in

COLLABORATORS			
	TITLE : in		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		June 25, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

in

1.1 SamEd, sound sample editor, © 1998 Matthew Hampton

```
                    __
     //\AMED
     \_-
     _-\
     \//ound sample editor.
      --
```

A new 16-bit stereo sound sample editor designed for easy use, featuring a versatile developers interface for an expandable system.

Please note: Procs and Filers from this version are incompatible with those supplied on Amiga Format's disc 32. See

1.2 Introduction to SamEd...

Introduction to SamEd sound editing software...

SamEd is a 8 / 16 bit, mono / stereo sound sample editor, designed to be easy to use and yet powerful and most importantly expandable. To fulfil this design SamEd has the following features:

· Uses AHI to be compatable with the majority of sound hardware;

· Uses MUI for several reasons including to speed up development, to supply a customisable and scalable interface, and eventually to allow dynamic creation of objects;

· Has only one edit window. Although I consider this a limitation, it does mean that it is easy to see what is happening as all focus is directed to the one display. You only edit the sample you can see;

· The (theoretical) maximum sample size is over two thousand mega bytes long, although I have not tested this and the ranging in the waveform window will not cope with these sizes.

· Copes with 8 / 16 bit, mono / stereo samples without any problems (I hope :)), and may soon support 32 bit floating point samples, as well as samples on any device, eg. hard disks.

· Loades and saves samples through filers. For example the 8SVX filer loads and saves 8SVX type IFF files. More filers can be produced for different file types. RAW samples are loaded internally by SamEd.

· A powerful 'external process' launcher allows any developer to produce his or her own sound effects. The 'external process' system has a quick and easy interface to allow programmers to get data from any sample and alter it as they wish. This system relies on multitasking and creates a multithreaded editing system.

The requirements of this program are:

· AHI;
· MUI;
· AmigaOS 3.0;
· about 200K chip and 640K fast RAM (based on figures from my title bar). This will increase as you load samples and effect windows;
· I recomend you have a 640 * 400 srceen, and a few MB of fast RAM.

Anything else? -if you have problems with a particular setup then please fell free to moan.

1.3 Registering SamEd...

Please register to support development...

If you wish to register SamEd, and get a personal key file allowing you

to use all updated versions of SamEd up to at least V2.0, then send your name, address, and any other relevant information (telephone no., email addr.) and £10 to me (address below). You will recieve a key file, all the latest sound effects, and the very latest version of SamEd. You will also recieve any major updates after that.

If you have any comments, complaints, registrations, bug reports, ideas, things you wish were implemented etc. feel free to write to me:

54 Myatt Road,
Offenham,
Evesham,
Worcs,

WR11 5SD

(England)

1.4 Installation...

Installing SamEd software...

To install copy all files to a new directory on your hard disk.

Further information:
SamEd looks for files in the current directory. It currently looks for filers in 'Filers/', and external processes in 'Procs/'. All libraries should be accessible through the LIBS: assign.

1.5 Starting SamEd from Workbench...

Starting SamEd from Workbench...

Obviously, to run SamEd double click on its icon. One tool type is supported by SamEd: OUTPUT = <file>. This redirects SamEd's debug information to a file. The file name could be CON:, RAW:, or KCON: etc. The debug information is usually memory allocations and messages recieved from external processes, but output from filers will also be directed here.

(v0.80 - output for ext. procs. is not yet directed to OUTPUT. Ext. procs. are currently opened with no input/output streams.)

1.6 Starting from CLI...

Starting SamEd from CLI, Shell, etc...

To run SamEd type SamEd (or SamEd#.##) into a CLI or Shell. SamEd's debuging information will automatically be outputted to the console. To stop it redirect it eg. SamEd >NIL: or SamEd >Ram:samed.debug.

SamEd supports the following arguments. First two arguments are scanned for:

NOINTF - will start SamEd without opening the MUI interface. You can load and play samples automatically from the CLI with more commands.

ICONFYFIRST - opens SamEd in an iconified state (ie. only an icon will apear on the desktop which you can double click to open the interface).

The following commands are then executed in the order which they appear:

CLOSEINTF / OPENINTF - Open or close the interface (all windows exept windows opened by external procs.).

ICONFY / UNICONFY - Iconify or uniconify the interface (all windows exept external procs.).

PLAY <n> - Play sample number <n>.

PLAYONCE <n> - Play sample number <n> once (ie. ignore loops - at present loops are ignored anyway).

PLAYLOOP <n> - Play the loop part of sample <n> only (activates playonce in current version).

PLAYWAIT - Wait until the sample has finished playing.

WAIT <ticks> - Wait <ticks> amount of time. The time is measured in 'ticks' or 50/60 ths of a second (ie. 50 ticks = 1 second).

FLUSH <n> - Flush sample number <n> from memory.

FLUSHALL - Flush all currently loaded samples from memory.

LOAD <n><name> - Load sample named <name> into sample number <n>

QUIT - Quit SamEd. SamEd may be running even though you cannot see it because it has no interface.

Any arguments or parameters for arguments not recognised will be treated as file names and SamEd will attempt to load the file into the first available sample slot.

Please note you can only load SamEd once, but you can send SamEd commands

while it is running. Eg. type SamEd, Then SamEd again with some of the above arguments, and they will be executed (but NOINTF and ICONFYFIRST will be ignored). Because there are wait commands included, and commands are executed in order this allows you to produce a simple 'script', especially if you hide the interface.

1.7 Editing Samples...

Editing samples and general usage...

Sorry, not completed :(

1.8 The different sample types...

The four supported sample types...

There are four sample types supported by SamEd:

8 bit mono
8 bit stereo
16 bit mono
16 bit stereo

As you may know 16 bit samples are generally of better quality than 8 bit samples but take up twice as much memory. This is due to them being 256 times as 'accurate'.

SamEd keeps note of each sample's type from when it is loaded, so that it may play back the data (through AHI) correctly. When you load a RAW sample (ie. a sample with waveform data but no pitch, volume, loop data etc.), you need to specify the samples type. The type window opens automatically, change the type then select OK. If it does not play back correctly try opening the type window again (Type button on main window), change the type, and press OK.

When you have found the samples type you will not need to use the OK button, in the type window, any more. If you wish to change the samples type use the convert button. This will not only set the right type but will also convert the samples data to the correct format.

Note that a sample may be loaded RAW even if it is not actually RAW. This happens when a sample format is not recognised. When this happens there may be some noise at the begining of the sample which you can easily delete.

1.9 Exec ports and messages...

Basics to writing effects for SamEd...

To develop for SamEd you will need to understand Exec Tasks, Ports, Messages, and Signals, at least in part. It is possible to create an external

process (effect) using any programming language which supports these objects. For example Blitz Basic has access to the Exec library.

I'll run through what is involved in creating an external process. The ideas are simple if you understand messages and ports. First what does SamEd do:

· First of all SamEd sets up a Message port which it listens to continually.

· If the user presses an ext. procs. button then SamEd launches it as a separate task passing it it's button number as a CLI type argument.

· If the ext. proc. decides it needs some information from SamEd (any thing from the version no., to sample data, or setting up a new signal), The proc. sends SamEd a message which SamEd returns with the data filled in.

Now what the ext. proc. must do:

· You will need an initialised message port for replies, a message, and possibly a signal.

· First, when the ext. proc. starts check SamEd's version number. This is done by sending a EPC_VERSION command in the message. This is only needed if you require a command which requires a certain version of SamEd's message port.

· Next if your proc should be allowed to be loaded multiple times then send an EPC_MULTI. If you don't then there will only ever be one copy of your program running at any time.

· When you need sensitive data (requiring a lock) obtain SamEd's lock by sending EPC_LOCK and CHECK FOR ERRORS. SamEd will return an error if it is already locked. Data requiring the lock includes sample data, and ranging data, etc.

· Obtain a pointer to the data. You may then alter it safely with no interference. Eg. apply the sound effect to the sound.

· It is important that you now unlock SamEd.

· When you quit, if you didn't send EPC_MULTI, send EPC_QUIT so that your button can be reused (your program can be launched again).

If you have to set up the mesage 'manually' each time then your written code will become long. However, if you are using C there are some functions to aid you considerably.

Using C
C functions

1.10 Writing external processes in C

Writing 'external processes' (effects) in C...

Magic numbers - SamEd now uses 'magic numbers' so ext. procs. and filers know that (a suppported version of) SamEd launched them.

Writing 'external processes' in C is now very easy thanks to some useful functions. These functions try to hide the long winded process of sending messages to SamEd. This includes setting up a message port to send and recieve messages and sending messages with all the data in the message properly initialised. SamEd's message structure looks like this:

```
static struct Ext_Proc_Msg {
    struct Message  epm_Msg;
    ULONG           epm_Command;
    ULONG           epm_Error;
    APTR            epm_Data;
};
```

To use it manually you need to set up the message epm_Msg, set epm_Command = EPC_#, set epm_Data to point to the required data, and then send it. Then you need to check epm_Error, before reading the contents of epm_Data. This is almost all replaced by the EP_SendMsg() function.

I'll now run through an 'external process' and the functions it uses:

· When your ext. proc. is launched it is passed three arguments. argv[0] = your button and program name; argv[1] = PROC_MAGIC which you should check to make sure SamEd launched you; And argv[2] = your button number (ULONG).

· Set up a new port using EP_NewPort(). This will not only set up a new port but will set up other data for you, eg. you pass it your button number for use with EP_Quit().

· You could now allocate a quit signal (SamEd will signal you when it quits so you can also). This is done by sending EP_AllocQSig(). You must pass it the EPP_HANDLE, which was returned by EP_NewPort(), and it will return either the allocated signal number or -1 indicating a failure.

· Next you could set up an interface (MUI ?), and wait for user input. You will need to wait on a combination of you signals and the quit signal if allocated. Remember to make the signal into a mask,
ie. sig_mask = 1 << sig_num.

· If the user selects an action which requires sample data (etc.), you will need to aquire the lock to SamEd using EP_Lock(). This function returns FALSE if there was an error, or TRUE if the lock was aquired or if you already hold it.

· Now you can get a pointer to the data to alter it. The easiest way to accomplish this is to use EP_GetSample(). This function will fill in the pointers you pass it. So if you pass a pointer to a sam_info structure as the 'sample' parameter, then this will be set to point to the current sample.

· You now have exclusive access to the data. See extproc.h. You can free the sample by using FreeMem (sample->waveform, sample->length); Then set the sample length to 0: sample->length = 0; This is the standard way to show an empty sample. To create new sample data, use
sample->waveform = AllocMem (new_length, MEMF_CLEAR | MEMF_PUBLIC);
Remember samples can be of different types (8/16 bit etc.). Try to ignore samples of strange types. You can set anything in the sam_info structure except file_type, and please leave dir and name pointing where they are set.

NOTE: length is always in BYTEs, loop_begin / _length is in sample frames.

· After altering data, you must unlock SamEd, as it is set to sleep while you hold the lock, using EP_UnLock();

· You can send any other command using EP_SendMsg(), and EP_SendIDMsg(). Both return TRUE if the message was sent, you must check for errors, ie. epp_han->message.epm_Error = NULL for success.

· When you need to quit send EP_Quit() to tell SamEd our button can be reused and to take us off SamEd's 'quit list'; Use EP_DeletePort to free the memory use by the message port; And finally use EP_DeallocQSig to deallocate the quit signal.

Further information:
SamEd stores the data for all samples in an array of sam_info structures. Currently there is a fixed number of 65 samples. sample[0] is the copy buffer. When you obtain the lock you have exclusive access to all of these samples. Please check for the maximum number of samples from SamEd if you plan on accessing samples randomly (ie. sample[num] where num may be picked by the user, don't assume 65 samples). Use
EP_SendMsg (epp_han, EPC_TOTALNUM, &max_sams);

See General.c and extproc.h for more information.

If you find any bugs or have any problems, don't hesitate to tell me.