Assampler

Henning Thielemann

COLLABORATORS			
	TITLE : Assampler		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Henning Thielemann	January 19, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Assampler

1.1 Assampler

Assampler 00.99

(banana version)

or

The quest for the formula the sounds are made of.

or

a new program,

a little revolution,

the most spectacular thing this year

-

or just rubbish?

1.2 Introduction

Assampler is the name of my new sound processor program and is a mix of "Assembler" (programming language of my synthesizer.library), generally "to assemble" since you can assemble several processes to algorithms and "as sampler" for one of my first wishes to make a audio digitizer unnecessary by using this program (what surely never will happen).

The guide and the program are passed to you just from within the development, bad tested and few documented, non-localized and with mixed German/English words, so call it preview pre-alpha version, if you want. Nevertheless some of you might be interested in what is going on here and two people surely want to see some results of the request they did on NATW.

The main program (includes file format) and the synthesizer.library are changing from time to time, but since there is no public release, I save the work to update version numbers :-(That means for further preview versions you must ensure all components to be installed when starting the program because it cannot detect if one component is not in the current version. Since I have already made some example algorithms with this version I will surely add adaptation routines for this files when the format changes in following versions. So you can save your work, but some conflicts are possible.

This guide shall only help you getting started with Assampler. A complete guide is currently written by me in German. This will be tested and proof-read then by some beta testers, after that the result will be translated to english which will replace this thing. For this time you must live with this document :-(

1.3 distribution

We do now enter the beta-test phase. Please play around with the program and tell me about bugs and inconsistencies. I won't add any new feature in the near future to prevent me from introducing new bugs.

1.4 Disc lamer

1. You are allowed to distribute this version of Assampler for free whereever you want.

2. For commercial usage a licence is required.

3. You are allowed to sell Assampler or to add Assampler to a public domain library, to a cover disk or cover CD or something similar as long as the price for the data medium plus the copy fee does not exceed 1 Euro.

4. Although I can't force you to do that, but if you wrote a workshop about Assampler (sometimes wonder happens) I ask you to let me confirm your article. I don't want that misunderstandings are distributed as the truth.

5. I'm not responsible for damages that result from using my program. At least I can guarantee that I didn't developed program parts that can format hard disks or that replicate Assampler code. Assampler is also not a trojan horse. For programs that are called Assampler, too, but which are not of me, this guarantee is not valid.

6. It is explicitly allowed to modify Assampler for private usage. But it is not allowed to re-distribute such modified code.

7. Files produced by Assampler can be spread without any restriction by me, of course.

1.5 installation

I have written an installer script, so installing should be very easy.

Please install all packets listed in the Required tag in the readme file (MUI, AHI, TableGroup) first. Then extract the Assampler archive to somewhere you want.

The debug version of the program produces lots of debugging output that is currently written to a device called staticram:. You have hopefully no device that is always called so. Otherwise start Assampler with ToolType or CLI parameter DEBUGOUTPUT=your_preferred_device:logfile_name, which should be on a fast device with about 2 or 3 MB of free space, the best is surely the Ram disk or NIL.

1.6 bugs

Explainable bugs

There are some bugs, that I already know - they needn't to be reported, of course:

Listtree

Kmel's Listtree.mcc gets confused if you remove an invisible node from a listtree. This occurs, if a display of a sound group or a sound select gadget is shown with closed nodes while you move around sounds that are within the groups which nodes are closed (clear?). It is to hard to work around this bug, please change and go back to the display if the listtree shows graphical trash.

Sound select

If you close a sound select you can still choose sounds by pressing the up and down cursor keys within the string gadget. You finally select them by pressing return. Since this list isn't further refreshed it might hold invalid data and you may select a sound which is already deleted. This leads to crashes.

Virtual groups

The virtual groups of MUI 3.8 (and below) are not consistent. If you drag a sound box out of an algorithm display you will drag some border graphics, too, if the box is not completely visible. It is also possible to press "through" some window elements, e.g. you select a sound box if you click on the scroller gadgets of a virtual group.

Sound displays

If you change values like SampleFreq or Begin in a information window of a data sound, the modifications will not correctly forwarded to the corresponding displays sometimes.

TearOff

TearOff.mcc does not properly refreshes display while layed out to a window. I have to check whether it is my or TearOff's fault.

Volume adaption

The volume of signal data between processes has to be adjusted automatically. This does not work always. Therefore it may be that an algorithm works if you calculate every process manually but not if you run everything in one go. You have to insert a limit process then.

Unexplainable bugs

Some bugs I could not reproduce until now. Can someone help me? Run Enforcer, Mungwall, Poolwatch along with the debug version of Assampler and send me every logfile, please.

Crash/Freeze/Guru on quitting Assampler

There could be much reasons which are responsible for that.

Assampler quits, when deleting a sound

Don't know what previous actions cause this. Check if open windows of the deleted sound influence this. Find out number, type of windows, whether the windows are of the sound itself or of a sub sound. Or this depends on previous calculations, cut operations, rearranging within groups.

"Variable not found" was raised without any visible reason

Be careful, you might get this error if you load corrupt data files or whenever you type wrong formulas in the expression gadgets. This nothing strange. But what I watched was while checking some examples from the Producer/melody directory. If this error occured I not much time to check out, which variable he meant, because one of the next clicks crashed the machine.

Got this problem two times but could neither reproduce it nor get a logfile (forgot to set the tooltype).

Deadlocks

When using sounds multiple times at once for calculating, cutting, playing, saving, etc. Assampler has to preserve, that no corrupt data is produced. The one problem is, that Assampler might allow you things, you should obviously avoid, the other problem is that Assampler might run into a deadlock by waiting that other parts of the program release there data. I think I have solved this problems now, but you never know ...

Inconsistencies

Information window

The variable window of an information window was designed as an attached window. This means, the contents changes when the information window switches to another sound. Practices shows, that this behaviour is not intuitive. Most people expect a requester here, that has an Ok and Cancel button. Improving this would again need some time and testing, so I will postpone this.

Preferences window

Similar problem - preferences window is not designed as requester, save and use button don't quit it. Every change is applied as soon as the corresponding item is refreshed. This is a very easy implementation that is supported by MUI. Will be improved when the MUI configuration system does.

Group display

Listtree shows it entries in a order that suggests left and right direction for switching between levels, up and down direction for going through the sounds of one level. But the arrow buttons uses the opposite interpretation. Do you find this confusing?

1.7 Taking first steps with Assampler

Conception

Classes

Although sometimes not very easy to understand all the following classes are treated as sounds by Assampler.

data sounds - sounds that have in any way a huge stock of data that describes the sound

sample - traditional audio samples

ram - sampled sounds hold in ram (in 16 bit); will be created by "Load"ing sounds

disk - raw sampled sounds remaining on disk all the time; they may be much larger than available ram; every operation on it destroys the original data on disk, sorry; if created from the classes window, the file on disk will be cleared!, if you don't want to overwrite an existing file then "Open" it; associated files are closed (but not deleted) when the disk sampled sound objects are deleted

spectrum - offers spectrogram analysis, synthesis and visualization; processes to manipulate them (mainly filters) are now enabled, but they may change in future, because they contradict the principle of reusability at different sample rates

spline - helps you creating control curves, waveforms etc. by hand, but is much better than freehand drawing

sound groups - holds multiple sounds

list - collection of sounds; every class for sub-sounds is allowed including again groups; every process started on a group is applied on all sub-sounds; list groups help you managing all the sounds you create within a session

stereo - multi-channel sounds; all sub-sounds must be of the same class which must a data sound class; all sub-sounds (=channels) are played and displayed parallel; with two channels you have traditional stereo sounds

algorithm - collection of sounds; with no restrictions to the classes of the sub-sounds; they are not necessary for building process connections but the algorithm display helps to do that by an graphical interface; processes started on algorithms are refused

sound processes - do operations on signals or generate some new; imagine that they sound like the signal they produce, e.g. an echo process sounds like its input signal + echo effect; that's one of the main ideas that makes Assampler so powerful

Variables

Every sound has some attributes, called variables. A set of predefined variables is created with every sound and may be extended by the user. The predefined variables are used by Assampler directly, user defined variables may be useful for part calculations and as frontend for self-made algorithms.

Variables belonging to a sound are called values. If you selected a sound somewhere in a sound selection pop up gadget you can override values of the selected sound by parameters only valid for this call. Compare it with calling procedures with tag lists in programming languages where some attributes should be changed to non-default states. This method allows you to reuse the same sound again and again with slightly different attributes. For e.g. the same sound created for use as an instrument, can be requested with various frequencies and the results can then be mixed to a chord. Although this technique is very powerful it should be used rarely, because it can make things very confusing.

Another important technique is the referring of variables. It allows to adopt variable contents of other variables. This was implemented to be able both to get variables from an algorithm's front end and to set multiple variables to the same value. The normal referring is available for all variable types when switching the cycle button on the left side of a variable's gadget. More

possibilities offer the expression variables, there you can use sound/variable paths to refer to other variables and to operate with them.

Expressions

Expression variables processes complex numbers, real SI units and some physical constants. The most important can be seen in the context menu of an expression gadget. The program really operates with and checks the units, so you have a simple scientific calculator when opening the calculator window - just right for a physicist.

Expressions are case-sensitive (because of some minor differences between mV and MV :-). Multiplication operator "*" can be omitted, "2 m" is in fact the product of the real number "2" and the unit "m" (metres)! Powers are calculated from right to left. Some special characters can be used ², ³ for powers, %, ¼, ½, ¾ for factors.

Sound related functions (sound characteristics) can be qualified by a path. Paths are built similar to these of MessyDOS (sorry, but the slash "/" is already used for division). Backslash "\" for separating sound names, point "." for separating variable identifiers, two points ".." for going to the parent sound, underscore "_" for refering the destination sound of a calculation. Example: "..\Osci.Frequency" specifies the variable Frequency of the sound Osci which is child of the parent of the current sound (so it is probably a sister sound ...).

Signals

The stuff that is interchanged between the sound processes are signals. Assampler treats them as infinite streams, if a length is required (e.g. waveform for oscillator), an expression variable of the name Length must be added to this sound. The Length is only visible for the direct calling process, other processes will simply ignore the Length - the signal remains infinite!

In the theory the signals are continuous, but in reality they must be quantified. So every signal has basic characteristics: Sample rate (SampleFreq), begin of signal (Begin), maximal elongation (Volume). The units of them are free choosable, and are only specified by the values of the used sounds. E.g. Begin of an exponential process determines the begin, Volume the volume of the produced signal. SampleFreq is not determined in this case. In case of data sounds all signal attributes are determined.

Quantification and fitting sounds of different sampling rates is done by Assampler automatically. It does so with the Volume.

GUI

Sound Creation

Because of the object oriented nature of Assampler there is only one function to load, one to save and one to create a new sound, no matter if data sound, sound group or process! But there are different ways to call these functions. Load/Save can be done from menu or from main window. Creation can be done from the classes window, either by double clicking in the classes listtree, or by dropping an entry to a group display, or by use of one of the Create-Buttons.

Windows

display windows - can be opened in any number for every sound; shows the contents of the sound; all cut/process/play/save/load actions are done on the active display window (this may confuse you, because activating the information window of a sound doesn't help); display windows are opened by default for data sounds and sound groups

information windows - can be opened once for every sound; holds the values of a sound; here you can define new variables by yourself; information windows are opened by default for processes

Drag&Drop

Drag&Drop and Context menus are the GUI elements that cannot be seen, so I give you some hints where to try to find some further functionality:

sources:

workbench icons, sound select pop up gadgets, string gadgets, variable labels, variable list entries, variable array entries, group display entries, algorithm display boxes, classes listtree entries, sound class' default values storage in the classes window, pen adjust gadget in preferences window

targets:

display windows, sound select pop up gadgets, string gadgets, variable lists, variable array's add/delete buttons, group displays, algorithm displays, algorithm display boxes, classes listtree, main value table in the classes window, sound class' default values storage, pen adjust gadget in preferences window

Context menus

expression string gadgets, sound select pop up gadgets, spectrum display, spline display, group display, sound boxes in algorithm display

Zoomable scrollers

The scrollers in data sound displays can be zoomed when you hold the left control key (configurable) while dragging the knob or clicking in the free space left and right of it.

First steps

How to get the examples running

At first you must load an example algorithm to somewhere or you create a new process (producers are preferred, as long as you don't have a sound to manipulate)

Then you can render it into a sample. So create a sampled sound with a sample frequency and length you like (five seconds will fit for most examples). Then open the information window for the example algorithm/process, check if the display window of your sampled sound is the active, then press start in the algorithm's information window.

Algorithms that use expression variables with _.Index in it, should be started on groups of sampled sounds. If _.Channel is used, start it on stereo groups. The underscore means the destination sound of every part calculation and the integer variables Index or Channel are inserted for every destination sound and contains the position within the sound's superior group. This allows to start several calculations with different values at once (see example MakeMultipitch, FilterBass0, StereoEcho0). The first sound of a group gets the index 0. The Index variable is only available during a calculation.

Another possibility is to play a sound directly. If applied on algorithms a strong Amiga (a 68030/50MHz don't allow a bit more complex algorithms) is required. You must open the display window, then hit the play sound button in the main window. To find out, what your Amiga lacks of performance, you can render an algorithm, then look at the required calculation time (threads window) and compare with the length of the rendered sampled sound. Up to 50% of CPU load is ok, more should be avoided (= better don't set the value higher in the Assampler preferences).

How to create algorithmus by myself

You need not to create an algorithm for every calculation. Start with one process (e.g. oscillator), created by double-clicking in the classes tree. Then create another for input or modulation purposes (e.g. exponential), and drag the sound select gadget from its information window to the modulation variable gadget in the other window. Now the oscillator will be modulated by the exponential curve.

For more complex buildings an algorithm is advised. You can drag processes directly from the classes tree to the algorithm's display to create and place them. Then you can connect everything. The next step is to double-click on one process box to get its information window. Here you can change some values.

Creating a frontend for this algorithm should be made in two phases. First open the information window of the surrounding algorithm and from there the variables window, then drag the labels of the sub processes variables you want to see in the front-end there and close the variables window. In the second phase you travel through the sub-processes activating the reference of variables that can be found in the front-end and drag the variable label from the front-end into the reference path string gadget. At last select the sub-process that determines the algorithm's output in the variable "Output".

That's all for now. Hopefully this was a good entry and the rest is intuitive enough to be recognized by yourself. Ask me if something is too strange.

1.8 expression

1.9 oscillator

An example from a former time when I thought I had enough patience to write a German and an English guide.

Function

An oscillator is used to produce tones - the basic element for melody instruments. You may use any other sound as waveform.

Values

Frequency

Frequency of the tone (usually in Hz = waves per second). The higher the value the higher the tone. Note that frequencies (that means sine waveform) below 50 Hz or above 20 kHz cannot be heard by humans. Another limit is the half of the sample frequency because of the sample theorem.

Useful is the Tone-function to get an specific tone or an expression like 1/length for usage as LFO.

Phase

Changing the phase allows a shifting of the wave along the time axis. The recommended range is 0 - 2Pi rad or 0 - 180°.

With phase set to zero the waveforms are odd functions (in case of the default waveforms except noise). That means they can be dissected in harmonic sine waves leaving out cosine (=even) parts. Therefore they are similar to the sine wave and can be exchanged easily. Setting the phase to Pi/2 rad or 90° produces cosine like waves.

The phase does not have any influence of how the static tone sounds. But you may need it if you produce phased sounds. You achieve this by mixing the output of two oscillators with slightly different frequencies. You will hear the typical phasing effect. Changing the phase of one of the oscillators allows to adjust phase in which the effect starts.

Waveform

Select the waveform here. It is useful to take some of the default waveforms in the "Waveforms" group or a spline sound for freehand editted waves or a spectrum sound for harmonic synthesized waves. You can even use processes e.g. for filtered impulses or noise.

Interpolation

The oscillator may need some samples between the really sampled values. It can take than the previous sample value (no interpolation) or an average value (linear interpolation) or an approximated value received by an cubical spline. The more complex interpolation the better results and the slower calculation.

It is recommended to use no interpolation for high tones (melody instruments), linear interpolation for low tones (bass instruments) and cubical interpolation for non-audible low tones (usage as LFO).

Round

This eliminates an ugly noise when working with low sampling frequencies. Selecting it detunes the requested frequency so that the period of one wave fits to an integer number of sample values.

Technical

The wave sound will be completely converted in sample data first. This implies that no modification by parameters is possible during oscillating. This simplest kind of oscillator is restricted to static tones!

Avoid sounds that need much memory because the waveform has to be placed in one memory block. Especially with spline sounds or processes you can loose track of memory usage. Normally the memory requirements of a spline sound don't depend on the spline's sample frequency (because it is not sampled) but here it is important. An spline of ten seconds with 16726 Hz replay rate takes 162.76 KB of memory - the default waveforms take 256 bytes.

Theory

1.10 Changes

04.10.1998

- added UniFilter (universal filter for high pass, band pass, low pass and band limit) which allows a lot new effects, examples Vocoder, FilterBass, Wind make use of it

- fixed bug that could have caused the 00...04 guru, added lots of debug output to the play routine in case it wasn't its fault

23.02.1999

- added FPU support to the synthesizer.library that will not satisfy you. 1. It does only influence the floating point based routines like the control curve processes and the universal filter. 2. Due to the overhead the FPU routines are sometimes slower than my self-written Non-FPU routines.

- modified internal handling of variables

- changed the meaning of Depth with linear modulation scale in several processes. Depth is now an absolute value (no longer relative to the base value).

- custom non-recursive filter is now able to process big filter windows (but it's still slow). Now you can use it to give a sound the room characteristics of any room where you have an impulse response from. Reverb0.algo demonstrates that.

- added mapping process

- fixed some ugly bugs, added some new :-)

06.03.1999

- the amplifier can now remove an envelope from a sound (inverse switch). Actually the input is divided by modulation and multiplied with volume. Do you find this senseful, or should the volume also be divided (which means input/(modulation*volume))?

23.03.1999

- some new processes: Clip, Concatenate, Differentiate, Integrate

- fixed some bugs regarding chaining of processes

- added a bug provocation which shall show hidden errors when writing to sample sounds (which is the case if you start a process on a sample sound)

- now distinction between sub-sounds of list groups (Index) and channel-sounds of stereo groups (Channel). That means it is now possible to distribute one stereo sound over a list of stereo sounds, what was not possible before.

06.04.1999

- fixed bug in spectrum display

10.07.1999

- localization almost ready

- interesting new examples: WobbleEcho (only useful when rendered into a sampled sound), Glitter, Helicopter

21.07.1999

- prepared to the first public release

09.09.1999

- fixed some crashes, deadlocks and other bugs

- completed German guide

- interesting examples: Reverberation2, RandomMelody

1.11 Author

henning.thielemann@student.uni-halle.de

http://home.pages.de/~lemming

1.12 Credits

Assampler takes full advantage of these projects (an important point of what I consider as efficient programming and the way of Amiga programming in general):

function libraries - the things you will use, when starting Assampler

MUI (MagicUserInterface), Stefan Stuntz, user interface

MCC_Listtree, Klaus Melchior, Mui-Custom-Class to display tree structures

AHI (AudioHardwareInterface), Martin Blom, device independent sound input and output

datatypes (V45), Christian Buchner/Roland Mainz, Import/Export in several file formats

ReqTools, Nico François + Magnus Holmgren, requesters

Cluster-OFunctions.mod, Viona, formula interpreter

iffparse, AmigaOS3.1

SearchGuide, LSAGS, search routine for AmigaGuides

development environment - the things I developed Assampler with

Cluster, Viona/MOM, programming language (excellent extension of ModulaII)

PhxAss, Frank Wille, Assembler (the right spelling for this word :-)

ADis, Martin Apel, Disassembler

KingCon, David Larsson, "only" a Console handler, but simplifies programming work a lot

and much others ...