dream

an ATMEL Company

SAM9407
PROSOUND

SAM9407 & SAM9503
Professional Sound Tools User Manual
& Sound Files Format

Overview

This document is shared in 2 parts:

- SOUND EDITORS USER MANUALS
  Instrument editor
  Sound bank editor
  PCM Loop Point Adapter
- SOUND FILES FORMAT

.94L	Sound list
.94I	Instrument definition
.94K	Sound bank
.94B	Binary sound bank
.SMP	Sample Vision sample format
.WAV	Microsoft sample format
.RMI	Microsoft Midi File format

Chapter 1.1

Instrument Editor

Overview

94WINST is an instrument editor for SAM9407 products.
This Windows editor is WYSIWYG, and each edition can immediately be heard with the 9407 sound card.

For all parameters in the editor, there are two ways to modify value:
- Double-Click on the value, and enter new value with the keyboard,
- Click right/left mouse button over the value increases/decreases it by one.

94WINST download the instrument in SAM9407 board. Downloading procedure uses the sound bank concept described in Chapter II. 94WINST creates a sound bank called EDITOR and reserves all memory in 9407 board. At the end of the session, 94WINST removes the EDITOR sound bank and downloads the default power up sound bank.

Create an instrument

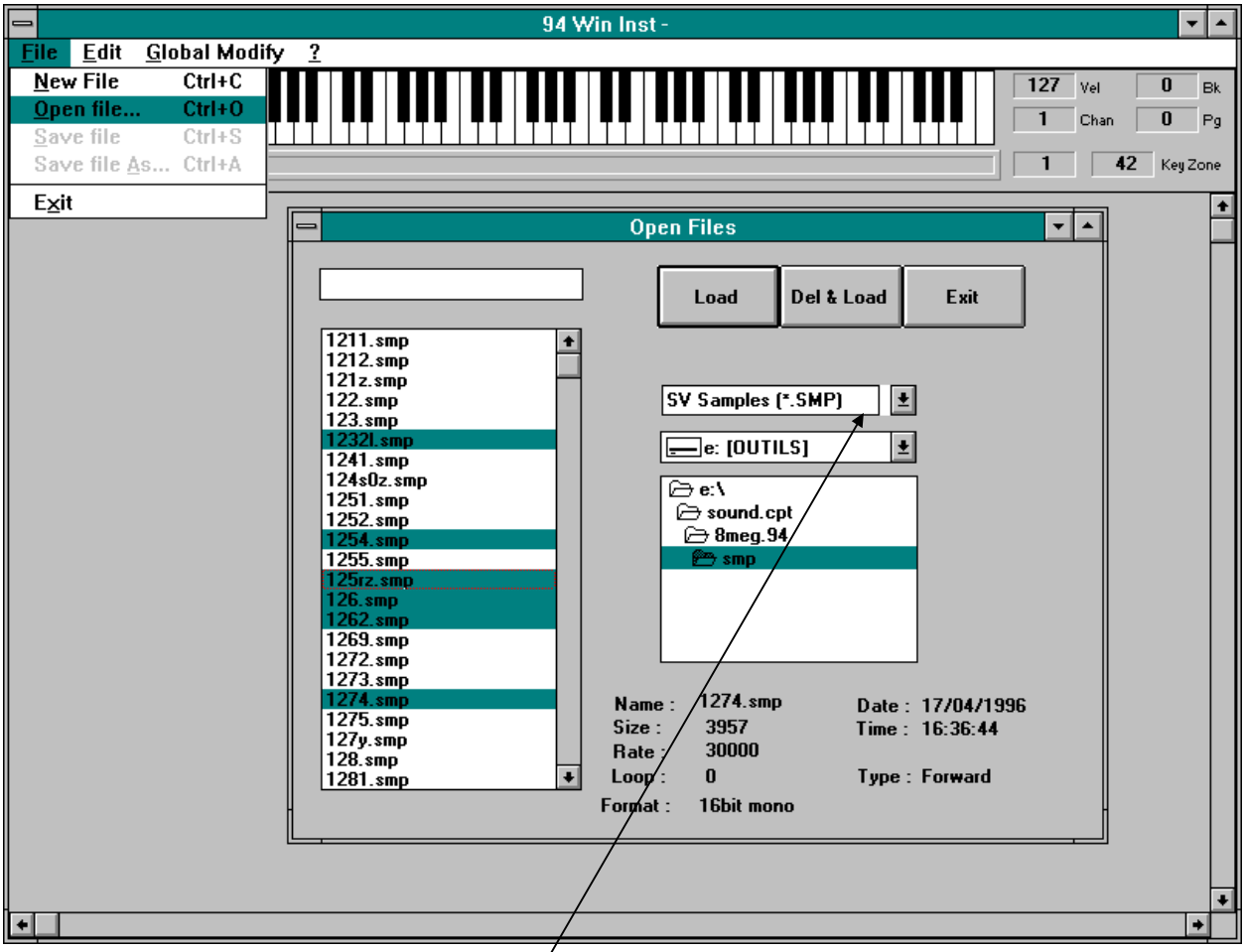
An instrument is made of splits. A split is a group of notes (with a defined velocity range) that have the same parameters.
The editor can define splits and edit parameters of one or several splits at the same time.
The sound generator can be PCM type, FM type or noise type.
In PCM case, PCM should be loaded before edition.

Then the edition steps to create an instrument are:
- Load all PCMs
- Define splits
- Modify parameters of splits
- Save the instrument

Load PCM

94WINST supports Windows Wave format (.wav) and SampleVision format (.smp).

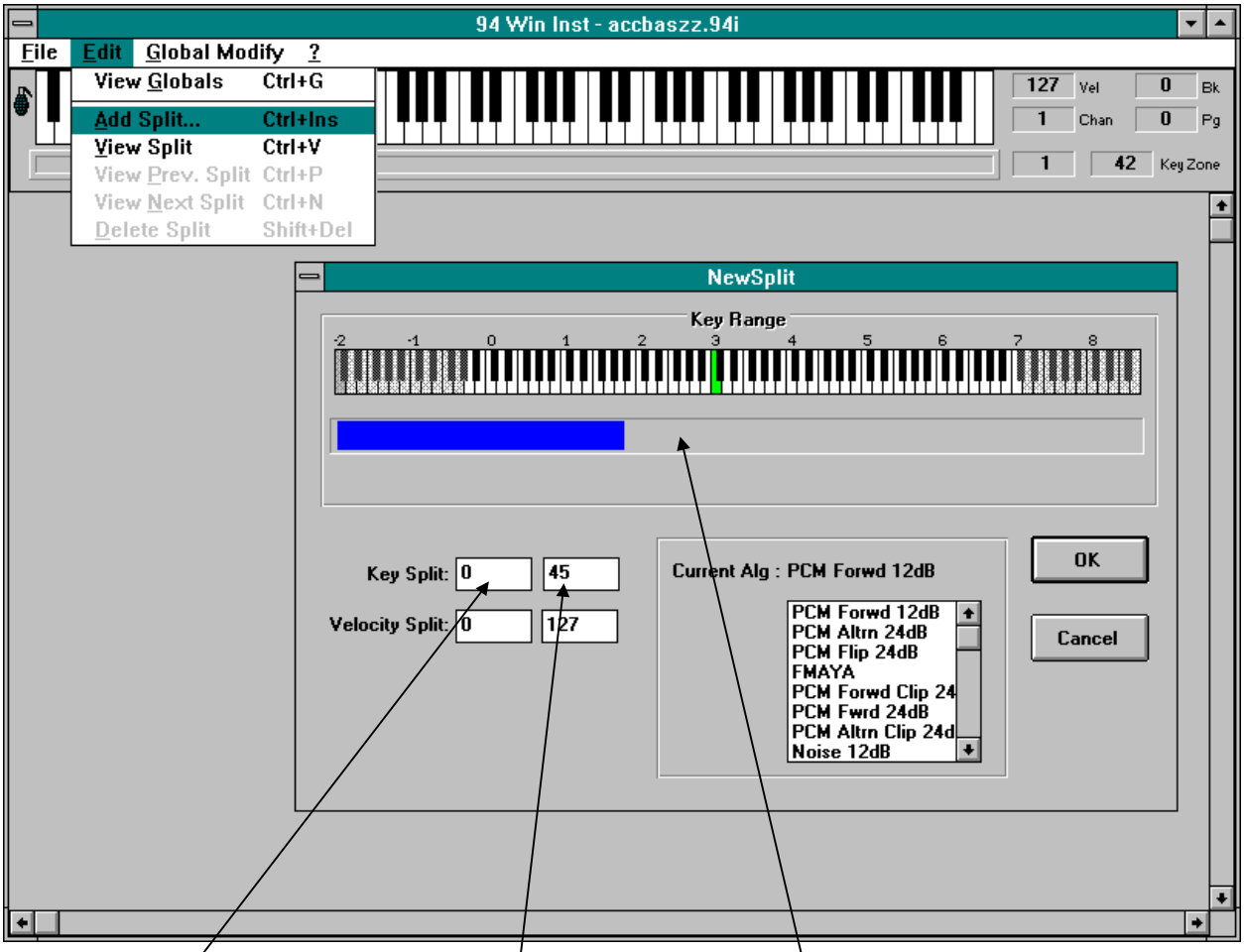
- Select the *File menu*
- Click on *Open File*
- *The Open Files* window appears
- Select format in the *File Format Combo Box*
- Select drive & directory
- Select sample files in *list box* (multiselection is allowed)
- Click on *Load* button



File Format Combo Box

Define split

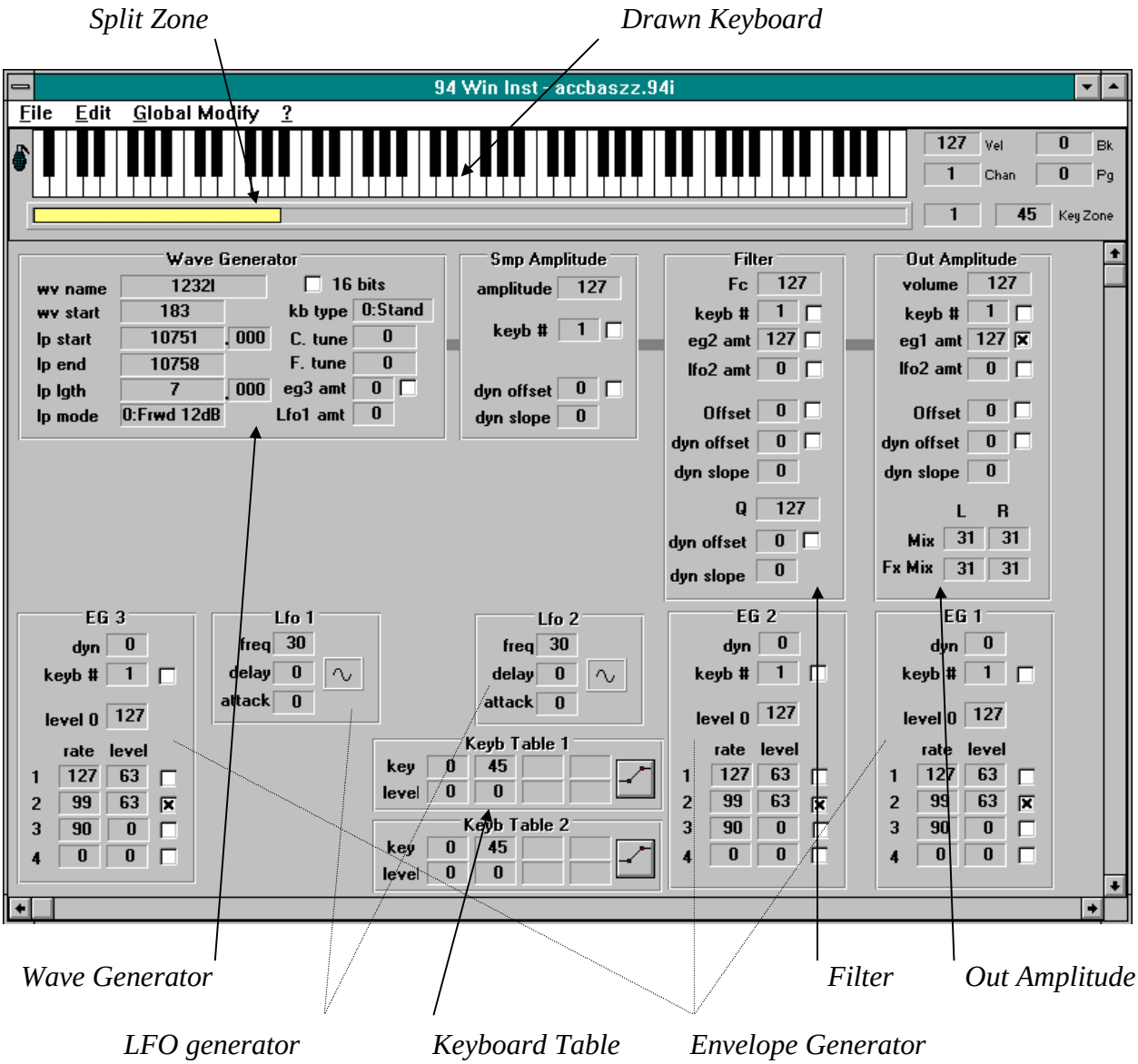
- Select the *Edit* menu
- Click on *Add Split*.
- The *NewSplit* window appears
- Set *Key Split* limits (E.G. 45 for end) Click with the left button in the *Range Bar* enters low key split, the right button enters high key split.
- Set *Velocity Split* limits if required in case of velocity split.
- Select algorithm in list box
- Click on *OK* button



Start Key Split *End Key Split* *Range Bar*

Modify parameters

After define split procedure the screen looks like this:



KEYBOARD

Click on the keyboard notes to play the instrument.

On the right of the keyboard

Vel: set the velocity of the keyboard note clicked by mouse

Chan: set the midi out channel

Under the drawn keyboard, the Split Zone indicates which split it is the current one. To select a split, just click on the right part of Split Zone: the current one is in yellow (others one are in grey).

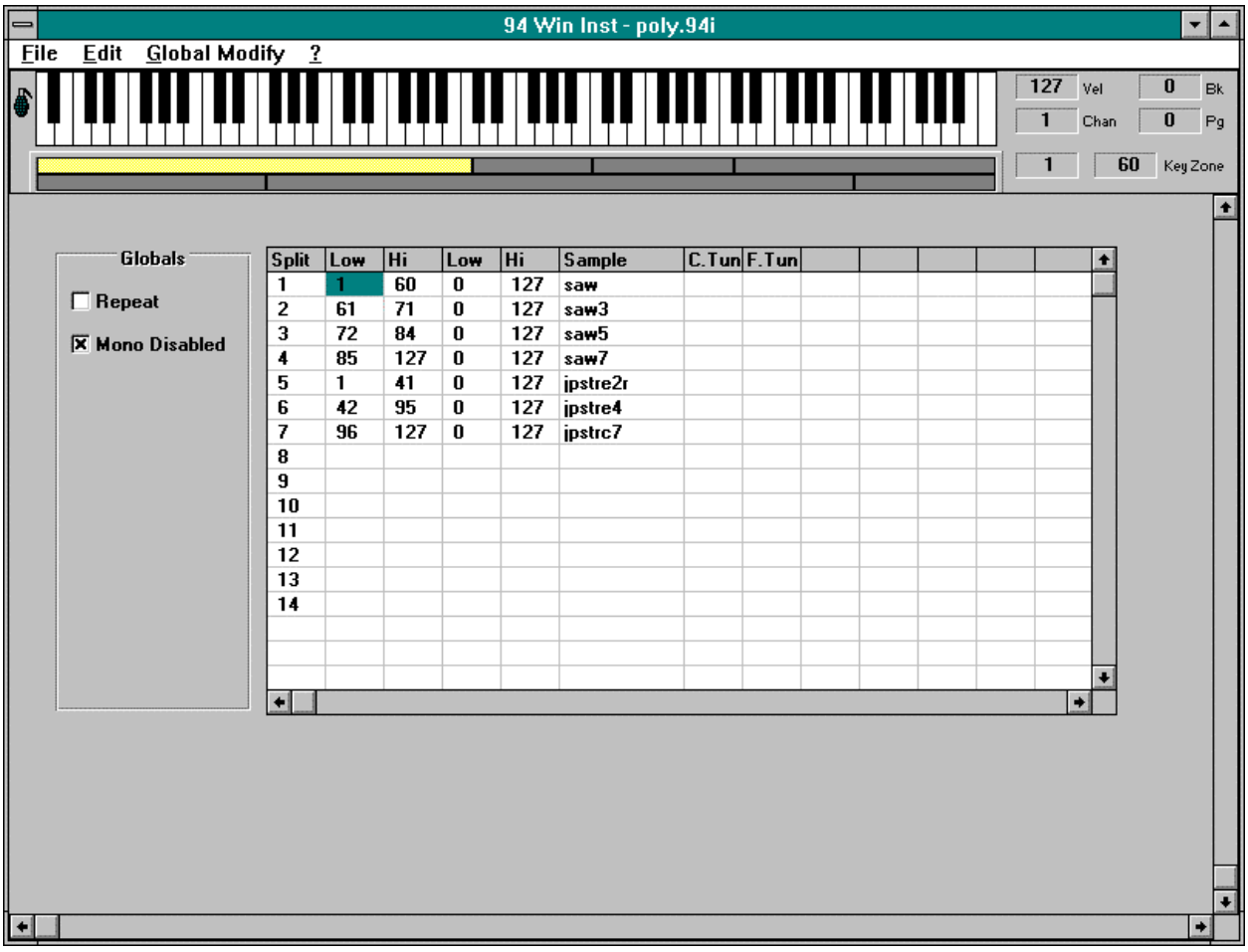
You can hear the sound of your instrument by clicking on the drawn keyboard. All modifications are real time.

In a split definition, there are several groups of parameters:

- o Wave Generator: takes care of what type of algorithm is used, which PCM with its sample pointers (Start, Loop Start, Loop End), tune, EG on pitch, LFO on pitch.
- o Filter: can be dynamic, have EG amount, resonance, LFO.
- o Out Amplitude: dynamic, EG amount, LFO, volume and Left&Right MIX.
- o Envelop Generator: a 4 segments envelop, with sustain point.
- o Keyboard Table: used to have keyboard modulation: apply on Filter, Out Amplitude, EG, pitch.
- o LFO generator: 2 independent oscillators: apply on Filter, Out Amplitude, pitch.

View all splits

Just select *Edit* in menu and click on *View Global*.



A split can be selected by clicking on its Split Zone part, at any time. Or commands View Prev. Split and View Next Split (in the Edit menu) to browse through the splits.

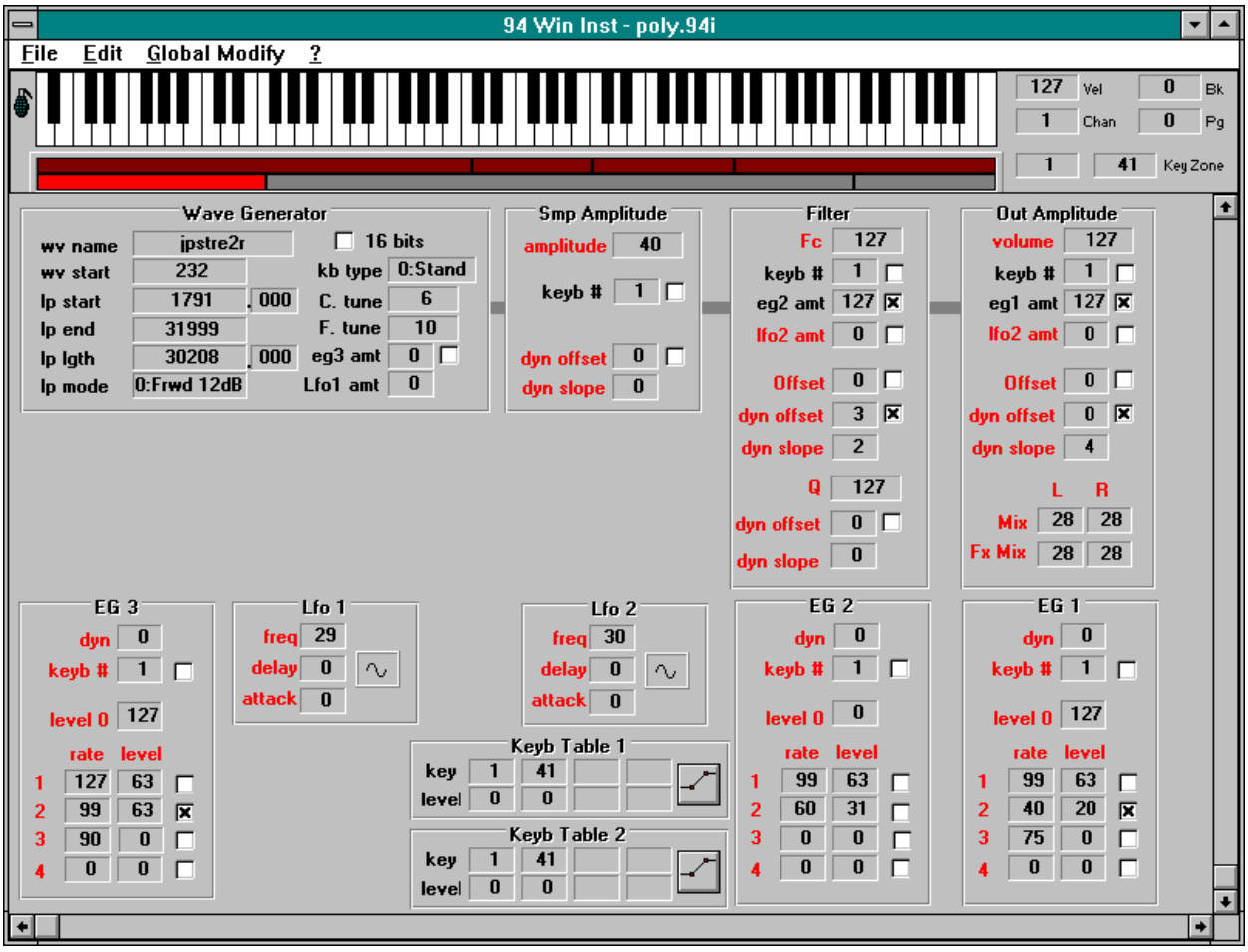
Modify parameters on several splits at the same time

Some parameters can be edited on several split at the same time: Volume, MIX, Cut-Off frequency, dynamic parameters.

To select splits, choose in menu *Global Modify*

- *Select All Splits*: ...,
- *Select Till current*: select from first split to current split,
- *Select From current*: select from current split to last split.
- *Deselect*: to deselect.

Selected splits are turned into red in the *Split Zone*. The parameters displayed in red are the only ones that can be globally changed for the selected splits.



Load & Save instrument

To load an instrument, select the *File* menu and click on *Open Files*. The file format for an instrument is Dream Tone (*.94i).

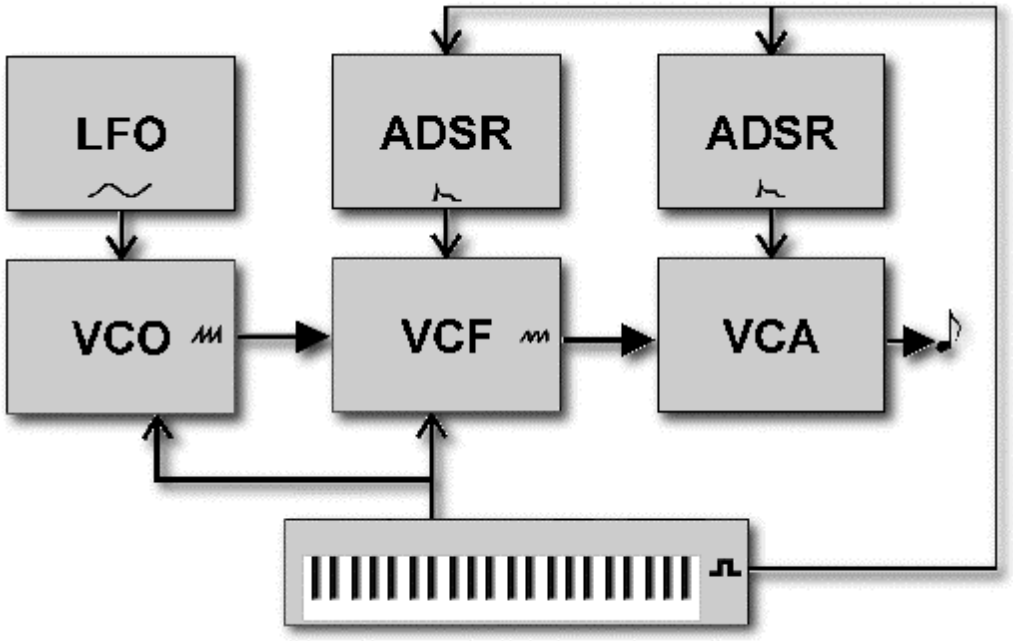
To save an instrument, select the *File* menu and click on *Save* or *Save As*.

Appendix: 9407 General concepts

The 9407 uses different algorithms for sound generation. Most of them use samples as sound sources.

The Analog Model

At the beginning of sound synthesis, the sound was generated by oscillators or noise generators, modified by filters and amplifiers modulated by envelope generators and low frequency oscillators.



Typical Analog Synthesis model

VCO = Voltage Controlled Oscillator (generating a saw wave or a square wave)
VCF = Voltage Controlled Filter (filtering the wave generated by the VCO)
VCA = Voltage Control Amplifier (shaping the amplitude envelope with the associated ADSR)
ADSR = Envelope Generator (generating envelopes to control VCF cutoff freqency, or VCA amplitude)
LFO = Low Frequency Oscillator (generating low freq saw or sine for vibrato, tremolo....)

The Sampling Algorithms

The Sam 9407 sampling algorithms use the same kind of architecture:
The oscillator (VCO) is replaced by a sample player called « **Wave Generator** »
This block is followed by an attenuator called « **Sample Amplitude** »
The « **Filter** » (VCF) with cutoff frequency control and Q control
The amplifier (VCA) is called « **Output Amplitude** »
The Envelope Generators (ADSR) are called **EG1**,**EG2** and **EG3**
The Low Frequency Oscillators (LFO) are **lfo1** and **lfo2**
The Keyboard tables are **keyb table 1** and **keyb table 2**.

The Wave Generator

the different fields in this block are:
wv name: displaying the Wave Name. Click here to select one of the sample that you have already loaded.
wv start: wave start. Default is zero. Can be changed to skip the start of sample.
lp start: loop start. Default is value found in .wav or .smp file. Can be changed if not correct. Can be fractionnal.
lp end: loop end . Default is value found in .wav or .smp file. Can be changed if not correct. (Setting this value beyond the last point of the sample will conduct to unpredictable results)
lp lgth: loop length. Is a function of loop start and loop end. Changing this value will change the loop start and leave the loop end unchanged.
lp mode:loop mode. Allows selection between 3 loop modes: Forward (standard), Alternate, and Flip. You can also select the type of filter which you want to use.
16 bits: Sample resolution. 16 bits if checked. 8 bits otherwise.
kb type:keyboard control type. Can be Standard (½ tone per key), Kbd1 (using keyboard table 1) or fixed
C.tune: Coarse Tune. Each step corresponds to ½ tone.
F.tune: Fine Tune. Each step corresponds to 1/127 halt tone
eg3 amt:EG3 amount. Amount of pitch modulation by Envelope Generator 3
Lfo1 amt:LFO1 amount. Amount of pitch modulation by LFO 1.

In this block you select your sample, you may change its loop points, you can tune your sample and use a frequency modulation.

The Sample Amplitude block

the different fields of this block are:
amplitude: amplitude of the sample at the filter input. You should optimize this amplitude to have the best Signal to Noise Ratio, without clipping in the filter.
kbd # : Keyboard table number. You may modulate the amplitude at the filter input by the key played to prevent the (resonant) filter from clipping. (To modulate amplitude for split level smoothing use kbd modulation on output amplitude)
dyn offset: Dynamic Offset
dyn slope: Dynamic Slope: used if the filter clips at high keyboard dynamics

The Filter

The model of filter used is Low pass Filter .
The different fields of this block are:
Fc: Cutoff Frequency. This can be used to make differences between splits smoother. It can be also used for special effects (with or without Q) on pads ...
kbd # : Keyboard table number. You may modulate the Fc by the key played to smooth timbre differences between splits or to make a keyboard tracking.
eg2 amt: EG2 amount: not implemented. Just check the box if you want to modulate the cutoff frequency by envelope 2.
lfo2 amt: LFO2 amount. To modulate the timbre by a low frequency oscillator. Mostly for special effects.
offset: filter offet (used with dynamic offset and dynamic slope)
dyn offset: Dynamic Offset
dyn slope: Dynamic Slope: used with Dynamic Offset to modulate timbre by keyboard velocity.

Q: resonance of the filter. Can make the filter clip. In this case reduce the sample amplitude at the filter input.
dyn offset: Dynamic Offset
dyn slope: Dynamic Slope: used with Dynamic Offset to modulate timbre by keyboard velocity (for special effects)

The Output Amplitude block

The different fields of this block are:
Amplitude: output amplitude. Use this filed to balance the different split volumes and the different instrument volumes.
kbd # : Keyboard table number. You may modulate the amplitude by the key played (for split level smoothing)
eg1 amt: EG1 amount: not implemented. This box is checked : the amplitude is always modulated by the envelope 1.
lfo2 amt: LFO2 amount. To modulate the amplitude by a low frequency oscillator.
offset: amplifier offet (used with dynamic offset and dynamic slope)
dyn offset: Dynamic Offset
dyn slope: Dynamic Slope: used with Dynamic Offset to modulate amplitude by keyboard velocity.

L/R: Left right
Mix : Right left mix: Used to pan the splits. (usually used on drums only)
FX Mix : Chorus and Reverb sends.Automaticly set to (Leftmix+ rightmix)/2 can be overridden.

The Envelope Generators EG

The envelope generator is a multiple segment EG.
The different fields of this block are:
dyn: dynamic amount. The keyboard velocity can modulate the EG rates
Keyb # : The EG rates may be modulated by a keyboard table
level 0 : Initial level . Set to maximum for highly percussive sounds.

rate : rate to go to current level
level : level to be reached.
Check a segment box to set a sustain point.
rate=0 will mark the end of envelope.
Last segment is usually rate=0 and level=0

The Low Frequency Oscillators

The different fields of this block are:
Freq: Frequency.
Delay: Delay before the LFO starts to beat.
Attack: Attack time befor the LFO beats with full amplitude.
and an image box: which displays the waveform of the LFO. Click on it to change LFO wave form.

The Keyboard Tables

This block is used to modulate a value (Filter FC, amplitude, EG rate....) by a value depending of the key pressed. (keyboard curve)
You can define this keyboard value by up to 4 points.
Each point is defined by a key number and a value.

The different Algoritms

PCM Forw 12dB: PCM forward loop with 12dB low pass filter.
PCM Altrn 24dB: PCM alternate loop with 24dB low pass filter
PCM Flip 24dB: PCM flip alternate loop with 24dB low pass filter
FM4Y4 : dual 2 operators FM
PCM Forwd Clip 24dB: PCM forward loop with 12dB low pass filter without anti clipping
PCM Fwrd 24dB: PCM forward loop with 24dB low pass filter
PCM Altrn Clip 24dB: PCM Alternate loop with 24dB low pass filter without anti clipping
Noise 12dB :Noise generator through a 12dB filter

The filter generally used has an anti clipping device which allows a progressive saturation of the filter.
This avoids common digital clipping which is generally unacceptable for musical instruments.
The drawback of this device is that it may distort some high harmonic sounds like cymbals, metal sounds....
For such sounds use algoritms without anticlipping device.

Loop Types

Suppose a sample saying hello word

Original sample (one shot):

Now we loop it on the last four letters:

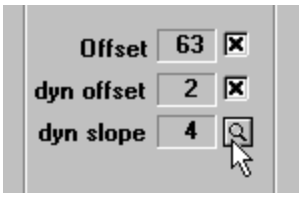
if the loop is « **Forward** », you will hear:

if the loop is « **Alternate** », you will get:

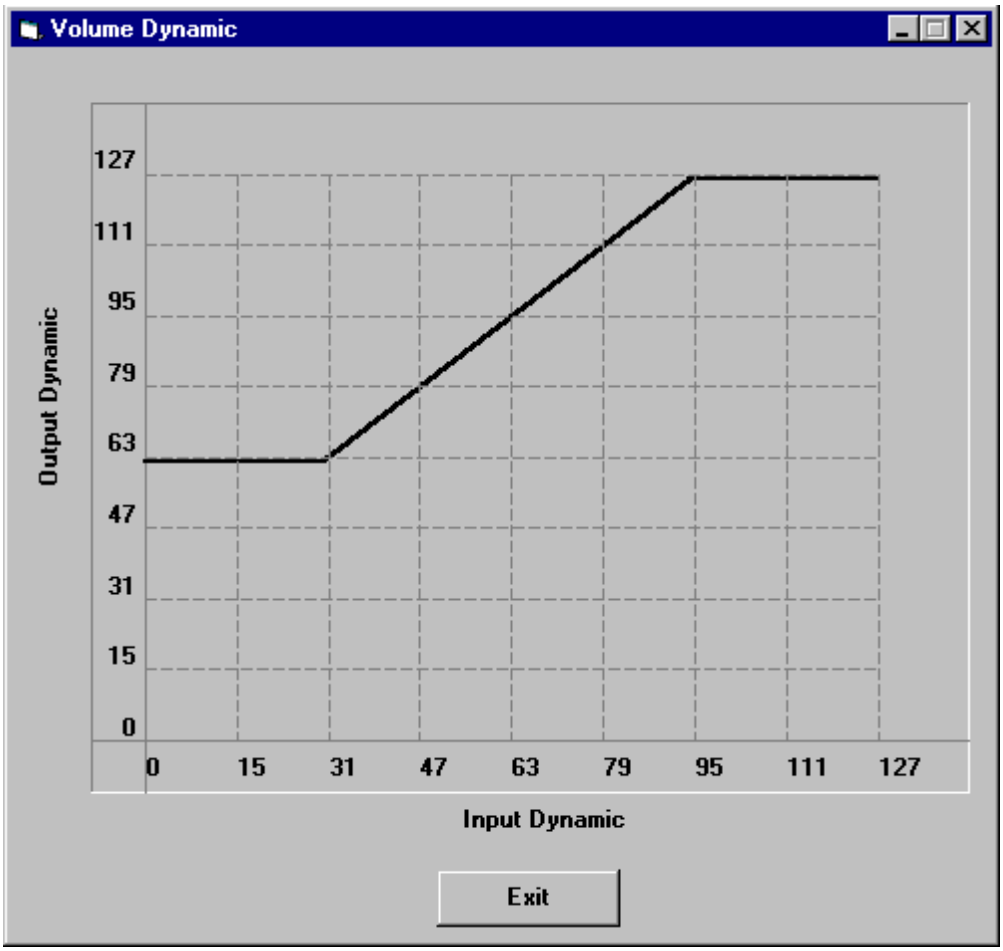
if the loop is « **Flip Alternate** » (Flip), you will have:

New on Version 3.5: Dynamic Curve View

Now it is possible to view the dynamic setting of one block.
Click on the magnify button near the **dyn slope** field



A dialog box showing the corresponding velocity curve will be displayed.



Chapter 1.2

Sound Bank Editor

FOREWORD

A Sound Bank is a set of instruments with its program change list.
The SAM9407 product can manage up to 8 Sound Banks simultaneously, each of them having a priority level. When a note is played, instrument is chosen from the highest priority Sound Bank containing the appropriate program change and variation.

Sound Banks can be used as follow :

-Create a complete GM bank (usually with priority 0).
-This standard bank can be enhanced with additional *user* defined Sound Bank containing selected instruments. This bank should have a higher priority level than GM bank in order to replace standard Sound Bank GM instruments.

OVERVIEW

The Sound Bank editor provides two groups of tools:

Sound Bank Tools:

- Create/edit a sound bank (.94K)
- Compile a sound bank (.94B)
- Create an absolute binary files (.BIN) for ROM application

Sound Card Tools:

- Add/Delete a sound bank in the card memory

CREATE A SOUND BANK

Sound Banks can be created by two ways :
-from a listing file describing the whole Sound Bank.
It's usually used for complete GM banks.
-In adding instrument through the editor.
It's usually used for banks containing a set of selected instruments.
We can't add Drumset by this way.

From a listing file

This requires an *instrument list file* ".94L", the instruments files ".94i" and the sample files ".SMP" or ".WAV".

- Select the *Sound Bank* menu
- Click on *New/From listing file*
- An *Open File* dialogue box appears.
- Select the input text file ".94I" of the *factory* bank to be created.
- Click on OK button. The *factory* bank is created with the same name as the input text file and ".94K" extension.

Empty Sound Bank

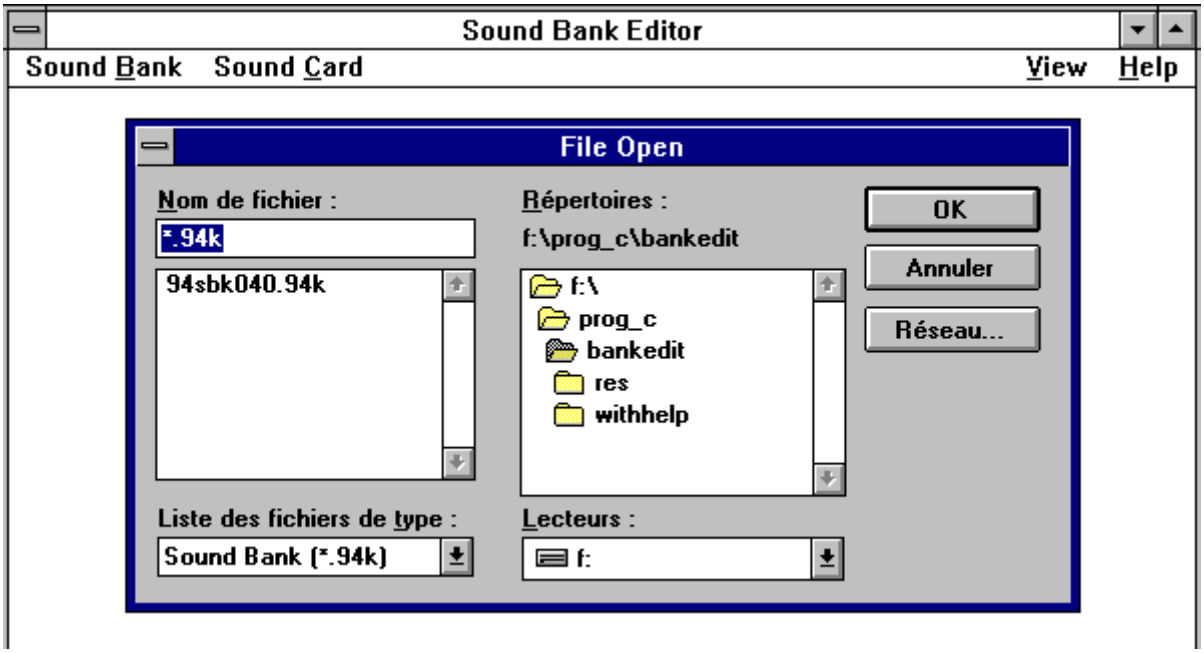
- Select the *Sound Bank* menu
- Click on *New/Empty Sound Bank*
- An *Open File* dialogue box appears.
- Enter a file name (".94K" extension) in the directory you want.
- Click on *OK* button

This creates a Sound Bank without any instrument. The Sound Bank is opened and you add instruments you want.

EDIT A SOUND BANK

Edit a Sound Bank in order to modify or to compile it.

- Select the *Sound Bank* menu
- Click on *Open*.
- An *Open File* dialogue box appears.
- Select the Sound Bank (.94k).
- Click on *OK* button

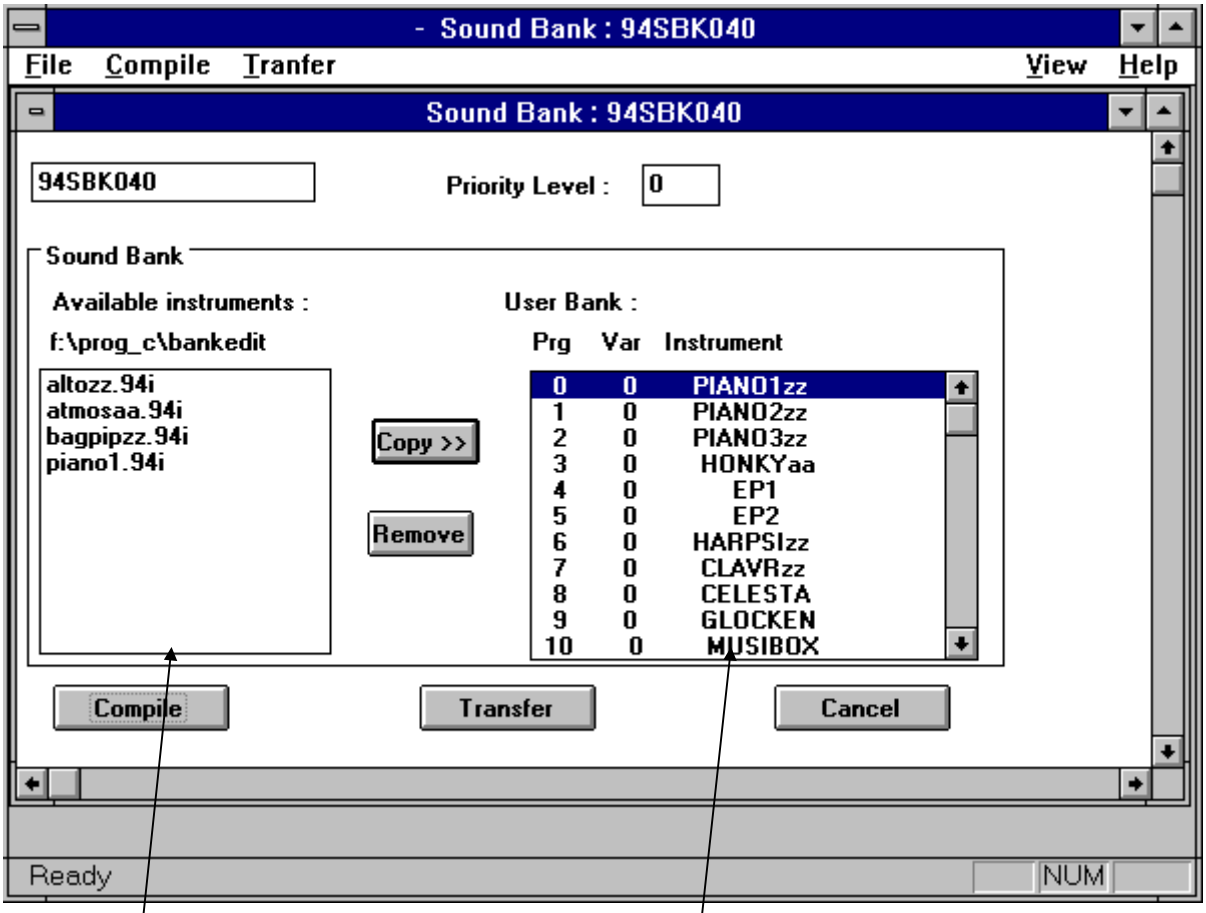


Delete an instrument :

- Select the instrument to delete in the *Sound Bank instrument list*
- Click on the *Remove* button

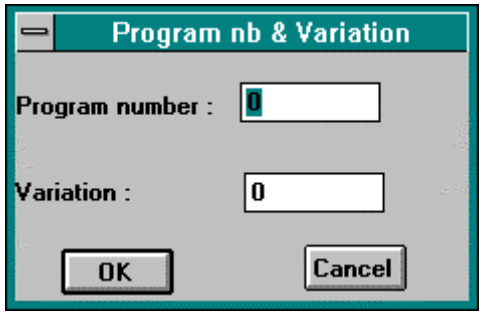
Add an instrument :

- Select the instrument to insert in *Available instrument list*
- Click on the *Copy>>* button
- enter a Program number & a variation number
- Click on the *OK* button



Available Instrument in current directory

Sound Bank instrument list



You can choose the current directory the first time you add an instrument after opening or saving Sound Bank. To change the current directory if some instruments have been added, just save Sound Bank.

Save User Sound Bank:

- Select the *File* menu
- Click on Save or Save as.

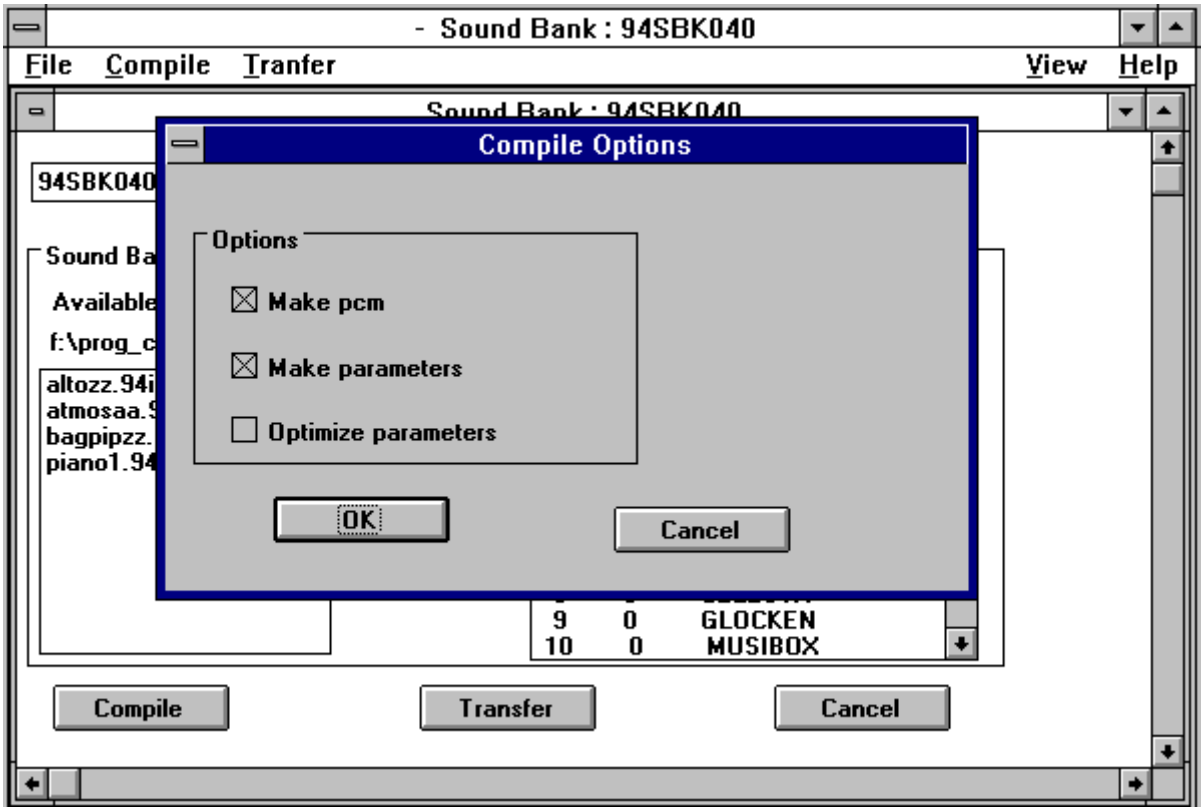
Note : The Sound Bank name is always the file name without extension. If one of them is change, the modification is reported on the other. For example, if you change the file name with ‘save as’, the Sound Bank name is automatically changed.

COMPILE A SOUND BANK

Before loading Sound Bank in card or in a ROM file, it must be compiled. (Press the '*Compile'* button or select *Build* in the *Compile* menu.)
You can choose options in the *Compile/Options* menu.

-Make PCM : Unselect it if you use PCM samples that are not in the Sound Bank.
In this case, you have to use a map.ini file to realize conversion.
-Make parameters : Unselect it to compile PCM only.
-Optimize parameters : Reduce the size of parameters (but increase compilation time).

This binary file can be immediately downloaded into SAM9407 by clicking on the *Transfer* button.



CREATE AN ABSOLUTE BINARY FILE (.94B)

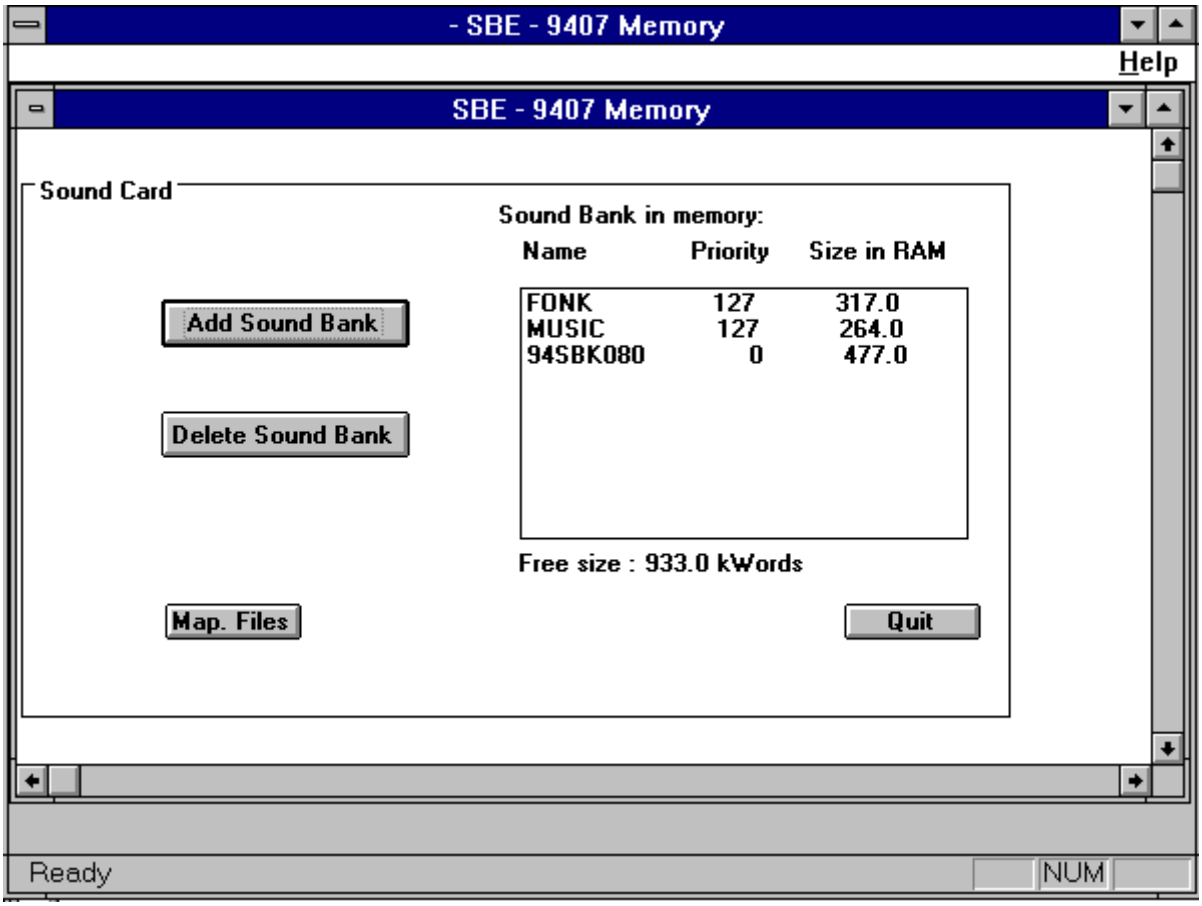
Actually Sound bank binary files ".94B" are not absolute binary files, because they can be mapped anywhere in SAM9407 memory.
ROM applications require absolute binary files with a specified start address.

- Open the desired Sound Bank
- Select the *Transfer* menu
- Click on *Binary File*
- A dialogue box is opened. Enter the *start address* and the *size* of the ROM.
- Click on the *OK* button.

The file created has the extension .BIN.

ADD / DELETE A SOUND BANK IN THE CARD MEMORY

- Select the *Sound Card* menu
- Click on View



Add a Sound Bank

- Click on the *Add Sound Bank* button
- Select the '.94b' file ' in the *Open File* dialogue box.

Delete a Sound Bank

- Select the Sound Bank from the Sound Bank list
- Click on the *Delete Sound Bank* button

Generate Mapping files

To look at the card memory mapping and the midi mapping (instrument program change and variation):
Press the 'Map. Files' button in the Sound Card/View editor
It creates the two text file (.txt) for midi and memory mapping.

Chapter 1.3

PCM Loop Point Adapter

FOREWORD

9407smp.exe is a Windows program that converts sample loop points from Sound Forge format to 9407 Instrument Editor format.

Sound Forge and 9407 sound tools use sample vision format *.smp.
But loop size in Sound forge is one sample less than 9407 editor.
Consequently samples created by Sound Forge have wrong loop (one point less) when they are used with 9407 editor.

When Sound Forge creates a sample, it adds a special comments in the *.smp file. 9407smp.exe check this comments, if the sample was created by Sound Forge, it modifies the loop points (add 1 sample) and modify also the comment (no more Sound Forge signature).
Then the new sample file (original is overwritten) is compatible with 9407 loop but is no more compatible with Sound Forge loop.

How to use 9407smp

Under Windows, run 9407smp program.
Drag the icons of the sample files to be modified (from a file manager) into the 9407smp window.
If the files are created by Sound Forge they will be modified .
If the files are not created by Sound Forge (or already modified) the *.smp is not modified and a message "no loop modification" will be displayed.

Chapter 2

Sound File Format

Ext	Type	Name	Content
94L	ASCII	list	list of 94I with associated MIDI prg chg & variation
94I	RIFF BIN	instrument	instrument parameter
94K	RIFF merge	bank	merged 94L 94I & samples
94B	RIFF bin	binary	merged parameters, samples, reloc table
SMP	BIN	sample	Sample Vision format samples
WAV	RIFF	wave	Microsoft format samples
RMI	RIFF	midi	Microsoft format midi file

FOREWORD

SAMPLE
A sample is a recorded sound with the maximum harmonic amount and the maximum flat amplitude curve (no decay).
Samples can be *looped* (piano, flute, guitar...helicopter) or *one shot* (drum, conga, cymbals... car crash)
When playing a sample with a keyboard, a *looped* sample sounds as long as the key is on, a *one shot* sample sounds during a fixed time independent of the key pressed time.

INSTRUMENT
The MIDI standard defines for a note 3 main parameters:
the note number (frequency)
the velocity (volume and filter amount)
the note on, note off message (duration)
An instrument is a set of samples playing through a variable filter and amplifier.
A sample can be assigned to a range of MIDI note and/or a range of velocity.
The Instrument parameters will affect the amplitude and the filter in real time.

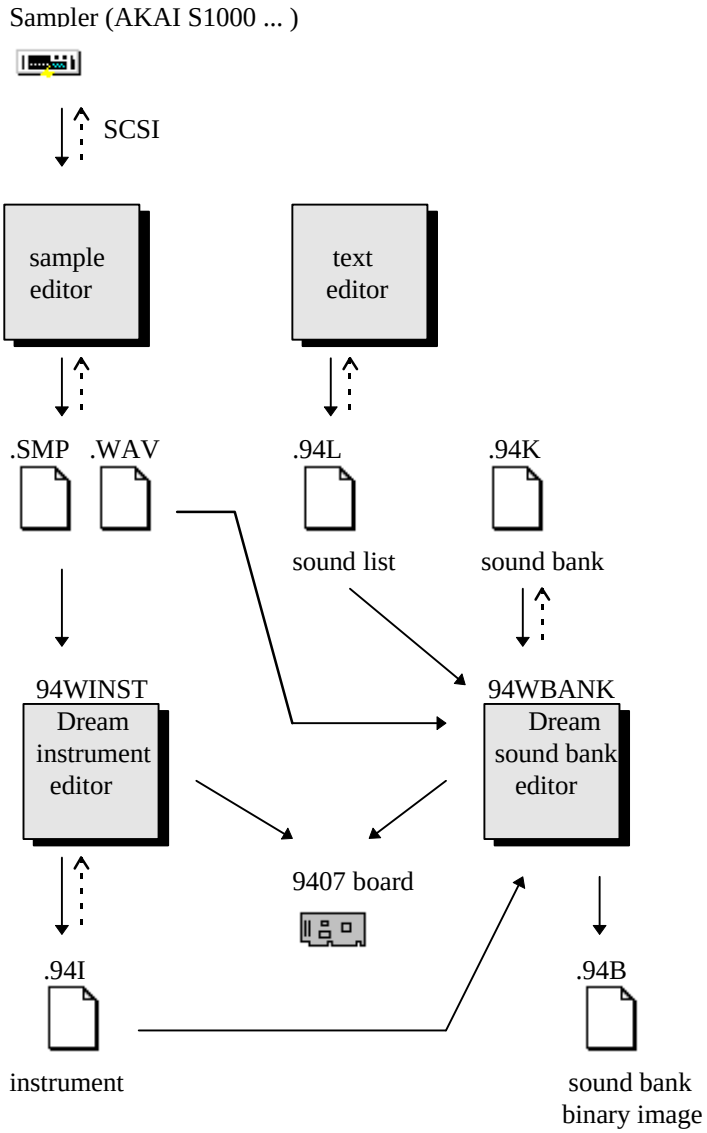
SOUND BANK
The MIDI standard defines also the program change and the bank select messages.
These features allow to select an *instrument* (piano, strings, drums ...) before playing notes.
The MIDI mapping of the *instruments* is called the *sound list*.
The *sound bank* includes *the sound list* the *instruments* and the *samples*

SOUND BANK CONCEPT

The *sound bank* is a set of instruments mapped according to MIDI specification.
SBE can manage up to 8 sound banks.
The basic function are insert a sound bank or delete a sound bank.
Because each sound has its own MIDI mapping, mapping conflicts will happen. To avoid this, a sound bank has a priority level (0 to 255) , a new sound bank with highest or equal priority enforces its MIDI mapping.

A typical multimedia application uses a GM factory sound bank plus a user sound bank with additional instruments.
The double sound bank capability allows to create simple user banks without editing and processing factory banks.

9407 SOUND TOOLS SYNOPTIC



NOTE: Dream doesn't provide any sample editor. We recommend to use *SOUND FORGE* from *SONIC FOUNDRY.*
Sampling can be done with Sound Forge using 94PnP2 card or using AKAI S1000 with midi dump.
Use The 9407smp.exe program (see chapter 1) to convert sample loop pint from *SOUND FORGE to SAM9407 format.*

".94L" SOUND LIST

The 94L files is an ASCII file created with a text editor.
Sound list describes the MIDI program change and variation mapping .

SYNTAX

Any character following ";" is considered as comment.
Space, tabulation or "," can be used to separate argument.
Command is 4 char long "$xxx", any unknown command is ignored by software.

HEADER

Sound list begin with header. Header ends with "$end".
Any header line start with a key word "$xxx".
$pro, $dir & $bin should always be specified and take place at the beginning of header.
Other parameter are optional (default value in bracket).

$pro		product name
$dir		product directory
$bin	["product"..94B]	Upload file
$map	["product" .map]	map listing file
$rep	["product" .rep]	report file
$lst	["product" .lst]	parameter listing file
$end		end of header

SOUND LIST FORMAT

After the header, instrument list is described with instrument name lines and command lines.

Command

$ins *.ins directory, (default dir is current dir), ins directory is used for all *.ins until a new "$ins".
$pcm *.16 directory same remarks as $ins

Instrument

PIANO1	$ p,[b]	p prog nb, b bank nb [default 0]

Drum set

STDDRM1	$drm p,s,e,[b]	p prog nb, s start note, e end note, b bank nb
	... (drums list)	
	$end	end of drum set definition

Drums

STD27	[$exc x]	x exclusion group

".94I" INSTRUMENT DEFINITION

.94I files begin with a HEADER. This header is used by the 94WINST editor.
The header is followed by the INSTRUMENT PARAMETER DEFINITION.

INSTRUMENT HEADER

```
char ckID[4]="SLIS"          ; 4 bytes = "SLIS", sample list information
long ckSize                  ; 4 bytes = length of sample list
   char ckID[4]="SAMP"       ; 4 bytes = "SAMP", sample information
   long ckSize               ; 4 bytes = length of sample information
   char Name[13]                     ; name of sample file in 13 bytes
   char Ext[3]               ; extension of sample file in 3 bytes
   long Smpstart             ; 4 bytes = sample start (relative to 1st sample)
   long Smpend               ; 4 bytes = sample end (relative to first sample)
   int pcm8                  ; 2 bytes = 08h,00h if 8 bit ; 010h,00h if 16 bit
   int SplitID               ; 2 bytes = split which is using sample
```

repeat SAMP block for each different sample. If same sample is used by two different splits, SAMP block must be repeated.
extension recognize by 94WINST are "SMP" and "WAV"

```
char ckID[4]="KLIS"          ; 4 bytes = "KLIS", keyboard tracking list
long ckSize                  ; 4 bytes = length of keyboard tracking list
   char ckID="KBDT"          ; 4 bytes = "KBDT", keyboard information
   long ckSize               ; 4 bytes = length of keyboard information
   int SplitID               ; 2 bytes = split which is using keyboard
   int KbdID                 ; 2 bytes = keyboard number
     char key                ; 1 byte= key number
     char value              ; 1 byte= keyboard value
```

key/value block number can be from 2 minimum till 4 maximum
repeat KBDT block for each different keyboard tracking table

```
char ckID[4]="MINS"          ; 4 bytes = "MINS", instrument information
   long ckSize               ; 4 bytes = size of instrument
   char data[]               ; n bytes, instrument definition (see § 0 PARAMETERS)
```

PARAMETERS

Instrument definition begins with an header, and then is followed by definition of all splits of the instrument.
All pointers are absolute pointers. All values are 16 bit values (word)

Parameter header

Split definition (3 x n words) :
split 1 start value
split 1 stop value
sound 1 pointer
...
split n start value
split n stop value
sound n pointer

Split 1 start format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
mn	note nb (0-127)							rep	dyn (0-127)						

mn : reserved for future use, must be always 0
rep : repeat bit. If repeat=0, when playing same note number new note kills the old one. If repeat=1, new note is added to old one (typically rep=0 for organ sounds type, rep=1 for piano or timpani sounds type).

Split n start format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	note nb (0-127)							0	dyn (0-127)						

Split stop format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	note nb (0-127)							0	dyn (0-127)						

if split start < |0| current note |0| current dyn | <= split stop, then sound pointed by pointer following split value will be played.

Sound pointer format

Absolute pointer to status word (see§)

Note :

This split definition format allows to do layering.
For example, we can define :

key	0...60	61.. 72	73...127
	piano11		piano12
	silence	piano13	

keys from 0-60 will play piano11
keys from 61 till 72 will play piano11+piano13
keys from 73 till 127 will play piano12+piano13

Split definition is :

```
DW 0000h     ; start note = 0, start dyn = 0
DW 0487Fh    ; stop note = 72, stop dyn = 127
DW piano11   ; pointer to piano11 sound status word
DW 04800h    ; start note = 72, start dyn = 0
DW 07F7Fh    ; stop note  = 127, stop dyn = 127
DW piano12
DW 03C00h    ; start note = 60, start dyn = 0
DW 0FF7Fh    ; E =1, stop note= 127, stop dyn = 127
DW piano13
```

This split definition format allows also to do dynamic split.

For example :

key	Key 0-127
dyn<=64	bass11
dyn>64	bass12

Keys with dynamic <=64 will play bass11 sound, >64 will play bass12 sound.
Split definition is :

```
DW 0000h     ; start note = 0, start dyn = 0
DW 07F40h    ; stop note = 127, stop dyn = 64
DW bass11
DW 0040h     ; start note = 0, start dyn = 64
DW FF7Fh     ; E = 1, stop note = 127, stop dyn = 127
DW bass12
```

Split format

Ptrtab pointer (1 word)
Status (1 word)
Kbd pointers (0 to 4 words) (Kbd = keyboard tracking)
Eg pointers (1 to 4 words) (Eg = Envelope Generator)
Mod pointers (0 to 2 words) (Mod = modulator)
Objects number (1 word)
Ma1 objects definition
Ma2 objects definition
Mb objects definition
Mx objects definition
My objects definition
Modulator definitions
Enveloppe generator definitions
Keyboard table definitions
Ptrpab table definitions

Ptrtab pointer

Pointer to Ptrtab. (see §)
Warning ! ! : Sound pointer defined in split definition is not pointer to this word but to next word following (Status).

Status

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	2x	alg					0	Eg nb		Mod nb		Kbd nb		

2x : double slot flag (always 0 for time being)
alg : algorithm number (0-31) (see also alglist.wri file)
0 : 24db filter, forward loop
1 : 24db filter, alternate loop
2 : 24db filter, alternate loop with sign inversion
kbdnb : number of keyboard table used : 100=0, 011=1, 010=2, 001=3, 000=4
modnb : number of modulator used : 10=0, 01=1, 00=2
egnb : number of envelope used : 11=1, 10=2, 01=3, 00=4
Algorithm list can change depending on version of program. On first version of program for sound compatibility with 9233, algorithm 0 is 12db filter forward loop instead of 24db filter forward loop. Algorithms 1 and 2 are 24db filters.

Objects number

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MA nb		MA2 nb			MB nb			MX nb			MY nb			Freq off	

MA1 nb : 11=1, 10=2, 01=3, 00=4
MA2 nb : 111=1, 110=2, ... 000=8
MB, MX and MY : 111=0, 110=1, ... , 000=7
Freq Off : Number of 1st frequency object number among MA2 objects

MA1 object

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
kbd	0	0	dyn	Filt		Kbd nb		rfu	Lfo	EG	lnb	Egnb		out	off
0	Amplitude (0,127)							dyn offset				dyn slope			
0	Offset (0,127)							lfo amount (-127,+127)							

Kbd : Keyboard table tracking yes/no
Dyn : Dynamic yes/no
Filt : 0 normal, 1 filter input, 2 filter Fc, 3 filter Q
Kbd nb : Keyboard table number (0-3)
Lfo : Modulator yes/no
Eg : Envelope yes/no
Lnb : Modulator number (0-1)
Egnb : Envelope number
Out : Amplitude being general volume of sound
Off : amplitude with offset
Dyn offset, Dyn slope : define curve applied to MIDI dynamic before applying it to amplitude :

Case of slope positive :
slope = 0000,0001,0010,0011,0100,0101,0110,0111
is respectively 0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75
offset = 1000, 1001, 1010, ... 1111, 0000, 0001, 0010, ... 0110, 0111
is respectively -128, -112, -96, ..., -16, 0, +16, +32, ... , + 96, + 112

Case of slope negative
slope = 1111, 1110, ... 1000
is -0.25, -0.5, ... -2
offset = 1000, 1001, 1010, ... 1111, 0000, 0001, 0010, ... 0110, 0111
is respectively 0, +16, +32, ..., +112, +128, +144, ... , , +224, +240

Notes : Word Offset/Lfo amount is present only if either Lfo or Off bits are set

MA2 Object

MA2 frequency object

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Lfo amount (-127,+127)								pg	kbd typ		Eg	fmt	Lnb	Eg nb	
detone (-63,+63)								detune (0,127)							typ
page								Eg amount (-31,+31)							

Typ = 0 to indicate MA2 frequency object
kbd typ : 0 normal, 1 not used, 2 Fix frequency, 3 Fix frequency modulated by kbd0
Eg : Eg yes/no
Lnb : Mod number (0-1)
Eg nb : Eg number (0-3)
pg : 1 if holds bank information
fmt : used for fm sounds. If 1, +64 is added to detone.

NOTES : In case of fix frequency, fix frequency is given in word detone/detune
with format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
a9	a8	a7	a6	a5	a4	a3	a2	a1	a0	f5	f4	f3	f2	f1	f0

(40h=nominal frequency)
detone is signed, detune is always positive

word Page/EG amount is present only if Eg or pg bit is set

MA2 Xfer object

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR 27-20								PAR 19-12							
PAR 11-0												0	0	0	typ

Typ=1 to indicate MA2 Xfer object type
PAR27-0 are 28 bit value written to SAM.

MB object

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR 11-0												0	dum	0	mix
PAR 27-20								PAR19-12							

mix : set if PAR 11-0 holds mix value (effect mix)
dum : set if dummy object (in that case 2nd word is skipped)

MX object

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR15-0															

Notes :
If PAR15-0 = 0000h, mx object is not coded, except if one mx object <> 0 is following
PAR15-0 = 0001h means damp object
Damp object must be coded only if one mx object <>0 is following.

MY object

MY Xfer object

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR 11-0												0	dum	typ	mix

Typ=0 to indicate MY Xfer type object
Mix : set if parameter holds mix value (main/aux mix)
Dum : set if dummy object

MY Amplitude object

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Amplitude (0,127)							dyn	kbd	0	0	filt		typ	0
dyn offset				dyn slope				0						kbd nb	

Typ=1 to indicate My amplitude type object
Dyn : Dynamic yes/no
Kbd : Keyboard table yes/no
Filt : 0 normal, 1 filter input, 2 filter Fc, 3 filter Q
KBD nb : Keyboard table number
Dyn offset, slope : same as MA1 object

Note : Kbd/dyn word is present only if either dyn or kbd bit is set

Modulator

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Lfo attack (0,127)							0	0	0	0	wave			1
0	Lfo speed (0,127)							0	Lfo delay (0,127)						

Lfo wave : 0 sinus, 1 up saw, 2 down saw, 3 square, 4 random, 5 positive sinus

Enveloppe generator

Header :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Init amplitude (0,127)							dyn eg (0-31)					kbd	kbd nb	

Kbd : Keyboard table yes/no
Kbd nb : Keyboard table number
Dyn eg : Dynamic applied on init amplitude, level and rate of each segment

Segment definition :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
code			Level (0,64)						Rate (0,127)						
Jump back															

Code : 0 continue on next segment
1 loop
2 sustain point, wait for key off
3 envelope end
4 envelope end at sustain point (case of sustain point amplitude = envelope end amplitude)

Jump back : number of segments back in case of loop (if 1 loop on loop segment, if 2 loop one behind loop segment)
Notes : Jump back needed only if code is 1, not present for other codes.
Maximal segment number is 8

Keyboard table

Length of keyboard table depends on split definition. For a sound defined as split from note N till note M (M excluded), keyboard table is (M-N)/2 byte long :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note N+2, N+3								Note N, N+1							
...															
Note M-2,M-1								Note M-3,M-4							

Note : Keyboard values are from -127 till +127

PTRTAB table

Give indication of position of mix, and modulators data.
word 0 pointer to mix1 value or 0 if none
word 1 mix1 address
word 2 pointer to mix2 value or 0 if none
word 3 mix2 address
word 4 pointer to mix3 value or 0 if none
word 5 mix3 address
word 6 pointer to mix4 value or 0 if none
word 7 mix4 address
word 8 0
word 9 pointer to mod0 pointer or 0

word 10 pointer to mod1 pointer or 0
word 11 0

mix address :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	block nb			0	0	0	0	0	M/S	ram ad	

block nb = 010 if MB, 100 if MY
ram ad = 0 to 3
M/S = 1 if slave slot in case of double slot algorithm.
Note : all pointers in PTRTAB table are pointers relative to status word in sound definition

".94K" SOUND BANK

The Sound Bank file includes all the source files required by the sound bank: Sound list, Instruments, Samples.

FORMAT

File format is based on RIFF chunk

```
char    ckID[4]= "PRIO"
long    ckSize
        char    name[8]
        word    Priority level

char    ckID[4]= "INSX"
long    ckSize
        index table (see §)

char    ckID[4]= "SMPX"
long    ckSize
        index table (see §)

char    ckID[4]= ".94L"
long    ckSize
        "xxxxxxxx.94L" file (see §)

char    ckID[4]= ".94I"
long    ckSize
        "xxxxxxxx.94I" file (see §)

char    ckID[4]= ".SMP"
long    ckSize
        "xxxxxxxx.SMP" file (see §)

char    ckID[4]= ".WAV"
long    ckSize
        "xxxxxxxx.WAV" file (see §)
```

INSTRUMENT INDEX TABLE

This table allowed direct access to INST chunk
It should take place at the beginning of file just after PRIO chunk

Instrument index definition:

char Name[13]	13 bytes	name of file
char Ext[3]	3 bytes	extension of file
long pt	4 bytes(LSB first)	abs byte pointer to chunk in ".94K"
checksum	long	checksum on first 200bytes

SAMPLES INDEX TABLE

This table allowed direct access to .SMP & .WAV chunk
It should take place after INSX chunk
The samples are sorted by decreasing size.

Sample index definition:

char Name[13]	13 bytes	name of file
char Ext[3]	3 bytes	extension of file
long pt	4 bytes(LSB first)	abs byte pointer to chunk in ".94K"
checksum	long	checksum on first non zero 200bytes
start	long	first used sample (start or loop start)
end	long	last sample
occurrence	word	number of split using sample (0 means that there is no need to keep sample inside sound bank)
sample length	word	8 or 16 (only 8 & 16 bits supported)

If SBE needs to insert the same samples with different sample length the sample is duplicated with both length and a warning message is displayed.

".94B" BINARY

The Binary File is the result of compilation of the Sound Bank (.94K).
The Instruments and the samples are stored as SAM9407 reloc format.

Format

File format is based on RIFF chunk

```
char    ckID[4]= "PRIO"
long    ckSize
        char    name[8]
        word    Priority level

char    ckID[4]= "IMAP"
long    ckSize
        variable Instrument table
        word            FFFFH
        variable drumset table
        word            FFFFH

char    ckID[4]= "PARA"
long    ckSize
        94 instruments parameter in reloc format (same as ".94I" without instrument header block (see §)

char    ckID[4]= "SAMP"
long    ckSize
        samples (see §)

char    ckID[4]= "RLOC"
long    ckSize
        param reloc table (see §)
```

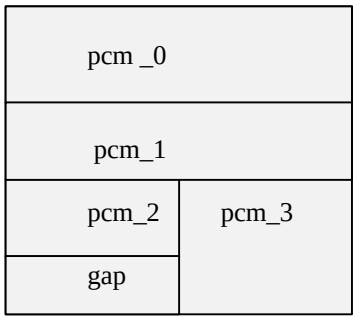
Instrument / drumset table

These tables includes instrument/drumset definition data with the following format:

```
word        program number
byte        variation number
byte        param page number (0/1)
word        16 bit pointer in param page
```

SAMPLES

Samples data are stored in either 8 or 16 bit signed format.
Sample data should be inpage 256K word.
Each 512KB page should start with 16 "0" word and end with 16 "0" word. This gaps are necessary to avoid address overflow in DSP address unit.
In the following example there are 4 samples to map: pcm_0 & pcm_1 on 16 bits, pcm_2, pcm3 on 8bits



Samples are mapped on 256K word pages numbered for 1 to n.
Page 1 to n-1 are always considered as full, Page n size is almost always inferior to 256K.

RELOC TABLE

POINTER LIST

instrument parameter (see §) includes many pointers referring to param or samples.
The following list gives the reloc parameter with their associated target (param, samples)

name	§	location in param	mem space	content
sound x pointer		param header	param	point to status word §
PTRTAB pointer		sound	param	pointer to Ptrtab §
KBD pointer		sound	param	pointer to KBD object §
EG pointer		sound	param	pointer to EG object §
MOD pointer		sound	param	pointer to MOD object §
MA2 Xfer obj		MA2 object	samples	MA2[0] PG, MA2[1] LOOP
MB Xfer object		MB object	samples	MB[0] PHI, MB[1] END
pointer to mix		PTRTAB table	param	pointer to mix address
pointer to mod		PTRTAB table	param	pointer to MOD pointer

FORMAT

long pointer point the word in param which is reloc type
byte type
byte block Page nb , 1 or 2 for param , 1 to n for Samples

type	
0	param pointer
1	MA2 [0] PG
2	MA2 [1] LOOP
3	MB[1] END

pointer to full 64K page are always correct (no needs to be reloc)
pointer to MA2[1] or MB[1] full 256K page are also correct.

".SMP" SampleVision SAMPLE FORMAT

This is the Sample format created by Sample vision. This Format enable loop sample.
A User comment has been added to support fractional loop not supported by standard .SMP format.
.SMP format is going to be replaced by .WAV format which is the PC standard.

HEADER

Offset	Size	Description
0 0H	18 12H	'SOUND SAMPLE DATA' ASCII file ID
18 12H	4 4H	'2.1 ' ASCII file version
22 16H	60 3CH	user comment : "Loop"/"Alt" " Size=xxx.xxx ... "
82 52H	30 1EH	Sample name (left justified 30 ASCII char)
112 70H	4 4H	Sample size (sample data count in words. store as double word Intel format)
Total	116 74H	

DATA

Offset	Size	Description
116 74H	2*Size	Sample data signed 16bit integer Intel format
2*Size +116	2 2H	reserved
Total	2*Size+2	

LOOP

They are 8 loop definition
Loop block start at : 118 + 2*Size / 76H + 2*Size
loop block end at : 205+ 2*Size / CDH + 2*Size

Offset	SIze	Description
0	4	Loop n Start (sample count -> word count)
4	4	Loop n End
8	1	Loop n Type (0 = loop off, 1 = forward, 2 = alternate)
9	2	Loop n Count (times to execute loop before jumping to next loop)
Total	11 BH	

MARKER

They are 8 marker definition
marker block start at : 206+ 2*Size / CEH + 2*Size
Marker block end at : 317+ 2*Size / 13DH + 2*Size

Offset	Size	Description
0	10	Loop n Start (sample count -> word count)
10	4	Loop n End
Total	14 EH	

marker 8 "Start " is used as sample start

MISCELLANEOUS

Misc. block start at :318+ 2*Size / 13EH + 2*Size
Misc. block end at : 330+ 2*Size / 14AH + 2*Size

Offset	Size	Description
0	1	MIDI Unity Playback Note (MIDI note that will play the sample at its original pitch)
1	4	Sample rate in Hz
5	4	SMPTE Offset in subframes
9	4	Cycle size sample count in one cycle of the sample sound. 1 if unknown
Total	13 DH	

".WAV" MICROSOFT SAMPLE FORMAT

This is the Microsoft Standard format for Sample.

File Format

".WAV" file used RIFF chunk format.

```
char ckID[4]="RIFF"                 ; 4 bytes = "RIFF"
long ckSize                                 ; 4 byte = chunk size
   char ftype[4]="WAVE"                     ; 4 bytes = "WAVE" Form type ("RMID" is the other
                                    form type)
   char ckID[4]="fmt "              ; 4 bytes = "fmt ", format information
   long ckSize                      ; 4 byte = chunk size
      word wformatTag               ; 2 byte = WAVE format, 1 WAVE_FORMAT_PCM
                        (see§)
      word nChannels                ; 2 byte = 1 mono, 2 stereo
      long  nSamplPerSec            ; 4 bytes = playback sampling rate Hz
      long nAvBytesPerSec           ; 4 bytes = average nb bytes per second
      word nBlockAlign              ; 2 bytes = block alignment
      word nBitsPerSample           ; 2 bytes = sample size 8 byte, 16 word
   char ckID[4]="data"              ; 4 bytes = "data", data information
   long ckSize                      ; 4 bytes = length of sample information
      PCM[]                                 ; bytes/word = samples
```

nAvBytesPerSec = nChannels*NBitsPerSecond*nBitsPerSample/8
Playback software can estimate the buffer size using nAvBytesPerSec value.

nBlockAlign = nChannels*nBitsPerSample/8
Playback software needs to process a multiple of nBlockAlign bytes of data at a time, so that the value of nBlockAlign can be used for buffer alignment

Wave Format

MM specify only one value wFormatTag=1 for Pulse Code Modulation Format (WAVE_FORMAT_PCM)
For WAVE_FORMAT_PCM, fmt chunk contains a single field nBitsPerSample.
This field specifies the number of bits of data used to represent each sample of each channel. If they are multiple channels, the sample size is the same for each channel.

sample 1	**sample 2**	**sample 3**	**sample 4**
byte	byte	byte	byte

Data packing for 8-bit mono PCM

sample 1		**sample 2**	
left chan byte	right chan byte	left chan byte	left chan byte

Data packing for 8-bit stereo PCM

sample 1		**sample 2**	
low byte	high byte	low byte	high byte

Data packing for 16-bit mono PCM

sample 1			
left chan low byte	left chan high byte	right chan low byte	right chan high byte

Data packing for 16-bit stereo PCM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCM [11-0]												0	0	0	0

12-bit PCM

One to eight bits PCM are **Unsigned integer**
Nine or more bits PCM are **Signed integer**

Additional chunk

These chunks are generated by "SampleVision for Windows".
These chunks are undocumented and are interpreted from the ".WAV" generated file.

```
char ckID[4]="smpl"          ; 4 bytes = "smpl" sample vision loop information
long ckSize                  ; 4 byte = chunk size (3CH)
   ??                        ; 12 bytes
  byte MIDINote              ; 1 bytes = MIDI unity playback note
  ??                         ; 31 bytes
   LoopStart                 ; 4 bytes
   LoopEnd                   ; 4 bytes
   ??                        ; 8 bytes
char ckID[4]="LIST"          ; 4 bytes = "LIST" comment information
long ckSize                  ; 4 byte = chunk size
   char ltype[4]="INFO"      ; 4 bytes = "INFO"
   char ckID[4]="INAM"       ; 4 bytes = "INAM" information
   long ckSize               ; 4 byte = chunk size
      char []                ; "Exported From Wave For Windows"
```

".RMI" MICROSOFT MIDIFILE FORMAT

RMI is Microsoft file format for midifiles.It allows to include additional information to the midi sequence.
A Dream RMI file includes a midifile and a soundbank.

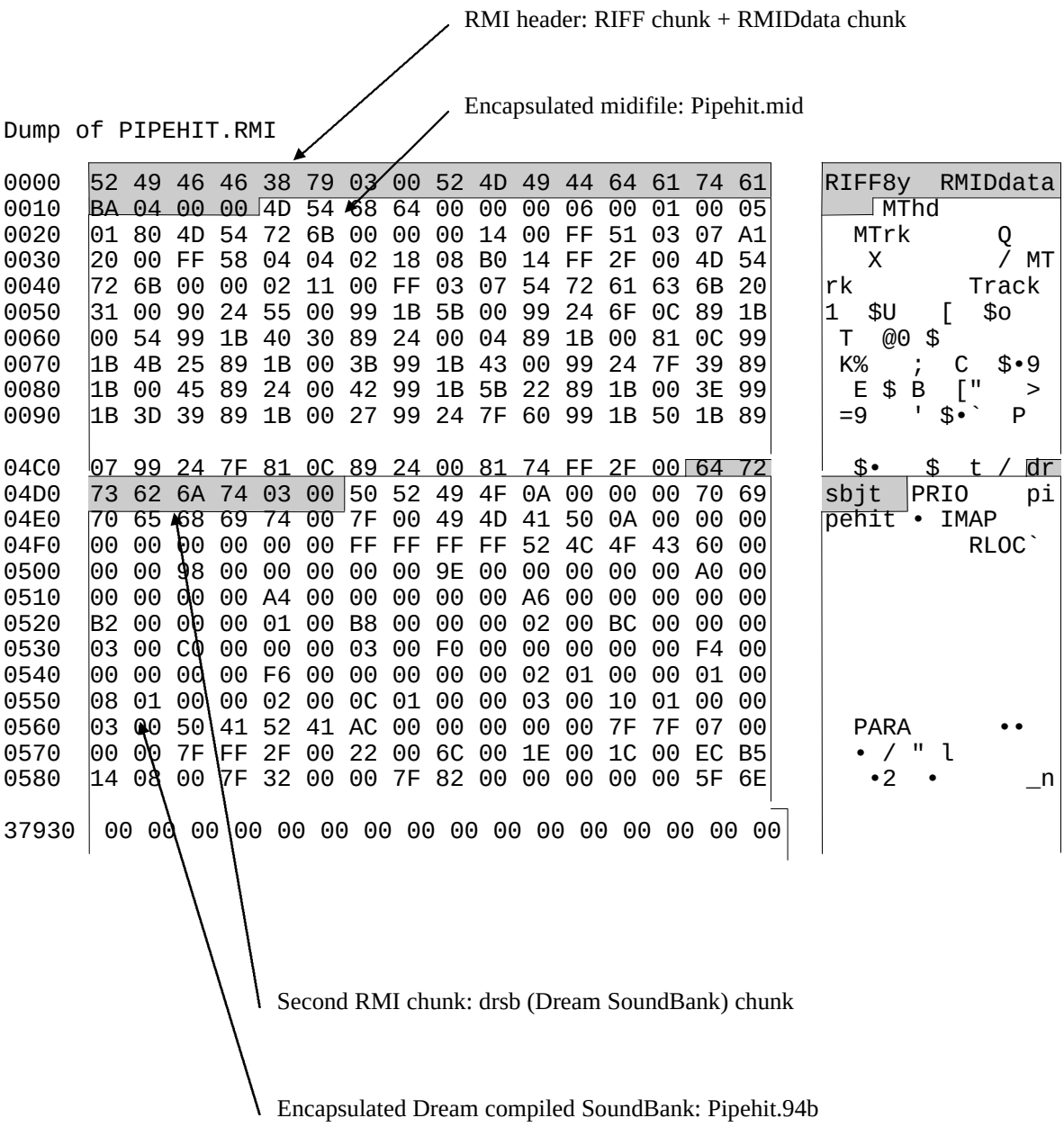
Dream RMI Format

```
char riff[4]=« RIFF »         RIFF chunk
long ckSize                   size of chunk

char rmid[4]=« RMID »         RMID chunk
char data[4]=« data »         data chunk
long ckSize                   size of chunk

char file1[]                  standard midifile (*.mid)

char drsb[4]=« drsb »         dream Soundbank chunk
long cksize                   size of chunk

char file2[]                  Dream compiled soundbank (*.94b )
```

Dream RMI files can be generated with RMI.EXE (DOS program)

Example

```
Dump of PIPEHIT.MID

0000  4D 54 68 64 00 00 00 06 00 01 00 05 01 80 4D 54    MThd          MT
0010  72 6B 00 00 00 14 00 FF 51 03 07 A1 20 00 FF 58    rk      Q      X
0020  04 04 02 18 08 B0 14 FF 2F 00 4D 54 72 6B 00 00          / MTrk
0030  02 11 00 FF 03 07 54 72 61 63 6B 20 31 00 90 24        Track 1  $
0040  55 00 99 1B 5B 00 99 24 6F 0C 89 1B 00 54 99 1B    U   [  $o    T
0050  40 30 89 24 00 04 89 1B 00 81 0C 99 1B 4B 25 89    @0 $         K%
0060  1B 00 3B 99 1B 43 00 99 24 7F 39 89 1B 00 45 89      ;  C  $•9   E
0070  24 00 42 99 1B 5B 22 89 1B 00 3E 99 1B 3D 39 89    $ B  ["   >  =9

04A0  24 00 81 79 99 24 7F 79 89 24 00 82 07 99 24 7F    $  y $•y $    $•
04B0  81 0C 89 24 00 81 74 FF 2F 00


Dump of PIPEHIT.94B

0000  50 52 49 4F 0A 00 00 00 70 69 70 65 68 69 74 00    PRIO    pipehit
0010  7F 00 49 4D 41 50 0A 00 00 00 00 00 00 00 00 00    • IMAP
0020  FF FF FF FF 52 4C 4F 43 60 00 00 00 98 00 00 00        RLOC`
0030  00 00 9E 00 00 00 00 00 A0 00 00 00 00 00 A4 00
0040  00 00 00 00 A6 00 00 00 00 00 B2 00 00 00 01 00
0050  B8 00 00 00 02 00 BC 00 00 00 03 00 C0 00 00 00
0060  03 00 F0 00 00 00 00 00 F4 00 00 00 00 00 F6 00
0070  00 00 00 00 02 01 00 00 01 00 08 01 00 00 02 00
0080  0C 01 00 00 03 00 10 01 00 00 03 00 50 41 52 41                PARA
0090  AC 00 00 00 00 00 7F 7F 07 00 00 00 7F FF 2F 00          ••    • /

37460  00 00 00 00 00 00 00 00 00 00
```



Dump of PIPEHIT.RMI

```
0000   52 49 46 46 38 79 03 00 52 4D 49 44 64 61 74 61    RIFF8y  RMIDdata
0010   BA 04 00 00 4D 54 68 64 00 00 00 06 00 01 00 05      MThd
0020   01 80 4D 54 72 6B 00 00 00 14 00 FF 51 03 07 A1    MTrk      Q
0030   20 00 FF 58 04 04 02 18 08 B0 14 FF 2F 00 4D 54     X        / MT
0040   72 6B 00 00 02 11 00 FF 03 07 54 72 61 63 6B 20   rk        Track
0050   31 00 90 24 55 00 99 1B 5B 00 99 24 6F 0C 89 1B   1  $U   [  $o
0060   00 54 99 1B 40 30 89 24 00 04 89 1B 00 81 0C 99    T  @0 $
0070   1B 4B 25 89 1B 00 3B 99 1B 43 00 99 24 7F 39 89    K%   ;  C  $•9
0080   1B 00 45 89 24 00 42 99 1B 5B 22 89 1B 00 3E 99     E $ B  ["   >
0090   1B 3D 39 89 1B 00 27 99 24 7F 60 99 1B 50 1B 89    =9   ' $•`  P

04C0   07 99 24 7F 81 0C 89 24 00 81 74 FF 2F 00 64 72     $•   $  t / dr
04D0   73 62 6A 74 03 00 50 52 49 4F 0A 00 00 00 70 69   sbjt PRIO    pi
04E0   70 65 68 69 74 00 7F 00 49 4D 41 50 0A 00 00 00   pehit • IMAP
04F0   00 00 00 00 00 00 FF FF FF FF 52 4C 4F 43 60 00            RLOC`
0500   00 00 98 00 00 00 00 00 9E 00 00 00 00 00 A0 00
0510   00 00 00 00 A4 00 00 00 00 00 A6 00 00 00 00 00
0520   B2 00 00 00 01 00 B8 00 00 00 02 00 BC 00 00 00
0530   03 00 C0 00 00 00 03 00 F0 00 00 00 00 00 F4 00
0540   00 00 00 00 F6 00 00 00 00 00 02 01 00 00 01 00
0550   08 01 00 00 02 00 0C 01 00 00 03 00 10 01 00 00
0560   03 00 50 41 52 41 AC 00 00 00 00 00 7F 7F 07 00     PARA      ••
0570   00 00 7F FF 2F 00 22 00 6C 00 1E 00 1C 00 EC B5    • / " l
0580   14 08 00 7F 32 00 00 7F 82 00 00 00 00 00 5F 6E     •2  •      _n

37930   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```