameter

They are then numeric version of “variable state filters” (two poles, two zeros). Respectively they are: SLF, low pass, SHF, high pass, SBF, band pass, SNF, notch filter.

Their characteristic is that f and Q are independent.

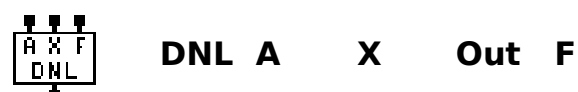
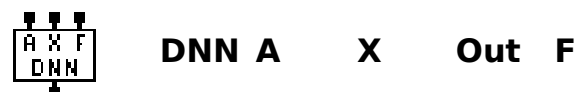


- I1: Input signal
- D: Delay
- Out: output block
- F: Function “delay”, its length must be greater than the delay
- G: Amplitude factor
- T: Table position storage.

DEL and DEI are delay lines (not interpolated and interpolated). They return the input value  $D$  samples “later” multiplied by  $G$  factor. In formula  $DEL(k) = G * I1(k-D)$ , where  $k$  represents the number of current sample.

The function  $F$  isn't used as usual: it acts as a “storage” for the values of  $I1$  being delayed. Such functions are usually initialised with silence and they obviously can't be used at the same time on more DEL, or DEI... instructions.

The most evident use of those modules is to create echoes.



- A: Amplitude
- X: point inside the function (between -1 and 1)
- Out: Output Block
- F: Distortion function used

DNN and DNL are non-linear distortion modules: they return the result in  $Out$  according to the formula:  $Out = A * F(X)$ .

The abscissa of  $F$  is normalised between -1 and 1 (bipolar function)

DNL is the interpolated version of DNN.

Example

```

INS 0 1;
CNV P10 = DUR (P4);
CNV P6 = HTZ(P6);
IOS P7 P10 B3 F1 P30;
IOS B3 P6 B3 F2 P29;
DNL P5 B3 B4 F3;
OUT B4 B1;
END;
GEN 0 1 1 512 0 0 1 100 0 512;
GEN 0 2 1 512 1 1;
GEN 0 13 3 512 0 .5 .4 .3 .2 .1 1;
COM -----;
NOT 0 1 3 8000 150 1;
TER 3;

```

### 3.5.6 Reverberators (sixth button)



**RVA I1 D Out F G T**



**RIA I1 D Out F G T**

I1: Input signal

D: Delay

Out: output block

F: Function "delay", its length must be greater than the delay

G: Amplitude factor

T: Table position storage.

RVA and RIA are reverberation modules (not interpolated and interpolated).

They are very similar to DEL and DEI, except that their output consists in the original signal added to an echo delayed of D samples and scaled of a G factor.



**RVB I1 D Out F G T**



**RIB I1 D Out F G T**

- I1: Input signal
- D: Delay
- Out: output block
- F: Function "delay", its length must be greater than the delay
- G: Amplitude factor
- T: Table position storage.

They are recursive filters that produce multiple echoes with amplitudes 1,  $g$ ,  $g^2$ ,  $g^3$ ,  $g^4$ ..., where  $g$  is the gain ( $G$ ).



**RVC I1 D Out F G T**



**RIC I1 D Out F G T**

- I1: Input signal
- D: Delay
- Out: output block
- F: Function "delay", its length must be greater than the delay
- G: Amplitude factor
- T: Table position storage.

RVC and RIC calculate a signal that is the input, changed of sign, followed by subsequent echoes of decreasing amplitude.



**RVD I1 D Out F G ET**



**RID I1 D Out F G ET**

- I1: Input signal
- D: Delay
- Out: output block
- F: Function "delay", its length must be greater than the delay

- G1: Gain of first impulse
- G2: Gain of echoes
- T: Table position storage.

They are intermediate between RVB and RVC: depending on G1 value they act more as RVB or RVC:

If  $G1 = 1$  then they act like RVC, if  $G1 = 0$  they act as RVB.



**RVE I1 D Out F G ET**



**RIE I1 D Out F G ET**

- I1: Input signal
- D: Delay
- Out: output block
- F: Function "delay", its length must be greater than the delay
- G1: Gain of low-pass filter ( $0 < G1 < 1$ )
- G2: Gain of feedback
- T: Table position storage.

Given an input signal they calculate an output for addition of subsequent echoes, filtered in relation to preceding one.

## 4. Acknowledgements and references.

### 4.1 Release notes

- The score compilation and execution code is a port (with minor changes) from original PC version by professor Daniel Arfib at MArseille University.
- A relevant part of chapter 3 (and all examples contained) have been taken from the original user manual of PC music 5 version written by professor Daniel Arfib at Marseille University, and part comes from music5 manual written by professors De Poli and Vidolin at Padova University.
- This application was written in Think Pascag 4.0.1 on a Macintosh SE (sigh) running system 7 for the first 50% and on a Macintosh II cx for the rest. It should run without many problems on any macintosh running system 6.0.7 and up.

### 4.2 Acknowledgements

I have to thank many people who, in various manners, helped me to lead this work to the end.

First of all my professors De Poli and Vidolin for their advice and help.

Professor Pupolin, who let me use the mac IIcx in his laboratory.

My friends:

Beppe, who lended me Inside Macintosh for some months

Roberto, for his precious advice and help.

And all the people who did bear my bad mood and my requests of help.

### 4.3 References

#### [1] **Music V :**

Max V. Mathews and others *The Technology of computer music* (The M.I.T. Press 1969)

Music V explained from its creator.

#### [2] **Macintosh user interface and its terminology**

Refer to the "Macintosh® System Software User's Guide" manual.

**I dedicate this work to my wife Laura and my daughter Marianna for the patience they had , and for the support they gave me.**