slab

Claude Heiland-Allen

1999–2019

Contents

1 Docs/slab.md . . . 3
2 Install . . . 44
3 README.md . . . 45
4 SLab . . . 45
5 Source/add.e . . . 46
6 Source/amp.e . . . 46
7 Source/bandfilter.e . . . 47
8 Source/bandpass.e . . . 48
9 Source/bandreject.e . . . 49
10 Source/.build . . . 50
11 Source/cbuffer.e . . . 55
12 Source/classnode.e . . . 58
13 Source/cli.e . . . 59
14 Source/constant.e . . . 64
15 Source/container.e . . . 65
16 Source/copy.e . . . 68
17 Source/debug.e . . . 68
18 Source/defs.e . . . 69
19 Source/delay.e . . . 71
20 Source/e2txt.script . . . 72
21 Source/e2txt_sub.script . . . 72
22 Source/echo.e . . . 72
23 Source/effect.e . . . 75
24 Source/exp.e . . . 77
25 Source/fbdelay.e . . . 77
26 Source/feedback.e . . . 78
27 Source/file.e . . . 79
28 Source/filter.e . . . 81
29 Source/hack.e . . . 83
30 Source/halfrectify.e . . . 83
31 Source/highpass.e . . . 84
32 Source/iff8svx_ff.e . . . 84
33 Source/in0out1.e . . . 85
34 Source/in1out0.e . . . 87
35 Source/in1out1.e . . . 88
36 Source/in1outm.e . . . 90
37 Source/inmout1.e . . . 93
38 Source/invert.e . . . 97
39 Source/kernel.e . . . 97
40 Source/link.e . . . 102
41 Source/link_.e . . . 104
42 Source/list.e . . . 104

43 Source/lowpass.e . . . 106
44 Source/main.e . . . 106
45 Source/mul.e . . . 108
46 Source/notch.e . . . 108
47 Source/osc.e . . . 111
48 Source/print.e . . . 113
49 Source/pulse.e . . . 114
50 Source/ramp.e . . . 115
51 Source/read8svx.e . . . 115
52 Source/read.e . . . 117
53 Source/readslab.e . . . 118
54 Source/rectify.e . . . 120
55 Source/rnd.e . . . 121
56 Source/scale.e . . . 121
57 Source/sine.e . . . 122
58 Source/slab_ff.e . . . 122
59 Source/split.e . . . 123
60 Source/string.e . . . 123
61 Source/testcbuffer.e . . . 124
62 Source/testrnd.e . . . 125
63 Source/toparam.e . . . 126
64 Source/triangle.e . . . 128
65 Source/value.e . . . 128
66 Source/vox.e . . . 129
67 Source/whitenoise.e . . . 131
68 Source/write8svx.e . . . 131
69 Source/write.e . . . 133
70 Source/writeslab.e . . . 134
71 Source/zfilter.e . . . 136

1 Docs/slab.md

```
# SLab

A System for Processing Digitally Sampled Sound

## Analysis

### Background to the problem

My main hobby is writing music using my computer and synthesizer. The
computer is at the centre of the system, controlling the synthesizer and
recording and playing back sound. The computer records sounds
digitally, and the resulting stream of numbers can be manipulated to
alter the characteristics of the sound.

I currently use several programs to process sounds, but I find that none
of them are flexible or powerful enough. Therefore for this project I
will create a system for processing digitally sampled sounds.

In a music studio different pieces of equipment like synthesizers and
effects processors are self-contained units which can be connected
together in different ways with cables carrying audio signals in
```

electrical form. This method of producing music has been refined over the past fifty years and has been shown to be flexible, powerful and successful.

However, hardware is expensive, and separate units often duplicate functions. All early equipment was analogue, based on the properties of electronic components. Nearly all new effects processors perform their actions mathematically on sounds in the digital domain and therefore they contain analogue to digital and digital to analogue converters (ADCs and DACs). The signal is often converted many times between analogue and digital forms in its path through the studio, which is both inefficient in terms of hardware cost and undesirable in terms of signal quality. Some effects processors, but still a small minority, have digital inputs and outputs but the small market implies a high price.

In the 1980s a new type of musical instrument became affordable. The sampler is essentially a digital record and replay system, potentially much more powerful and universal than traditional synthesizers, and although the first samplers only had enough memory for one or two seconds of record time they became popular for the ease in which sounds can be sampled and processed. Some popular computers could soon play back samples.

In an initially separate trend to the digitalisation of effects processors, the ownership and power of personal computers have risen dramatically. Coupled with the development of the musical instrument digital interface (MIDI) standard, originally developed so that notes played on one keyboard could trigger sounds on another synthesizer, allowing computerised recording and editing of performances, computers began to be found in music studios. At first the computers were used only for sequencing (recording and editing control data) but soon sampling and hard disk recording became popular.

At this point the final vision became clear. It would eventually be possible to have nearly all of a recording studio in one box, bar the microphones and loudspeakers. Instead of connecting hardware units with wires, audio data would pass through processing software, entirely digital and therefore with no intermediate signal degradation and interference.

Research

Sound waves are continuous changes in air pressure, which microphones convert to varying voltages, but computers can only deal with discrete numbers. The continuous input signal is converted to a stream of numbers by sampling and quantisation. Sampling records the value of the signal at specific time instants, and quantisation converts these continuous values into numbers with finite precision. Once sampled, the sound data can be manipulated in many ways, for example changing the relative amplitude and phase of frequency components (filtering) or reducing changes in volume (compression).

Much of signal processing is based on rigorous mathematical foundations, which provides efficient algorithms for carrying out modifications and explaining how the data is being modified. As this information is rather technical, it will be considered in an appendix.

Existing software

I use several existing application programs to process sound samples. Most of these have very similar user interfaces, thus sharing both its advantages and its disadvantages.

Typically a large part of the screen is occupied with a graphical waveform display. Dragging with the mouse in this section of the display marks a range on which following operations are to be performed. Controls also exist to zoom in to look more closely at a particular portion of sampled data. The use of a graphical display allows the user to identify different parts of a long sound, for example bass and snare strikes in a drum part or different syllables in speech.

Below the waveform display are the most commonly used functions, such as controlling the display and playing back and recording sounds. Common range operations are also available, the usual cut, copy and paste found in almost all application software. Readily accessible, it makes simple editing of sounds quick and easy.

Less commonly used functions tend to be hidden away in sub menus, and as each operation simply replaces the marked range with the processed version it can be hard to combine different effects easily or perform the same operations on many different sounds. An unconnected problem is that often the parameters of the various effects are obscure, with values not connected to the real world of sound.

OctaMED Professional

v6.00o © Teijo Kinnunen and Ray Burt-Frost (1995.11.11)

OctaMED was developed from MED, a program originally created to allow programmers to create music for computer games. The program became much more advanced, the "Octa" part of the name coming from its ability to play eight recorded sounds at once through only four hardware sound c hannels. In addition to the tracker editor (for writing music) OctaMED has a sample editor, so the sounds used in compositions can be modified without having to use other programs.

The sample editor is of the type described above, with various windows and menus.

The interface is satisfactory for simple modifications, but the units used are not directly related to the sound. For example, the echo rate is the number of samples between echoes, so to create a specific time delay one has to perform calculations with the sample rate. The filter window has two parameters, distance is the period of the frequency to be filtered (so again the sample rate is involved if one wishes to filter a certain frequency) and averaging determines the strength of filtering. The filter window also provides access to the boost command.

OctaMED SoundStudio features control through ARexx, a system of passing textual commands between applications. This would enable batch processing scripts to be created.

Unlike many audio applications for the Amiga range of computers, OctaMED has a user interface consistent with the rest of the operating system.

This is desirable as it is easier for a user to use a familiar interface style than to have to learn different symbols.

Bars and Pipes Professional

v1.0e © The Blue Ribbon Soundworks Ltd. (1991, 1992)

Bars and Pipes is a MIDI sequencer, designed to record, edit and replay data from keyboards and synthesizers. This aspect is not relevant to signal processing, however Bars and Pipes has a powerful system of tools controlled by a graphical user interface. What makes Bars and Pipes special is its system of "pipes" and "pipe tools". The flow of data from input to standard track storage to output is represented analogously to the flow of liquid through pipes, and processing blocks can be inserted to modify or reroute the data.

Tools are dragged from the toolbox window to the pipe using the mouse. Tools are shown with symbols, but the '[?]' icon at the start of the toolbox gives a list by name. A tool in place can be moved left or right in the pipe or deleted. Tools with more than one output can be connected to a tool with more than one input. Double clicking on a tool calls up its parameter window, from where the tool can be controlled.

Tools present in this distribution of Bars and Pipes (in order from left to right in the toolbox window shown above) include: Branch, Counterpoint, Echo, Invert, Keyboard Split, Merge, Modulator, MIDI In, MIDI Out, Quantize, Transpose, Triad, Flip, Loop, UnQuantize, Phrase Shaper, Sforzando, Subdivider, Spare Keys, Accompany B, Articulator, Doctor of Velocity, Easy Off, Elbow, Feedback In, Feedback Out, Harmony Generator, Note Filter, Plug, Reverse, Stop!, and Velocity Splitter.

There is developer information available, so new pipe tools can be created. The system is designed in such a way that the main program does not need to be recompiled, the tools are separate object code files. C is used with clever coding of object oriented techniques, presumably because it was the standard operating system language and because C++ was not widely available.

AmiSOX

v3.3 (1994.02.28)

SOund eXchange v6.11 for Amiga

Created and maintained by Lance Norskog (thinman@netcom.com), Amiga port by David Champion (dgc3@midway.uchicago.edu).

"SOX is intended as the Swiss army knife of sound processing tools. It doesn't do anything very well, but sooner or later it comes in very handy."

AmiSOX originated under the UNIX operating system as a universal sound file format translator. UNIX is a text-based OS, and AmiSOX is command line driven, although a separate graphical user interface is available.

Only one effect may be applied at once, for multiple effects a pipe may be used.

```
Scripts are useful for hiding options, for example "X2Y file" contains
195 the command for converting file.X to file.Y.

Command line syntax:

```
200 sox [ options ] [ format ] infile [ format ] outfile [ effect [ fxopts ] ]
```

Options include volume change relative to 1. Format specifiers define
either recognized types (sample files with header data) or raw data
205 (file contains no information about data so it must be given).

Effects in AmiSOX

- `copy` (no effect)
210
- `rate` (resample at new rate (given by output format) by linear
  interpolation)

- `avg` (reduce number of channels by averaging)
215
- `reverse` (reverse entire sample)

- `echo [ delay volume ]+` (simple echo, delays given in seconds,
  volumes given relative to 1)
220
- `vibro speed [ depth ]` (volume tremolo, speed given in cycles per
  second (less than 30), depth given as fraction of full modulation
  (less than 1))

225 - `lowp frequency` (gentle low pass filter at frequency given in cycles
  per second)

- `highp frequency` (gentle high pass filter at frequency given in
  cycles per second)
230
- `band [ -n ] freq [ width ]` (band pass filter between f - w and
  f + w, frequency and width given in cycles per second, the manual
  states that "the default mode is oriented to pitched signals, the
  alternative -n (noise) mode is for unpitched sounds; noise is
235   introduced in the shape of the filter")

- `stat` (list statistics about input file, no output file is generated)

The Sound Tools Library
240
Previously on alt.sources, now on comp.sources.misc. This is the C
source library for sox containing the protocols for components (like
file handlers and effects) to interact, and effects algorithms. Full
developer material is included so new formats and effects can be
245 implemented, made easier by skeleton drivers. Internal data is signed
32-bit integer.

SOXGUI
```

v1.2 © Stephan Klein (1995.05.25)

This is a basic graphical interface to AmiSOX. SOXGUI is an easier way of specifying the command line for AmiSOX, using the mouse to select options. SOXGUI then executes AmiSOX with the command line it generates. SOXGUI is very simple, but effective. The interface is self explanatory, the large "AmiSOX!!!" button starts AmiSOX.

ScreenX

v3.0 © Steve Tibbett

This utility program saves the screen image to a file when a key combination is pressed.

Deluxe Paint III

v3.25 © Electronic Arts (1985, 1990)

This is a graphics program which I used to crop the images saved by ScreenX.

Black's Editor

v1.01 © Marco Negri (1996.04.01)

This is a text editor which I used to type much of the project.

System Requirements

The system needs to function in a similar way to real music studios, that is, processing units are connected with paths for sound and control data.

There should be a variety of processing units available, some modelling complete audio effects devices (for example an echo unit) and others acting as simpler building blocks (for example a delay unit).

The system needs to read and write sound sample files in IFF 8SVX and RIFF WAVE formats. These are the types of files I use most frequently, and other formats can be converted easily using SOX or some other such program.

The system needs to be portable, that is, easily adapted to different computers and operating systems. The system will be created for the Acorn Archimedes computer running the RiscOS operating system, but I create music on the Amiga computer running AmigaOS. As most of the implementation will be mathematical, portability will only be an issue for certain parts of the system.

The system needs to be controlled by textual commands, with the possibility of batch script files containing many commands to ease repetitive operations. The grammar of the commands should be as simple and general as possible, whereas the vocabulary needs to be able to expand to include new types of processing unit.

Hardware and Software Requirements

The system is unlikely to require any particular software to run, as the interface is textual. Textual interaction with the user is supported within C++, so the software may need to be simply recompiled for a new operating system.

Hardware requirements are unlikely to be specific. The system will not support real time processing because this requires specific hardware, so speed is not absolutely critical. However, the faster a computer is the more complex routings and effects it will be able to perform without the user waiting a long time. The script facility will enable users of slow computers to leave the computer performing complex processing while they do something else.

Design

System Architecture

There are three main elements of the system, these are the command line interface, the effect algorithms, and the management kernel to link them together.

Effects

Effects processing is the aim of the system, so this section will be discussed first.

There are two main strategies for processing sampled data, each with variations.

Simple programs, like OctaMED's sample editor, store the entire sound in memory, along with a spare memory buffer. Each effect works in its own way, reading directly from one data array and writing to the other.

More refined programs, such as AmiSOX, can handle files larger than available memory by splitting them into smaller blocks. This forces the effects to work in a broadly time-ordered manner, starting at the start of the sound and working through to the end. AmiSOX developer material can be found in the appendix.

These two "ad hoc" methods are rather inflexible. To use more than one effect, multiple runs of the programs are required. A pipe allows serial chains to be created relatively easily but parallel routing is more awkward.

However, what is impossible with this method is feedback between separate effects (although feedback within an individual effect is possible). This is quite limiting. For example, a chorus effect (consisting of several copies of the original sound superimposed at varying pitches) can be turned into a more dramatic flanger effect with the simple addition of a feedback loop. Feedback in this way requires that only one sample is processed at once. Without the flexible routing that single sample operation allows, for example, the flanger effect would have to be rewritten from the chorus effect, requiring much more work.

As each effect processes only one sample before passing it on, and the

vast majority of effects require information from more than one input sample to generate the output, coupled with the fact that each effect is likely to have parameters that can be modified, the most sensible method of solution is to have objects that contain both the data outlined and effect algorithm code and that can be linked together in many different ways.

There are still two alternatives within this method. Either the input sources (sound files, for example) push their data into the system, or outputs pull data from the system. The latter may possibly be better suited to real-time operation, but the former is conceptually closer to the real world, and may also be easier to implement.

Each effect object has things in common, for example it can have inputs and outputs along which sound data passes, and it can have parameters that affect the way the inputs are modified to form the outputs. Therefore there needs to be a class effect, a base class for all of the different effects which incorporates all of these facilities.

As new effect classes can be developed, there needs to be a way of identifying the various inputs, outputs and parameters of different effects. Text is sensible, as it is a natural method of communication, but it is slow to compare strings. Therefore, text should be used to obtain a more efficient identification code, for example a small integer.

Effects need to be linked together, but each input can be connected to only one output (which has only one input linked to it). Therefore each effect needs to identify the other effects it is linked to in both directions, so the if a different effect links to it, the original can be found to unlink it. A link needs to pass the output from one effect to another effect's input, identified by the id code described above.

As it is useful to make new effects from existing effects, inheritance is an appropriate mechanism. This means that certain methods of the effect class must be virtual so that they can be redefined. The three most important things that an effect does is get input, process it, and send output, so all of these must be able to be redefined to allow for the addition of new inputs, outputs and parameters. The parent class's function can be called within the redefined version, allowing the parent's attributes to still be present.

Sound data is passed through the system one sample at a time, to allow feedback. Some effect objects are designated as sources, so for each sample the process method of each of these is called in turn. The process method of each effect (unless it is a sink, having no outputs) call its output method, which calls the input method of the destination effect. The input method checks whether all of the inputs have been filled for this sample time, if they have it calls the process method for that effect object. In this way, data passes through the system from sources to sinks.

The effect's input method needs to store the input sample within its data space, as well as recording that the input has been set this sample. To allow for inheritance, the input method is called with the input id and the sample. If the id is not recognised then the parent's input method should be called to deal with it.

The process method takes the input samples, then modifies them and calls the output method. The process method can be a complete replacement of the parent's process method, but it may also call the parent's method if it is simply adding some extra functionality.

The output method takes an id and a sample, and if the id is not recognised then the parent method is called. The output method finds the corresponding output link, allowing the data to be passed to the next effect.

Sink effects have no outputs, for example writing the sample data a file would not be counted as an output here because it is not an output to another effect.

After processing the inputs need to be cleared before the next sample. To allow for feedback, however, the inputs must be cleared before the output method is called, in case this causes input to be given to the effect in question. It is impossible for an infinite loop to occur with feedback like this, because an effect getting feedback must have at least one input not in the loop, otherwise the loop could not be started. The output can be fed back to some of the inputs, but the others will not be filled until the next sample.

The individual effects algorithms are in an appendix.

Effects processing flow

```
effect::process()
        generate outputs, perhaps calling super::process(), and store in data space
        call clearinputs(), which calls super::clearinputs()
        call sendoutputs(), which calls super::sendoutputs()
            call output(), perhaps calling super::output()
                call destination.input()
                    if recognise input id
                        store sample data in data space
                        if destination.inputready()
                            call destination.process()
                    else
                        call destination.super::input()
```

Command Line Interface

The command line interface needs to be simple, so that it is easy to learn how to use, yet powerful enough to perform all useful operations. Simplicity can be achieved through use of similar syntax for different commands.

There are only a few lexical elements, which are the names of the commands ("new", "delete", "link", "set", "run"), and values for them: identifier strings for effects and classes, including "." to separate parts from objects, numbers (floating point), and character strings (for example filenames).

The commands can be expressed as a grammar, there are no control structures so there is no recursion to complicate matters.

The language tools LEX and YACC can be used to create efficient lexical analysers and grammar parsers from high level definitions, this saves effort and the resulting table driven programs are very efficient.

The commands and effect class and object names should be case insensitive, but case should be preserved, as some operating systems have case sensitive filenames.

Command line interface grammar

```
command ::= new    ; create new object
  | delete     ; delete an object
  | link     ; link an output to an input
  | set    ; set an object parameter
  | run    ; start processing

new ::= "new" new_type new_name
new_type  ::= string     ; the type of object to be created
new_name  ::= string     ; the name to give the object

delete  ::= "delete" delete_name
delete_name ::= string     ; the object to be deleted

link  ::= "link" link_source link_out link_dest link_in
link_source ::= string     ; the source object
link_out  ::= "." string     ; a named output
  | .    ; the main output
link_dest ::= string     ; the destination object
link_in ::= "." string     ; a named input
  | .    ; the main input

set ::= "set" set_name set_param set_value
set_name  ::= string     ; the object
set_param ::= "." string     ; parameter name
set_value ::= number     ; a number
  | string "." string ; an object with part specifier
  | "'" chars "'"    ; a character string

run ::= "run"
```

Kernel

The kernel has to link the command line interface to the effects. The command line interface calls kernel functions corresponding to the commands, with values converted to their correct form (for example, numbers converted to float values). The kernel finds the classes and effect objects corresponding to the textual identifiers, and calls the object methods that perform the command.

This separation of command line interface and kernel allows error checking to be simplified greatly. The command line interface has to deal with user input, which may be incorrect. However, the kernel has

only correct data to deal with, so error checking is redundant and can be removed when the system has been thoroughly tested. This is especially important for the effects processing section, because code here is executed very frequently.

Pseudo-code for kernel

Note that as some of the commands are the same as C++ keywords, the actual name of the corresponding functions must be different in the implementation (for example, use new_() instead of new()).

```
kernel::new(string type, string name)
        find node of type in class list
        if node not found then error, no such effect type
        if find node of name in effect list then error, already exists
        call node::new(effect list, name)
            create a new effect object
            create a new effect node
            link the effect node into the effect list

kernel::delete(string name)
        find node of name in effect list
        if node not found then error, no such effect
        delete effect node
            remove node from list
            delete effect object
                remove effect from processing network
                frees resources, close files and so on
kernel::link(string sname, string soutput, string dname, string dinput)
        find node of sname in effect list
        if node not found then error, no such effect
        find node of dname in effect list
        if node not found then error, no such effect
        find id of soutput
        if output not found then error, no such output
        find id of dinput
        if input not found then error, no such input
        create a new link object containing the objects and ids
        set sname's soutput to the link object
            delete existing link from the source
        set dname's dinput to the link object
            delete existing link to the destination

kernel::set(string name, string param, value val)
        find node of name in effect list
        if node not found then error, no such effect
        find id of parameter
        if parameter not found then error, no such parameter
        set parameter
            recalculate affected variables in effect object

kernel::run()
        while there is data left to process
            for all nodes in effect list
                if node is an input effect
                    call process() of effect
```

```
```

Effect Class Hierarchy

Inheritance leads to a hierarchy of different effect classes, each of
which is ultimately derived from the effect base class.

```
effect    ; base class
+-- in0out1   ; effects having only one output, "main"
|   +-- readfile  ; read from a file
|   |   +-- read_8SVX
|   |   +-- read_WAV
|   |   \-- ...
|   +-- constant  ; constant output, but set by parameter
|   +-- oscillator  ; generate a waveform, with parameters like "frequency"
|   |   +-- osc_sine
|   |   \-- ...
|   \-- ...
+-- in1out0   ; effects having only one input, "main"
|   +-- writefile ; write to a file
|   |   +-- write_8SVX
|   |   +-- write_WAV
|   |   \-- ...
|   +-- toparam ; sets a parameter of an object when the input data changes
|   \-- ...
+-- in1out1   ; effects having one input and one output, both "main"
|   +-- feedback  ; initialises a feedback loop
|   +-- delay   ; delays the input by a certain time
|   \-- ...
+-- in0outs   ;
+-- insout0   ;
+-- insouts   ; stereo versions, with "left" and "right" instead of "main"
\-- ...
```

Implementation

Development

Utility Functions

Several low-level data types are needed by the implementation.

Lists are needed to store the various effect objects and classes.
Doubly linked lists can be manipulated easily, only a few functions are
needed (addnode, removenode, findnode).

Circular buffers are needed by many effects, to store previous input
samples. A circular consists of an array with two pointers, one for
writing and one for reading. These are incremented simultaneously,
maintaining a constant offset between them. This allows a certain
amount of previous data to be stored, without having to copy the entire
buffer each time.

Changing the length during use should change the read pointer, for two
reasons. Firstly, it is better to have a "jump" in data now,

predictably, rather than at some point in the future. Secondly, it is desirable that any length, not just integers, can be used, using linear interpolation. The write pointer must be an integer to be able to write into a certain array element, so changing this would limit length changes to integer steps.

History

During the summer, before finally deciding on this project, I created a simple system for processing sounds, using QBasic. This system had severe limits on the size of files it could process, but it allowed effects algorithms to be tested. The system showed that it was feasible to develop a sound processing application.

- 1999.01.04 Implemented and tested class circularbuffer.
- 1999.01.05 Started implementing class effect.
- 1999.01.09 Joined all sources and headers into one large source file because I couldn't make Acorn C++ understand multiple files properly.
- 1999.01.12 Compiles without error only when removing const and initialiser from class member constants.
- 1999.01.14 Joined with LEDA vector and matrix classes.
- 1999.01.25 Tried to use LEX and YACC, but generated source would not compile. Started on state tables for class cli, table is currently global.
- 1999.02.25 Implemented class classlist and class classnode.
- 1999.02.25 Implemented class effectlist and class effectnode.
- 1999.02.25 Split file into many sources in preparation for using Make.
- 1999.02.26 Successfully made project. Had to edit makefile by hand to get it to link with the library files, the link options menu failed as couldn't write options.
- 1999.03.04 Only compiles if classnode::cli_new() is not pure virtual, even though it is never called.
- 1999.03.07 Implemented class effectin1out1, class fx_copy. Decided on naming scheme for classes: audio effect classes are fx_<effect>, class nodes are cn_<effect>.
- 1999.03.08 For some reason Make doesn't work interactively any more, have to add new files manually.
- 1999.03.12 Attempts to read a string from cin failed, which meant that the command line interface would be impossible.
- 1999.03.15 Abandoned Acorn C++.

At this point I decided to use the programming language E on the Amiga computer, which I had succesfully used for some other applications. I considered using BOOPSI (Basic Object Oriented Programming System for Intuition), however the advantages conveyed by using this (classes shared between applications, new classes can be created at run time) were outweighed by the disadvantages (one function has to deal with all methods), so I decided to use the inbuilt features of E.

- 1999.03.16 Implemented effect base class and linking routines.
- 1999.03.17 Implemented intermediate level effect classes to handle inputs and outputs.
- 1999.03.18 Implemented some basic effect classes to test the linking: constant, copy, print.
- 1999.03.20 Implemented kernel, the system can now link effects together and pass sample data from one effect to another (currently

```
  run() only runs for a certain amount of time).
- 1999.03.21  Implemented command line interface, using ReadArgs() means
  grammar is changed (uses " " to separate objects and parts instead of
  ".").
- 1999.03.24  Implemented command line command "set".

The use of sample data to control parameters is implemented awkwardly.
I realised that it depended on what order the effect objects were
created, whether parameters would be set before or after the sample at
that particular time interval was processed by the controlled effect.
A fix was added to ensure the setting of parameters was always after
processing, as in order to ensure it is before processing every effect
object would have to have a priority, and the whole sample routing
strategy would have to be changed.

### Testing

The three sections of the system need to be tested in different ways.

#### Command Line Interface

The command line interface has to deal with user input, which may be
incorrect.  Therefore the command line interface needs to be tested
thoroughly, to ensure that incorrect data is not passed to the other
parts of the system.

Each command needs to be tested, with and without valid arguments.
Random input should also be tested, to make sure that the interface is
resilient.  The tests should provoke every error response.

Once the interface has been tested for resilience, it needs to be tested
for ease of use and functionality.

#### Kernel

The kernel is always given data in the correct format, so method of
testing is different to the command line interface.  Here the testing
consists of verifying that the kernel functions as it is supposed to.
This can be done by checking that the result of each operation is
correct.  Compiler macros can be used to remove this extra testing code
from the final program, as it is not necessary after testing.

The common features of all of the effect classes can be tested together,
such as linking together and transferring sample data, as the classes
are largely similar.  However, many of the functions implemented in each
class are so small that they can be easily verified to be correct
without inserting special testing code.

#### Effects Processing

The effects processing classes need to be tested with real input, so the
quality of the results can be judged.  Testing of the code is only
necessary for the more complicated effects classes, like the z-plane
filters.  The speed of processing needs to be tested, both for simple
and complicated effects.

## Evaluation
```

The system performs as specified, except for a few minor details.

C++ was indicated as the language for implementation, but C++ is a strongly typed language, and there were too many problems in trying to implement the dynamic linking required by the system. This led me to abandon it, and use E, a programming language similar to Pascal, but with object oriented features. Currently E is only implemented for the Amiga range of computers, so porting the system to other platforms would be difficult.

The command line interface required some changes. To enable the use of the operating system function ReadArgs(), which provides powerful command line argument parsing support, the use of "." to separate effect object names and their inputs, outputs and parameters was dropped.

Commands (for example list) were added to the command line interface, to make using the system easier. These are documented in the user guide. The ability to set global parameters (such as how many samples to process) was needed, so this was incorporated into the set command.

The system is powerful enough to do just about any sound processing, but the command line interface can be awkward to use. The main problem is in keeping track of which effect objects have been created and what links there are between them, and the only way to show this is a graphical interface, which was ruled out as being too complex to implement.

Further Enhancements

As indicated above, a graphical interface would increase the ease of use of the system. Bars and Pipes, considered in the analysis section, has a graphical interface, but this can be awkward because the "pipe tools" are placed in rigid lines. Free placement of effects is essential. A variety of methods are appropriate for the various commands, for example "new" could allow the user to select the type from a popup menu, parameters could be set in a window opened by double-clicking on the effect object's icon, and the effect objects could be linked by dragging with the mouse held down from a region of one effect's icon to a region of another, representing the inputs and outputs.

The system could be altered internally to cope transparently with multichannel sample data. At present each stereo effect has to have its left and right connections linked separately, which is inconvenient.

If sufficiently fast computer hardware is available, new effect types could permit real-time processing of external input. This would require specific drivers for different computer operating systems, and there would have to be a way of checking that the computer was fast enough to cope with the input, because otherwise it may not generate the output samples before the next input arrives. Effects could be added to utilise extra hardware such as signal processing chips on sound cards.

More effect classes can be added easily to the system, but at present they are part of the main program. A plugin system, whereby new effects can be added without recompilation, would allow users and other developers to create their own effects. It is feasible that the system

could become a small part of a large music composition, editing and recording application.

More complicated effects can be built from existing simple ones, to the extent that an entire synthesizer could be simulated within the computer, built from various oscillators, envelopes and filters.

User guide

System requirements

The application requires AmigaOS v2.04 or greater.

Installation

To install the application, double click on the install icon. The installer asks you in which directory you want to install the application, and then copies all necessary files to that location.

Using the application

To start the application, double click on the application icon. A console window opens, in which you give commands. To exit the application, click the close gadget of the window with the mouse, or hold the control key and type "\".

The application is centered around effects objects, a concept similar to the different effects units found in an ordinary music studio. The commands create and manipulate effect objects. To process sounds, you create effect objects to read the sound from disk, process the sound, and write the new sound to disk. Then you instruct the application to perform the processing.

Command reference

This section describes all of the commands available.

‘new effecttype name‘

Create a new effect object of type effecttype. All the effect objects you create have to be given a name, so that you can refer to them later. The new effect object is initialised with default settings depending on the type.

You will be shown an error message if there is no effect type with the effecttype you specified, or if there is already an effect object with the name you specified (you can't have more than one effect object with the same name).

For an overview of which effect types are available see the effect reference.

‘delete name‘

Delete an effect object you have created earlier. You use this command when you no longer need an effect object, and want to get rid of it to free up the memory it requires.

```
    You will be shown an error message if there is no effect object with the
    name you specified.

    `link source.output destination.input`

    Link an output of one effect object to an input of another.  You use
    this command like you would connect cables between different effect
    units in a music studio, only here you don't have to scrabble behind
    racks of equipment.

    You will be shown an error message if the source or destination effect
    objects do not exist, or if there is no output or input with the name
    you gave in the source or destination object.

    To find out which inputs and outputs the different effect types have see
    the effect reference.

    Some linking can cause problems.  For example, you can't link an output
    of an effect object to one of its inputs, even via other effect objects.
    This is because an effect object needs to know all of its inputs to
    generate the output, but as it needs its output as an input it gets
    stuck before it can get started.

    Feedback (having output loop back as an input) can be very useful, so a
    special effect object type called "feedback" is available.  Simply
    create a new feedback object and link it into the feedback loop at some
    point.  Usually the best place to put it is just before the feedback is
    returned to the first effect object in the loop.

    `set name.parameter value`

    Set a parameter of an effect object.  Many effect objects have
    parameters you can change to change the sound of the effect.  For
    example, the "decay" parameter of an echo object would change the how
    quickly the echoes die away.  Different parameters take different
    values.  Most need you to type in a number, but some require special
    keywords, and some require a character string (for example a filename,
    like "MySounds:Voices/BigChoir.8svx", including quotes (")).

    Numbers should be entered normally.  You can enter both integers (whole
    numbers like 5 or -7) and real numbers (like 3.5 or -.01).  For very
    large or small numbers you can use standard form (also called scientific
    notation), in which the letter "e" (or "E") represents "multiplying by
    ten to the power of", for example -1e3 is equal to -1000, and 3.5e-4 is
    equal to 0.00035.

    You will be shown an error message if there is no effect object with the
    name you gave, or that effect object doesn't have the parameter you
    specified, or you gave a value that wasn't of the correct type.

    To find out which parameters the different effect types have see the
    effect reference.

    `run`

    Process sounds through the network of effect objects you have set up.
```

The processing stops when there is no more input from sources (like reading sound data from disk) and all of the outputs (like writing sound data to disk) have become quiet (so that the "tails" of echoes are not cut off too quickly).
You will be shown an error message if the effect objects are linked together incorrectly, for example if there is a feedback loop without a feedback effect object in it, or if there are some inputs or outputs that are not connected to anything. Other things that can go wrong include not being able to open sound files to read from (for example the file doesn't exist) or write to (for example the disk is write protected).

Comments

You can add comments to the command you are typing in, so that you can remember what what you have done is for more easily. There are two types of comment. If you type "//" (without "") everything until the end of the line is ignored by the application. For longer comments, anything between "/*" and "*/" (without "") is ignored. You can "nest" layers of these, so "/* my /* nested */ comment */" is allowed, but there must be an equal number of "/*" and "*/", otherwise you will be told about the error.

Effect reference

This section describes all of the effects available.

add

The output is the sum of all the inputs.

Inputs

in1
in2
... up to the inputs parameter

Outputs

main

Parameters

inputs the number of inputs to add together

bandpass

A band pass filter effect, that allows frequencies within a certain range to pass and blocks those outside the band.

Inputs

main

Outputs

main

```
990
    ##### Parameters

    lowfreq low cutoff frequency in Hertz (defaults to 250)

995 highfreq  high cutoff frequency in Hertz (defaults to 2000)

    #### bandreject

    A band reject filter effect, that blocks frequencies within a certain
1000 range and allows those outside the band to pass.

    ##### Inputs

    main
1005
    ##### Outputs

    main

1010 ##### Parameters

    lowfreq low cutoff frequency in Hertz (defaults to 250)

    highfreq  high cutoff frequency in Hertz (defaults to 2000)
1015
    #### compand

    This is a dynamic range compression and expansion effect.  If the
    control input is above the threshold level, then the output is a scaled
1020 according to the ratio.  If the ratio is less than one, differences
    between amplitudes are reduced (the signal is *comp*ressed).  If the
    ratio is greater then 1, then differences in level are exagerated (the
    signal is exp*and*ed).

1025 The time parameter controls the level detection.  If the time is too
    short then low frequency signals can cause "pumping".  A long time can
    result in rapid changes in level not being affected.

    ##### Inputs
1030
    main  the signal to be manipulated

    sidechain the control signal, if this is not linked then the main input
    is used to modify itself
1035
    ##### Outputs

    main

1040 ##### Parameters

    time  the time over which to calculate the average level, in seconds
    (defaults to 0.05)

1045 threshold the cutoff level (defaults to 0.5)
```

```
ratio the compression ratio (defaults to 1)

#### delay

The output is the input delayed by the delay time. The output is zero
until the delay time has passed. Changing the delay time by large
amounts during processing can result in "glitches", the output jumping s
uddenly from one value to another. Slow changes can result in the pitch
being altered, as the output passes more quickly or more slowly than the
input.

##### Inputs

main

##### Outputs

main

##### Parameters

delay the delay time in seconds (defaults to 0.1)

#### echo

An echo effect. Each echo is quieter by the decay factor (which should
be less than 1), and they are separated by the delay time.

##### Inputs

main

##### Outputs

main

##### Parameters

decay how much quieter succesive echoes are (defaults to 0.5)

delay the time between echoes in seconds (defaults to 0.25)

#### envfollow

The output is the volume envelope (average signal level) of the input.
This can be used to control effects according to the signal level.

The time parameter controls the level detection. If the time is too
short then the output will contain low frequencies from the input. A
long time can result in sudden changes not being followed.

The output envelope is delayed by half of the time parameter, relative
to the input sound.

##### Inputs

main
```

```
##### Outputs

main

##### Parameters

time  the time over which to calculate the average level, in seconds
(defaults to 0.05)

#### fbdelay

This effect should be used instead of one delay in a feedback loop.
See delay and feedback.

##### Inputs

main

##### Outputs

main

##### Parameters

delay the delay time in seconds (defaults to 0.1)

#### feedback

A feedback effect must be present in any feedback loop.  The output is
the same as the input, delayed by one sampling period (the shortest
possible time).  For accurate delay times, the effect fbdelay should be
used in place of one delay in the feedback loop.

##### Inputs

main

##### Outputs

main

##### Parameters

n/a

#### gate

If the average sidechain input level is below the threshold parameter
then the output is scaled to zero, otherwise the output is the main
input.  Gating is useful for removing background noise during gaps in
the main signal.

The time parameter controls the level detection.  If the time is too
short then low frequency signals can cause the gate to open and close in
time, resulting in "pumping".  A long time can result in short quiet
sections not being masked.
```

```
The output is delayed by half of the time parameter, relative to the
input.

##### Inputs

main  the signal to be manipulated

sidechain the control signal, if this is not linked then the main input
is used as the control

##### Outputs

main

##### Parameters

time  the time over which to calculate the average level, in seconds
(defaults to 0.05)

threshold the cutoff level (defaults to 0.5)

#### halfrectify

This is a half-wave rectifier effect. When the input is positive, the
output is the same as the input, otherwise the output is zero, so the
parts of the waveform below the axis are "cut off". This leads to an
increase in frequencies one octave above the fundamental, although the
results are not as pronounced as full rectification (see rectify).

##### Inputs

main

##### Outputs

main

##### Parameters

n/a

#### highpass

A high pass filter effect, that allows high frequencies to pass but
blocks low frequencies. The freq parameter indicates the cutoff
frequency, below which lower frequencies are reduced.

##### Inputs

main

##### Outputs

main

##### Parameters
```

```
freq  cutoff frequency in Hertz (defaults to 2000)

#### invert

The output is the input inverted, so peaks in the waveform become
troughs and vice versa.

##### Inputs

main

##### Outputs

main

##### Parameters

n/a

#### limit

If the average level is above the threshold, the amplitude is scaled
down to the threshold level (similar to a compressor (see compand), but
more severe).

The time parameter controls the level detection.  If the time is too
short then low frequency signals can cause "pumping".  A long time can
result in sudden loud sections not being reduced in level.

The purpose of a limiter in music recording is to prevent the signal
from exceeding a certain level, so that the recording device doesn't
overload and distort.  See the various write effects for details.

The output is delayed by half of the time parameter, relative to the
input.

##### Inputs

main

##### Outputs

main

##### Parameters

time  the time over which to calculate the average level, in seconds
(defaults to 0.05)

threshold the maximum level (defaults to 1)

#### lowpass

A low pass filter effect, that allows low frequencies to pass but blocks
higher frequencies.  The freq parameter indicates the cutoff frequency,
above which higher frequencies are reduced.
```

```
1275
      ##### Inputs

      main

1280  ##### Outputs

      main

      ##### Parameters
1285
      freq  cutoff frequency in Hertz (defaults to 250)

      #### mul

1290  The output is all of the inputs multiplied together.  This can be used
      to change the volume of sounds (if one input is slowly varying) or add
      new frequencies (for sounds of similar pitch).

      ##### Inputs
1295
      in1
      in2
      ... up to the inputs parameter

1300  ##### Outputs

      main

      ##### Parameters
1305
      inputs  the number of inputs to multiply together

      #### pitchshift

1310  A pitch shifter changes the pitch of a sound without changing the speed.
      The ratio parameter sets how much to change the pitch by (for example 2
      will raise the pitch by one octave).  The freq parameter controls the
      frequency at which the sound is repeated,  the effect works by recording
      short sections and repeating them more quickly or more slowly.  For best
1315  results, when pitching up the freq parameter should be close to the
      fundamental frequency of the sound, but lower when pitching down.

      ##### Inputs

1320  main

      ##### Outputs

      main
1325
      ##### Parameters

      ratio the pitch change factor (defaults to 1)

1330  freq  the shifting frequency in Hertz (defaults to 256)
```

readslab, read8svx, readwav

Reads the output from a sample file of format SLab, IFF 8SVX, or RIFF WAVE (respectively). The output has a maximum amplitude of 1 for each type other than SLab's own, which may contain any value.

By default SLab files are normalised when they are read in. This means that the sound is scaled so that the maximum amplitude is 1, which is what is wanted for normal sounds, but probably not for control signals, for which normalisation may be turned off.

Inputs

n/a

Outputs

main

Parameters

file the file to read from

normalise (SLab format only) set to "yes" or "no" (defaults to "yes")

rectify

This is a full-wave rectifier effect. The output is the absolute value of the input, so parts of the waveform below the axis are folded over. This leads to an increase in frequencies one octave above the fundamental.

Inputs

main

Outputs

main

Parameters

n/a

reverse

This effect reverses sound in time. This effect has to store the sound coming in before it can play it backwards, so the time parameter indicates how much to store. The first output is after the time parameter, after which the first part of the input is output in reverse, followed by later sections in reverse.

To reverse an entire sound, simply set the time to longer than the sound. Interesting effects can be obtained using very short times (for example, 0.002)

Inputs

main

Outputs

main

Parameters

time the reverse time in seconds (defaults to 0.25)

split

The input is sent unaltered to all the outputs. This is often used to combine effects in parallel.

Inputs

main

Outputs

out1
out2
... up to the outputs parameter

Parameters

outputs the number of outputs to send to

vox

This effect absorbs all of the input, not passing it on, until it rises above the threshold. After that, the output is equal to the input. This is useful for preventing output files starting with a period of silence.

Warning

As this effect doesn't send input for a time, it should be used with caution. Unpredictable results will occur if the output of one vox effect is linked (directly or indirectly) to an effect that has an input not linked to the same vox effect. It is recommended that this effect is used directly before the final output.

Inputs

main

Outputs

main

Parameters

threshold the cutoff level (defaults to 0.001)

widen

This effect changes the width of a stereo image. If the size of the width parameter is greater than 1, left and right seem further apart, otherwise they seem closer. Negative width parameters swap left and right.

Inputs

left

right

Outputs

left

right

Parameters

width the width of the stereo image (defaults to 1)

writeslab, write8svx, writewav

Write the input to a sample file of format SLab, IFF 8SVX, or RIFF WAVE (respectively). SLab's own format is the only one that doesn't clip the signal. The others distort for input signals with an amplitude greater than 1, so a limiter may be necessary (see limit).

Warning

Any already existing file will be overwritten, so make sure there is no file with the same name before starting.

Inputs

main

Outputs

n/a

Parameters

file the file to create

zfilter

This is a z-plane filter effect. The z-transform is a mathematical technique that allows filters to be made according to design, however this can be complicated. Some preset filters have already been set up (see lowpass, bandpass, highpass, bandreject) so that they can be used more easily. Essentially, frequencies are represented as going around a semicircle, and poles and zeros are placed within the semicircle. Poles make frequencies near them louder and those further away quieter, and zeros make frequencies near them quieter and those further away louder.

```
The details of designing filters will not be gone into here, for more
information consult a good book on the topic (for example, "An
introduction to the analysis and processing of signals" by P. A. Lynn,
1973-89). Make sure that no poles have a radius greater than 1, and
that for each pole or zero with a frequency not equal to zero or half of
the sampling rate there is another with the same radius but negative
frequency.

##### Warning

This effect is very powerful, but you do need to know what you are doing
to be able to use it properly.

##### Inputs

main

##### Outputs

main

##### Parameters

poles the number of poles in the filter

zeros the number of zeros in the filter

pole1r
pole2r
...
zero1r
zero2r
... the radius of the poles and zeros

pole1f
pole2f
...
zero1f
zero2f
... the frequencies of the poles and zeros

## Tutorial

This section is a step by step guide in using the application. In this
section, things you need to type in are printed like this, and the
output of the application is printed like this.

### Adding an echo to a sound

As a first tutorial, we will add some echo to a sound that you have on
disk.

First we need to get the sound from the disk. Here we will use the file
"MySounds:Funky/OrchStab.8svx", you will need to use one of your own
files. As this is an IFF 8SVX file (indicated by the extensions .8svx or
.iff), we will need an 8SVX reader:
```

```
>> new read_8svx reader
>> set reader.file "MySounds:Funky/OrchStab.8svx"
```

Now we need to decide where to put the echoed sound. We will use "MySounds:Funky/OrchStab_Echo.8svx", again you should choose your own name for the new sound file. We will write the file as an IFF 8SVX, although you can choose a different format if you want to:

```
>> new write_8svx writer
>> set writer.file "MySounds:Funky/OrchStab_Echo.8svx"
```

We now have a reader and a writer, time to put the echo in between. We will have a fairly long echo time of one and a half seconds, but which dies away relatively quickly (by having the decay close to zero):

```
>> new echo echo
>> set echo.delay 1.5
>> set echo.decay 0.25
```

Note how you can have an effect object with the same name as an effect type. The computer doesn't get confused, although with more complicated processing than this simple echo you might confuse yourself!

With all of the effect objects set up, now we have to link them together:

```
>> link reader.main echo.main
>> link echo.main writer.main
```

Now all of the setting up is done, we can process the sound:

```
>> run
```

All being well, a new file will be created containing the echoed sound. You will be informed of any problems, for example if there is not enough space on the disk for the new sound file.

Making your own echo effect

Although there is a built in echo effect, here we will show how to make your own echo effect out of simpler building blocks.

An echo effect is quite simple. Even so, we will need six effect objects for one echo effect, as you can see from the diagram. First we create the objects we need, the names start with e_ so that we know that they are all part of one echo effect:

```
>> new add e_add
>> set add.inputs 2
>> new split e_split
>> set split.outputs 2
>> new delay e_delay
>> new mul e_scale
>> set mul.inputs 2
>> new feedback e_fb
>> new constant e_decay
```

Then we link them together:

```
>> link e_add.main split.main
>> link e_split.out2 e_delay.main
>> link e_delay.main e_scale.in1
>> link e_decay.main e_scale.in2
>> link e_scale.main e_feedback.main
>> link e_feedback.main e_add.main
```

Now that the effect objects making up the echo are linked together, we can set the echo parameters (the decay value should be between 1 and -1, otherwise the echo would make the sound get louder and louder):

```
>> set e_decay.out .5
>> set e_delay.delay .33333
```

This gives an echo half the volume of the previous one, about 3 times per second.

Now that our echo is set up, we can link it to a reader and a writer to process a sound. This is described in detail in an earlier tutorial. You need to link to e_add.in1 and from e_split.out1.

Dynamically controlled effects

Here the real power of the application begins to show itself. We are going to control some effects with other effects, to create a very unusual sound.

The diagram explains what we are going to do, only a few notes will be placed as comments in the following. You can type the comments in, they do not affect the results.

Warning: the output file will be nearly 700 kB in size, so make sure there is enough space before you run.

```
new read8svx reader                 ; Set up sample source
set reader file "Tutorial3/Input.8svx"
new add readmix                     ; Repeat sample every 0.5s
```

```
    set readmix inputs 2
1675 new split readsplit
    set readsplit outputs 2
    new feedback readfb
    new fbdelay readdelay
    set readdelay delay 0.5
1680 link reader main readmix in1
    link readmix main readsplit main
    link readsplit out2 readdelay main
    link readdelay main readfb main
    link readfb main readmix in2
1685 new rampup volume                  ; Set up volume oscillator
    set volume freq 0.5
    new constant volscale               ; Scale to between 0.5 and 1
    set volscale value 0.25             ; (1 - 0.5) / 2
    new constant volshift
1690 set volshift value 0.75            ; scale * 3
    new mul volmul
    set volmul inputs 2
    new add voladd
    set voladd inputs 2
1695 link volume main volmul in1
    link volscale main volmul in2
    link volmul main voladd in1
    link volshift main voladd in2
    new mul changevol                   ; Modulate volume
1700 set changevol inputs 2
    link readsplit out1 changevol in1
    link voladd main changevol in2
    new zfilter filter                  ; Set up filter
    set filter poles 4
1705 set filter zeros 4
    set filter pole1r 0.95              ; Poles just outside zeros
    set filter pole2r 0.95              ; give isolated peaks
    set filter pole3r 0.95
    set filter pole4r 0.95
1710 set filter zero1r 0.9
    set filter zero2r 0.9
    set filter zero3r 0.9
    set filter zero4r 0.9
    link changevol main filter main
1715 new sine freq                      ; Set up frequency oscillator
    set freq freq 0.25
    new constant frqscale               ; Scale to between 700 and 2100
    set frqscale value 700              ; (2100 - 700) / 2
    new constant frqshift
1720 set frqshift value 2100            ; scale * 3
    new mul frqmul
    set frqmul inputs 2
    new add frqadd
    set frqadd inputs 2
1725 link frqume main frqmul in1
    link frqscale main frqmul in2
    link frqmul main frqadd in1
    link frqshift main frqadd in2
    new toparam ctrl1                   ; Set up filter control
1730 new toparam ctrl2
```

```
new toparam ctrl3
new toparam ctrl4
new toparam ctrl5
new toparam ctrl6
new toparam ctrl7
new toparam ctrl8
set ctrl1 to filter
set ctrl2 to filter
set ctrl3 to filter
set ctrl4 to filter
set ctrl5 to filter
set ctrl6 to filter
set ctrl7 to filter
set ctrl8 to filter
set ctrl1 param pole1f
set ctrl2 param pole2f
set ctrl3 param pole3f
set ctrl4 param pole4f
set ctrl5 param zero1f
set ctrl6 param zero2f
set ctrl7 param zero3f
set ctrl8 param zero4f
new split ctrlsplit1                ; Set up filter control routing
set ctrlsplit1 outputs 4
new split ctrlsplit2
set ctrlsplit2 outputs 3
new split ctrlsplit3
set ctrlsplit3 outputs 2
new split ctrlsplit4
set ctrlsplit4 outputs 2
new mul ctrlmul
set ctrlmul inputs 2
new constant ctrlband
set ctrlband 1.5
link ctrlband ctrlmul in1
new invert ctrlinv1
new invert ctrlinv2
link frqadd main ctrlsplit1 main    ; Link filter routing
link ctrlsplit1 out1 ctrl1 main
link ctrlsplit1 out2 ctrl5 main
link ctrlsplit1 out3 ctrlinv1 main
link ctrlinv1 main ctrlsplit3 main
link ctrlsplit3 out1 ctrl2 main
link ctrlsplit3 out2 ctrl6 main
link ctrlsplit1 out4 ctrlmul in2
link ctrlmul main ctrlsplit2 main
link ctrlsplit2 out1 ctrl3 main
link ctrlsplit2 out2 ctrl7 main
link ctrlsplit2 out3 ctrlinv2 main
link ctrlinv2 main ctrlsplit4 main
link ctrlsplit4 out1 ctrl4 main
link ctrlsplit4 out2 ctrl8 main
new writewav writer                 ; Set up sample output
set write file "Tutorial3/Output.wav"
link filter main writer main
set . runtime 8                     ; Run for 8 seconds
run
```

```
```

Evaluation

Ease of use

The system is laborious to use. The command line interface is long winded, however all of the commands are necessary. A graphical user interface would make many operations possible with one or two mouse clicks rather than a line of text. A second advantage is that all linkages would be visible, so you would not have to remember what names had been given to each effect object or what links had already been made.

The scripting facility hinted at in the analysis section has not been implemented. However, there are two workarounds.

Firstly, the input and output handles for the SLab command can be redirected using the system Shell:

```
SLab <mycommands.txt >NIL:
```

The redirected input file should contain commands as they would be entered into the console window. The quit command is not strictly necessary at the end of the input file, as SLab quits when an EOF is read from the input.

Secondly, commands can be pasted into the console window using the standard system key (Right Amiga - V), copied from any source (for example a text editor). Multiple commands can be pasted simultaneously, however as all of the lines are entered at once the system cannot display the prompts for each command until the input stops, at which point all of the prompts are displayed on one line. This looks unaesthetic but has no effect on the correct working of the system. This method was used during testing.

The scripting facility need not be implemented within SLab, rather there should be an ARexx port. ARexx is a simple interpreted language, but applications can create their own (named) ARexx port. Using the ARexx ADDRESS command, any command line not recognised by ARexx as an ARexx command is passed to this port.

A mechanism is in place at the effect level for a get command to get the current values of parameters, however this has not been implemented in the kernel or the command line interface.

Ease of implementation

Simple effects, such as rectify, can be implemented very easily. However, the command line interface parts of the effects (those that convert strings to id codes) show a large amount of repetition of simple code. To make implementing effect classes easier, the inputs, outputs and parameters could be stored in a table containing the name, id code and various properties (such as the offset of the link structure in the effect object for inputs, the type of parameters, and whether linking or

setting this parameter requires recalculation). This table could be used by generic methods of the effect base class, any classes that require more complicated arrangements (for example zfilter or the multiple input or output classes) could use the current arrangement.

A useful side effect of this table driven method is that it would be simple to list all of the inputs, outputs and parameters of a given effect class or object. With the addition of textual descriptions this could also become an online help system.

Quality of results

The quality of the output is very high, especially the filter effects. When compared to simple moving average filters (as found in OctaMED), the sound is much clearer. OctaMED's filters tend to make the sound seem muffled. The lack of clipping ensured by floating point implementation makes it easier to combine effects; in OctaMED's sample editor some effects (like echo) can lead to volume increase and clipping so the volume must be reduced first.

The echo effect (see testing, first echo) exibits a slight loss of high frequencies in the echos, this is due to the linear interpolation used in the delay effect when the delay time is not an integer number of samples. Natural echos from soft or irregular surfaces tend to exhibit loss of high frequencies, so this property may be useful.

The (mathematically) correct interpolation requires summing the function $y=sin(at)/(at)$ for every sample (past and future), this function has a peak at the sample in question and is zero at all other sample points. However an implementation of this interpolation would be slow, and is not really necessary. Alternatively, a switch parameter added to the delay effect could ensure that the delay time is adjusted to be an integer number of samples.

Speed of processing

The sound processing is very slow, but this is due to obsolete hardware being used (the CPU is a 7 MHz Motorola 68000, with 16 bit integer multiplication taking 70 clock cycles). Modern computers are easily a thousand times faster at floating point maths, so the system would be much more useable. On the current hardware, simple processing of one second of sound (at 22050 Hz sampling rate) takes about one minute, rising to over five minutes if parameters are continually changed that require recalculation (for example filter frequencies). Speed could be increased by assembly language optimisation of critical sections, but there are unlikely to be large gains as there are few loops in the code.

Faults

The command line interface is currently case sensitive. The functions that need to be changed are in the source code file string.e, and should be made to use the system standard utility.library functions. There is little checking on names, with the result that an effect object can end up with the name "" (or more dangerously "."), confusing the user and the system.

The run command doesn't check for the end of sounds. Currently it runs

for a fixed number of samples (22050), longer waveforms can be processed by using the run command several times in succession. This can be fixed by having effect.issink() and effect.isdone() methods, and run finishes when all sinks are done. Alternatively (for use when the sound will never cease) the set command can be expanded to include global parameters, with "." as the effect name for consistency with the list command. Global parameters could include rate (global sample rate, perhaps defaulting to the maximum sample rate in use), runtime (time to run for), or runsamples (number of samples to run for).

The effect linkage checking enters an infinite loop if effects are linked in a loop without a feedback effect, with the recursion leading to stack overflow which could crash the operating system. This could be fixed by having every effect check for loops, this would require the effect.check() method to be split into two methods, one defined by the effect base class to prevent loops, and one defined in each derived class to do the checking. The latter function would be called by the loop prevention method. This fix would require that feedback effects are last in the loop, so that these can stop the loop (otherwise effects in the loop after the feedback effect would not be checked).

A simpler fix would be to check free stack space, if this is very low it is due to either extremely long chains of effects or the recursive loop described above. However, this method would give only a vague error message, that there was a loop somewhere in the effects linkage.

File handling leaves much to be desired. Error reporting is poor, no errors are reported if files cannot be opened, and read / write errors currently exit the system ungracefully (as does running out of memory). The files are opened too early (when the name is set) and closed too late (when the name is set to something else or the effect is deleted (including exiting the system)).

The write effects do not check whether the file exists, and as a side effect the reset command causes the file that has been written to be erased. A workaround is to use a command like "set mywriter file NIL:" before using the reset command.

Appendix

Mathematical Background

Transform Theory

Fourier Transform

The Fourier transform is derived from the Fourier series, a method of representing periodic functions by infinite series of sine and cosine functions. The series is extended to aperiodic functions by having a continuous (rather than discrete) frequency spectrum, expressed more concisely using complex exponentials:

Laplace Transform

The Laplace transform is an extension of the Fourier transform, which is valid for more functions. The Laplace transform uses a complex frequency variable :

```
1960 ##### z Transform

     The Fourier and Laplace functions are for continuous functions, but
     sampled data signals are made up of many discrete points. A signal is
     represented by a sum of impulse functions, separated by the sampling
1965 time interval . The Laplace transform is easily found, and by setting ,
     the z transform can be derived:

     #### Practical Uses of the Laplace and z Transforms

1970 ##### Transfer Function and Impulse Response

     A linear system can be represented by a transfer function, because the
     transform of the output is the transform of the input multiplied by the
     transfer function. The response of a system to a unit impulse is its
1975 impulse response, the transform of which is the transfer function.

     ##### Convolution

     Multiplication of transforms is the same as convolution in the time
1980 domain. Convolution for sampled data means that each sample is replaced
     by a scaled copy of the impulse response and the output is the sum of
     all of them. This is expressed as an integral for continuous functions:

     ##### Poles and Zeros
1985
     Many systems can be represented as a set of poles and zeros in the s
     or z plane, specifying the transfer function, from which the frequency
     and phase response characteristics of the system can be determined.
     Conversely, a filter can be designed by placing poles and zeros to
1990 create a desired characteristic.

     ##### z Domain Filtering

     The z domain transfer function can be used to generate a simple
1995 recursion formula to process sampled input. The formula gives the
     current output in terms of the current and previous inputs and previous
     outputs. Each entire signal can be shifted to minimize delay (if so
     desired) or create a realisable filter (where effect is later in time
     than cause), this is equivalent to having extra poles or zeros at the
2000 origin of the z-plane, unaffecting the frequency response.

     ### File Formats

     #### IFF 8SVX
2005
     The IFF (Interchange File Format) standard defines a generic file
     structure, built around chunks. Different types of file (sound,
     graphics etc) define new chunks.

2010 ##### Data types

     - ULONG unsigned long, 4 bytes, msb first
     - UWORD unsigned word, 2 bytes, msb first
     - UBYTE unsigned byte
2015 - SBYTE signed byte (two's complement, -128..+127)
```

Chunk structure

```
  ULONG chunk id (usually a character string)
  ULONG data length
  ... data
```

FORM

Every IFF file is a FORM chunk, containing other chunks:

```
  ULONG "FORM"
  ULONG length
  ULONG type
  ... chunk list
```

For 8SVX sound files, the FORM type field is "8SVX", and then a VHDR and a BODY chunk are required (in this order). All IFF files may contain other chunks, but these can be skipped using the length field.

VHDR (Voice Header)

```
  ULONG "VHDR"
  ULONG length
  ULONG samples in high octave 1-shot part
  ULONG sample start offset of high octave repeat part
  ULONG samples per cycle in high octave repeat (0 = no repeat)
  UWORD samples per second
  UBYTE number of octaves
  UBYTE compression: 0 = none, 1 = Fibonacci-delta encoding
  ULONG volume (65536 maps to 1.0)
```

BODY

```
  ULONG "BODY"
  ULONG length
  SBYTEs sample data
```

RIFF WAVE

The RIFF (Resource Interchange File Format) standard is Microsoft's own duplication of the IFF file structure. The main difference is that the data words are stored lsb first.

RIFF

Every RIFF file is a RIFF chunk, containing other chunks:

```
```

```
  ULONG "RIFF"
  ULONG length
  ULONG type
  ...   chunk list
```

For WAVE sound files, the RIFF type field is "WAVE", and then a fmt and a data chunk are required (in this order). All RIFF files may contain other chunks, but these can be skipped using the length field.

fmt

```
  ULONG "fmt "
  ULONG length
  UWORD encoding (1 = PCM)
  UWORD number of channels
  ULONG sampling rate
  ULONG bandwidth (= rate * channels * [bits/8])
  UWORD block align (= channels * [bits/8])
  // encoding specific data, here encoding = 1
  UWORD bits per sample
```

data

```
  ULONG "data"
  ULONG length
  // data format for encoding = 1
  // bits = 1 to 8, UBYTE (least significant bits 0)
  // bits > 8, signed integer of least number of bytes required (for
  // example 3 bytes for 20 bit) (least significant bits 0)
  ... sample data, channels interleaved
```

RIFF WAVE files are quite complicated, and can include cue points (with text labels and comments), silent sections, data compression, play lists, and even embedded files (images to be displayed at cue points, for example).

SLab file format

```
ULONG "SLab"
ULONG length

  ULONG "Info"
  ULONG length = 12
  FLOAT rate (def = 44100.0)
  FLOAT bias (def = 0.0) \ for normalisation
  FLOAT ampl (def = 1.0) / when reading in

  ULONG "Data"
  ULONG length = number of samples
  FLOATs  sample data
```

```
2130
    ### SLab class heirarchy

    ```
    () = base classes not for direct use
2135 - = not yet implemented

    (effect)   base class
        (container)   container base class
            echo   input + output "main", params "decay" + "delay"
2140   - pitchshift
            notch   input + output "main", params "depth" + "frequency"
            - reverse
            - widen   parameter "width" -1 = swap, 0 = mono, 1 = same
        (in0out1)   one output "main"
2145   constant   output = param "value"
            (osc)   oscillator, params "rate", "frequency", "amplitude",↙
                ↳ "phase"
                pulse   pulse, param "width"
                ramp   ramp (up; down <=> negative amplitude)
                sine   sine wave
2150   triangle   triangular wave
            (read)   read from param "file", with sample rate "rate"
                                (def = rate stored in file)
                read8svx   file type = IFF 8SVX
                readslab   file type = SLab, param "normalize" = "on", "off"
2155   - readwave   file type = RIFF WAVE (16 bit)
            whitenoise   white noise
        (in1out0)   one input "main"
            print   write to screen
            toparam   converts data to parameters, param "to" (objpart)
2160   (write)   write to param "file"
                write8svx   file type = IFF 8SVX
                writeslab   file type = SLab
                - writewave   file type = RIFF WAVE (16 bit)
        (in1out1)   one input "main", one output "main"
2165   amp   amplify by param "gain"
            copy   output = input
            delay   output(t) = input(t - param "delay")
                fbdelay   delay fixed for feedback loops
            - envfollow   envelope follower, parameter "time"
2170   (- envchain)   optional input "sidechain", parameter "threshold"
                    - compand   compressor / expander
                    - gate   gate
                    - limit   limiter (ducker when used with sidechain ?)
            feedback   allow feedback loops
2175   (filter)   simple filter, param "frequency"
                (bandfilter)   param "bandwidth"
                    bandpass   band pass filter
                    bandreject band reject filter
                highpass   high pass filter
2180   lowpass   low pass filter
            halfrectify   output = if input > 0 then input else 0
            invert   output = - input
            rectify   output = abs(input)
            - vox   output = input, after input > param "threshold"
2185   zfilter   params "poles", "zeros", "poleXr", "poleXf", "zeroXr↙
```
```

```
            ↳ ", "zeroXf"
    (in1outm)               one input "main", param "outputs" "out1" etc
        split               outputX = input
        - multidelay        outputX = input(t - delayX)
    (inmout1)               param "inputs" "in1" etc, one output "main"
        add                 output = in1 + in2 + ...
        mul                 output = in1 * in2 * ...
```

Testing

Most of the effects were tested as follows. The test input structure
was the same, with the effect specific commands inserted at "...".
The audio cassette contains for each effect the original and the
modified sound twice each.

```
new read8svx r
set r file Test/Beat.8svx
set r rate 22050
new write8svx w
set w file Test/Beat.<effect>.8svx
new <effect> f
...
link r main f main
link f main w main
run
quit


amp
set f gain 0.5

bandpass
set f frequency 1000
set f bandwidth 500

echo 1
set f delay 0.39
set f decay 0.5

echo 2
set f delay 0.01
set f decay 0.9

halfrectify

highpass
set f frequency 1000

lowpass
set f frequency 300

rectify
```

Oscillators were tested differently. The audio cassette contains short

sections of output from each of the oscillators with default parameters.

```
new <effect> o
set o rate 22050
...
new write8svx w
set w file Test/Osc.<effect>.8svx
link o main w main
run
quit


sine

ramp

pulse

pulse

set o width 0.2

triangle

whitenoise
```

The notch filter was tested as follows. A short section of the output is shown, you can see that the fundamental frequency has been removed from the otherwise square wave. Other features of the output are phase changes (leading to asymmetry of the waveform) and time taken to reach the steady state response (the start is slightly different to the remainder of the wave). The diagram is a screenshot taken from OctaMED's sample editor, showing 256 samples.

```
new pulse r
set r frequency 1000
set r amplitude 0.5
new write8svx w
set w file Test/Notch.8svx
new notch f
set f depth 0.95
set f frequency 1000
link r main f main
link f main w main
run
quit
```

References

LEDA

The matrix and vector classes are taken from LEDA with slight modifications.

```
"In the fall of 1988, we started a project (called LEDA for Library of
Efficient Datatypes and Algorithms) to build a small, but growing
library of data types and algorithms in a form which allows them to be
used by non-experts. We hope that the system will narrow the gap
between algorithms research, teaching, and implementation.

LEDA is available by anonymous ftp from:

ftp.cs.uni-sb.de (134.96.7.254) /pub/LEDA

ftp.maths.warwick.ac.uk (137.205.232.4) /pub/sources/c++

The distribution contains all sources, installation instructions, a
technical report, and the LEDA user manual. LEDA is not in the public
domain, but can be used freely for research and teaching. A commercial
license is available from the author."

#### Amiga Developer CD v1.1

The developer CD contains documentation on operating system functions
and the IFF standards, among other things.

#### Internet Sites

There are several useful sites on the Internet concerned with sound
processing. A description of various effects processors can be found
at:

http://www.eden.com/~keen/effxfaq/fxtaxon.htm
?
http://www.hut.fi/Misc/Electronics/dsp.html

#### An introduction to the analysis and processing of signals

by P. A. Lynn, (c) 1973-89. This book describes how to use the Laplace
and z transforms to process signals, including using z-plane filters and
recursion formulae to filter sampled sound.
```

2 Install

```
;------------------------------------------------------------------------;
; Install                                                                ;
; Installer script for SLab                                              ;
;                                                                        ;
; Icon properties:                                                       ;
;   Icon:         Install.info                                           ;
;   Icontype:     PROJECT                                                ;
;   Defaulttool: Installer                                               ;
;   Tooltypes:                                                           ;
;       APPNAME=SLab                                                     ;
;       LOGFILE=T:Install_SLab.logfile                                   ;
;       LOG=TRUE                                                         ;
;       DEFUSER=AVERAGE                                                  ;
;       PRETEND=FALSE                                                    ;
;------------------------------------------------------------------------;
```

```
(set @app-name "SLab")
(welcome "SLab Installation")

(set #Msg_WrongOS           "You need at least AmigaOS v2.04 to run SLab.")
(set #Msg_Installing        "Installing SLab.")
(set #Msg_SelectPath        "Select path for SLab:")
(set #Msg_DoInstall         "Installation complete.")
(set #Msg_Failed            "Installation failed.")
(set #Msg_Tutorial          "Do you want to install the tutorial files?")
(set #Msg_TutorialHelp      "The tutorial files offer guidance on using SLab, ↙
    ↳ but are not needed to run SLab.")

(set #OS_VER (/ (getversion) 65536) )
(if (< #OS_VER 37) (abort #Msg_WrongOS) )

;---------------------------------------------------------------------------;

(complete 0)
(set @default-dest (askdir (prompt #Msg_SelectPath) (help @askdir-help)
                           (newpath) (default "SYS:SLab")
                   )
)

(copyfiles (prompt #Msg_Installing) (help @copyfiles-help)
           (source "SLab") (dest @default-dest) (files) (infos)
)

;---------------------------------------------------------------------------;

(complete 50)
(if (askbool (prompt #Msg_Tutorial) (help #Msg_TutorialHelp) (default 1) )
        (copyfiles (prompt #Msg_Installing) (help @copyfiles-help)
                   (source "Tutorial") (dest @default-dest) (infos) (all)
        )
)

;---------------------------------------------------------------------------;

(complete 100)
(exit #Msg_DoInstall)

;---------------------------------------------------------------------------;
; END: Install                                                              ;
;---------------------------------------------------------------------------;
```

3 README.md

```
This is my AS-Level Computing project from 1999.

Source code is in Amiga E.

Documentation is missing images.
```

4 SLab

(application/octet-stream; charset=binary)

5 Source/add.e

```
/*=============================================================================+
| add.e                                                                        |
| Effect class "add"                                                           |
| Sums all inputs                                                              |
+-----------------------------------------------------------------------------*/

OPT MODULE

MODULE '*defs', '*inmout1', '*debug'

EXPORT OBJECT add OF inmout1
ENDOBJECT

PROC class() OF add IS 'add'

PROC process() OF add
        DEF i, o = 0.0
        SUPER self.process()
        FOR i := 1 TO self._inputs() DO o := ! o + self._in(i)
        self.output(ID_MAIN, o)
ENDPROC

/*-----------------------------------------------------------------------------+
| END: add.e                                                                   |
+=============================================================================*/
```

6 Source/amp.e

```
/*=============================================================================+
| amp.e                                                                        |
| Effect class "amp", amplifies input by parameter "gain"                      |
+-----------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*in1out1', '*string', '*value', '*debug'

/*-----------------------------------------------------------------------------*/

EXPORT OBJECT amp OF in1out1
PRIVATE
        gain : LONG
ENDOBJECT

PROC class() OF amp IS 'amp'

/*-----------------------------------------------------------------------------*/

PROC new(list, name) OF amp
        SUPER self.new(list, name)
        self.gain := 1.0
ENDPROC

/*-----------------------------------------------------------------------------*/
```

```
PROC param2id(str) OF amp
ENDPROC IF strcmp(IDS_GAIN, str) THEN ID_GAIN ELSE SUPER self.param2id(str)

PROC id2param(id) OF amp
ENDPROC IF id = ID_GAIN THEN IDS_GAIN ELSE SUPER self.id2param(id)

PROC paramtype(id) OF amp
ENDPROC IF id = ID_GAIN THEN TYPE_NUMBER ELSE SUPER self.paramtype(id)

/*-----------------------------------------------------------------------*/

PROC set(id, data) OF amp
        SELECT id
        CASE ID_GAIN; self.gain := data
        DEFAULT;      SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-----------------------------------------------------------------------*/

PROC get(id) OF amp
        SELECT id
        CASE ID_GAIN; RETURN self.gain
        ENDSELECT
ENDPROC SUPER self.get(id)

/*-----------------------------------------------------------------------*/

PROC process() OF amp
        SUPER self.process()
        self.output(ID_MAIN, ! self._main() * self.gain)
ENDPROC

/*-----------------------------------------------------------------------+
| END: amp.e                                                             |
+=======================================================================*/
```

7 Source/bandfilter.e

```
/*=======================================================================+
| bandfilter.e                                                           |
| Effect base class "bandfilter"                                         |
| Adds a parameter "bandwidth", used by bandpass and bandreject          |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*filter', '*link', '*string', '*value'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT bandfilter OF filter
PUBLIC
        band : LONG     -> bandwidth (Hz)
ENDOBJECT
```

```
PROC class() OF bandfilter IS 'bandfilter'

/*-------------------------------------------------------------------------*/

PROC new(list, name) OF bandfilter
        SUPER self.new(list, name)
        self.freq := 440.0
        self.band := 50.0
        self.setrecalc()
        self.reset()
ENDPROC

/*-------------------------------------------------------------------------*/

PROC param2id(str) OF bandfilter
        IF strcmp(IDS_BANDWIDTH, str) THEN RETURN ID_BANDWIDTH
ENDPROC SUPER self.param2id(str)

/*-------------------------------------------------------------------------*/

PROC id2param(id) OF bandfilter
        IF id = ID_BANDWIDTH THEN RETURN IDS_BANDWIDTH
ENDPROC SUPER self.id2param(id)

/*-------------------------------------------------------------------------*/

PROC paramtype(id) OF bandfilter
        IF id = ID_BANDWIDTH THEN RETURN TYPE_NUMBER
ENDPROC SUPER self.paramtype(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF bandfilter
        IF id = ID_BANDWIDTH
                IF ! data <= 0.0 THEN Throw(ERR_BAD_RANGE, id)
                self.band := data
                self.setrecalc()
        ELSE
                SUPER self.set(id, data)
        ENDIF
ENDPROC

/*-------------------------------------------------------------------------*/

PROC get(id) OF bandfilter
        IF id = ID_BANDWIDTH THEN RETURN self.band
ENDPROC SUPER self.get(id)

/*-------------------------------------------------------------------------+
| END: bandfilter.e                                                        |
+=========================================================================*/
```

8 Source/bandpass.e

```
/*=========================================================================+
| bandpass.e                                                               |
| Effect class "bandpass", band pass filter                                |
```

```
+-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*bandfilter', '*link'

EXPORT OBJECT bandpass OF bandfilter
ENDOBJECT

PROC class() OF bandpass IS 'bandpass'

PROC new(list, name) OF bandpass
        SUPER self.new(list, name)
        self.setrecalc()
        self.reset()
ENDPROC

PROC recalc() OF bandpass
        DEF c, d, in : PTR TO link
        SUPER self.recalc()
        in := self.getinput(ID_MAIN)
        c := ! 1.0 / Ftan(! PI  * self.band / in.rate)
        d := ! 2.0 * Fcos(! PI2 * self.freq / in.rate)
        self.a0 := ! 1.0 / (! 1.0 + c)
        self.a1 := 0.0
        self.a2 := ! -self.a0
        self.b1 := ! c * d * self.a0
        self.b2 := ! (! 1.0 - c) * self.a0
ENDPROC

/*-------------------------------------------------------------------------+
| END: bandpass.e                                                          |
+=========================================================================*/
```

9 Source/bandreject.e

```
/*=========================================================================+
| bandreject.e                                                             |
| Effect class "bandreject", band reject filter                            |
+-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*bandfilter', '*link'

EXPORT OBJECT bandreject OF bandfilter
ENDOBJECT

PROC class() OF bandreject IS 'bandreject'

PROC new(list, name) OF bandreject
        SUPER self.new(list, name)
        self.setrecalc()
        self.reset()
ENDPROC

PROC recalc() OF bandreject
```

```
        DEF c, d, in : PTR TO link
        SUPER self.recalc()
        in := self.getinput(ID_MAIN)
        c := ! Ftan(! PI  * self.band / in.rate)
        d := ! 2.0 * Fcos(! PI2 * self.freq / in.rate)
        self.a0 := ! 1.0 / (! 1.0 + c)
        self.a1 := ! -d * self.a0
        self.a2 := ! self.a0
        self.b1 := ! -self.a1
        self.b2 := ! (! c - 1.0) * self.a0
ENDPROC

/*-----------------------------------------------------------------------+
| END: bandreject.e                                                      |
+=======================================================================*/
```

10 Source/.build

```
#=========================================================================#
# .build                                                                  #
# Build file                                                              #
#                                                                         #
# start     build everything (default target)                             #
# clean     remove output files                                           #
# <...>     build that module (listed in alphabetical order)              #
#                                                                         #
#-------------------------------------------------------------------------#

start: main testcbuffer testrnd e2txt.script e2txt_sub.script
        ;Echo "*nJoining sources to print...*n"
        ;Execute e2txt.script
        Echo "*nMaking distribution...*n"
        Copy main       /SLab
        Copy SLab.info /SLab.info
        Echo "*nAll built!*n"

#-------------------------------------------------------------------------#

clean:
        Echo "Cleaning..."
        Delete QUIET #?.m main testcbuffer testrnd
        Echo "All cleaned!"

#-------------------------------------------------------------------------#

add.m: add.e defs.m inmout1.m debug.m
        Echo "add.e"
        EC QUIET add.e

amp.m: amp.e defs.m in1out1.m string.m value.m debug.m
        Echo "amp.e"
        EC QUIET amp.e

bandfilter.m: bandfilter.e defs.m filter.m link.m string.m value.m debug.m
        Echo "bandfilter.e"
        EC QUIET bandfilter.e
```

```
bandpass.m: bandpass.e bandfilter.m defs.m link.m debug.m
        Echo "bandpass.e"
        EC QUIET bandpass.e

bandreject.m: bandreject.e bandfilter.m defs.m link.m debug.m
        Echo "bandreject.e"
        EC QUIET bandreject.e

cbuffer.m: cbuffer.e debug.m
        Echo "cbuffer.e"
        EC QUIET cbuffer.e

classnode.m: classnode.e list.m debug.m
        Echo "classnode.e"
        EC QUIET classnode.e

cli.m: cli.e defs.m kernel.m list.m string.m value.m debug.m
        Echo "cli.e"
        EC QUIET cli.e

constant.m: constant.e defs.m in0out1.m string.m value.m debug.m
        Echo "constant.e"
        EC QUIET constant.e

container.m: container.e defs.m effect.m hack.m list.m debug.m
        Echo "container.e"
        EC QUIET container.e

copy.m: copy.e defs.m in1out1.m debug.m
        Echo "copy.e"
        EC QUIET copy.e

debug.m: debug.e
        Echo "debug.e"
        EC QUIET debug.e

defs.m: defs.e
        Echo "defs.e"
        EC QUIET defs.e

delay.m: delay.e cbuffer.m defs.m in1out1.m link.m string.m value.m debug.m
        Echo "delay.e"
        EC QUIET delay.e

echo.m: echo.e add.m amp.m container.m defs.m fbdelay.m feedback.m split.m ↙
    ↳ string.m debug.m
        Echo "echo.e"
        EC QUIET echo.e

effect.m: effect.e defs.m list.m debug.m
        Echo "effect.e"
        EC QUIET effect.e

exp.m: exp.e defs.m debug.m
        Echo "exp.e"
        EC QUIET exp.e
```

```
fbdelay.m: fbdelay.e cbuffer.m defs.m delay.m debug.m
        Echo "fbdelay.e"
        EC QUIET fbdelay.e

feedback.m: feedback.e defs.m in1out1.m debug.m
        Echo "feedback.e"
        EC QUIET feedback.e

file.m: file.e
        Echo "file.e"
        EC QUIET file.e

filter.m: filter.e defs.m in1out1.m string.m value.m debug.m
        Echo "filter.e"
        EC QUIET filter.e

hack.m: hack.e
        Echo "hack.e"
        EC QUIET hack.e

halfrectify.m: halfrectify.e defs.m in1out1.m debug.m
        Echo "halfrectify.e"
        EC QUIET halfrectify.e

highpass.m: highpass.e defs.m filter.m link.m debug.m
        Echo "highpass.e"
        EC QUIET highpass.e

iff8svx_ff.m: iff8svx_ff.e
        Echo "iff8svx_ff.e"
        EC QUIET iff8svx_ff.e

in0out1.m: in0out1.e defs.m effect.m link.m string.m debug.m
        Echo "in0out1.e"
        EC QUIET in0out1.e

in1out0.m: in1out0.e defs.m effect.m link.m string.m debug.m
        Echo "in1out0.e"
        EC QUIET in1out0.e

in1out1.m: in1out1.e defs.m effect.m link.m string.m debug.m
        Echo "in1out1.e"
        EC QUIET in1out1.e

in1outm.m: in1outm.e defs.m effect.m link.m string.m value.m debug.m
        Echo "in1outm.e"
        EC QUIET in1outm.e

inmout1.m: inmout1.e defs.m effect.m link.m string.m value.m debug.m
        Echo "inmout1.e"
        EC QUIET inmout1.e

invert.m: invert.e defs.m in1out1.m debug.m
        Echo "invert.e"
        EC QUIET invert.e

kernel.m: kernel.e add.m amp.m bandpass.m bandreject.m classnode.m constant.m ↙
```

```
    ↳ copy.m defs.m delay.m echo.m effect.m exp.m fbdelay.m feedback.m hack.m ↙
    ↳ halfrectify.m highpass.m invert.m link.m link_.m list.m lowpass.m mul.m ↙
    ↳ notch.m print.m pulse.m ramp.m read8svx.m readslab.m rectify.m scale.m ↙
    ↳ sine.m split.m string.m toparam.m triangle.m value.m vox.m whitenoise.m ↙
    ↳ write8svx.m writeslab.m zfilter.m debug.m
        Echo "kernel.e"
        EC QUIET kernel.e

link.m: link.e defs.m effect.m debug.m
        Echo "link.e"
        EC QUIET link.e

link_.m: link_.e link.m
        Echo "link_.e"
        EC QUIET link_.e

list.m: list.e string.m debug.m
        Echo "list.e"
        EC QUIET list.e

lowpass.m: lowpass.e defs.m filter.m link.m debug.m
        Echo "lowpass.e"
        EC QUIET lowpass.e

# LARGE code / data model needed as executable size > 32 kB
main: main.e cli.m rnd.m string.m debug.m
        Echo "main.e"
        EC QUIET LARGE main.e

mul.m: mul.e defs.m inmout1.m debug.m
        Echo "mul.e"
        EC QUIET mul.e

notch.m: notch.e container.m defs.m string.m zfilter.m debug.m
        Echo "notch.e"
        EC QUIET notch.e

osc.m: osc.e defs.m in0out1.m string.m value.m debug.m
        Echo "osc.e"
        EC QUIET osc.e

print.m: print.e defs.m in1out0.m list.m debug.m
        Echo "print.e"
        EC QUIET print.e

pulse.m: pulse.e defs.m osc.m string.m value.m debug.m
        Echo "pulse.e"
        EC QUIET pulse.e

ramp.m: ramp.e osc.m
        Echo "ramp.e"
        EC QUIET ramp.e

read.m: read.e defs.m in0out1.m string.m value.m debug.m
        Echo "read.e"
        EC QUIET read.e
```

```
read8svx.m: read8svx.e defs.m file.m read.m iff8svx_ff.m debug.m
        Echo "read8svx.e"
        EC QUIET read8svx.e

readslab.m: readslab.e defs.m file.m read.m slab_ff.m debug.m
        Echo "readslab.e"
        EC QUIET readslab.e

rectify.m: rectify.e defs.m in1out1.m debug.m
        Echo "rectify.e"
        EC QUIET rectify.e

rnd.m: rnd.e
        Echo "rnd.e"
        EC QUIET rnd.e

scale.m: scale.e defs.e in1out1.m
        Echo "scale.e"
        EC QUIET scale.e

sine.m: sine.e osc.m
        Echo "sine.e"
        EC QUIET sine.e

slab_ff.m: slab_ff.e
        Echo "slab_ff.e"
        EC QUIET slab_ff.e

split.m: split.e defs.m in1outm.m debug.m
        Echo "split.e"
        EC QUIET split.e

string.m: string.e debug.m
        Echo "string.e"
        EC QUIET string.e

testcbuffer: testcbuffer.e cbuffer.m debug.m
        Echo "testcbuffer.e"
        EC QUIET testcbuffer.e

testrnd: testrnd.e rnd.m
        Echo "testrnd.e"
        EC QUIET testrnd.e

toparam.m: toparam.e defs.m effect.m in1out0.m string.m value.m debug.m
        Echo "toparam.e"
        EC QUIET toparam.e

triangle.m: triangle.e osc.m
        Echo "triangle.e"
        EC QUIET triangle.e

value.m: value.e debug.m
        Echo "value.e"
        EC QUIET value.e

vox.m: vox.e defs.m in1out1.m string.m value.m debug.m
```

```
        Echo "vox.e"
        EC QUIET vox.e

whitenoise.m: whitenoise.e defs.m in0out1.m rnd.m debug.m
        Echo "whitenoise.e"
        EC QUIET whitenoise.e

write.m: write.e defs.m in1out0.m string.m value.m debug.m
        Echo "write.e"
        EC QUIET write.e

write8svx.m: write8svx.e defs.m file.m iff8svx_ff.m rnd.m write.m debug.m
        Echo "write8svx.e"
        EC QUIET write8svx.e

writeslab.m: writeslab.e defs.m file.m slab_ff.m write.m debug.m
        Echo "writeslab.e"
        EC QUIET writeslab.e

# lots of loops => OPTImise
zfilter.m: zfilter.e cbuffer.m defs.m in1out1.m link.m string.m value.m debug.m
        Echo "zfilter.e"
        EC QUIET OPTI zfilter.e

#------------------------------------------------------------------------------#
# END: .build                                                                  #
#==============================================================================#
```

11 Source/cbuffer.e

```
/*============================================================================+
| cbuffer.e                                                                   |
| Circular buffer class                                                       |
|                                                                             |
| For correct behaviour with short lengths, write read next should be done    |
| in that order (or rotation: rnw, nwr)                                       |
|                                                                             |
| cbuffer.new(length)          constructor, length is float, 1.5 times the    |
|                              length is allocated                            |
| cbuffer.end()                destructor                                     |
| cbuffer.length()             get the current length (float)                 |
| cbuffer.setlength(length)    set the length (float), read position is       |
|                              changed, memory is reallocated if necessary    |
| cbuffer.read()               read from the current read position, using      |
|                              linear interpolation                           |
| cbuffer.readrel(offset)      read relative (offset is float) to the         |
|                              current read position, using linear            |
|                              interpolation ( -length <= offset <= 0.0)      |
| cbuffer.write(data)          store (float) at the current write position     |
| cbuffer.next()               move to next position in buffer                |
| cbuffer.clear()              reset contents to zero                         |
|                                                                             |
+----------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*debug'
```

```
    RAISE "MEM" IF New() = NIL      -> Automatic exceptions

/*-----------------------------------------------------------------------*/

EXPORT OBJECT cbuffer
PRIVATE
        maxlen  : LONG          -> integer physical length
        l       : LONG          -> float length
        r       : LONG          -> float read position
        w       : LONG          -> integer write position
        data    : PTR TO LONG   -> floats
ENDOBJECT

-> E cannot have float constants
#define MINLEN 16.0

/*-----------------------------------------------------------------------*/

PROC new(length) OF cbuffer      -> length is float
        assert(! length >= 0.0, 'cbuffer.new')
        self.maxlen := ! (! length * 1.5) + MINLEN !   -> allow space for growth
        self.l := length
        self.r := self.maxlen - 1 ! - length
        self.w := self.maxlen - 1
        self.data := New(self.maxlen * SIZEOF LONG)
ENDPROC

/*-----------------------------------------------------------------------*/

PROC end() OF cbuffer IS Dispose(self.data)

/*-----------------------------------------------------------------------*/

PROC length() OF cbuffer IS self.l

/*-----------------------------------------------------------------------*/

PROC setlength(length) OF cbuffer       -> length is float
        DEF newmaxlen, newdata : PTR TO LONG, i, offset
        assert(! length >= 0.0, 'cbuffer.setlength')
        IF ! length ! < (self.maxlen - 2)
                -> Enough space, change pointer
                self.r := wrapf(! self.r + self.l - length, self.maxlen !)
                self.l := length
        ELSE
                -> Not enough space, allocate new
                newmaxlen := ! (! length * 1.5) + MINLEN !
                newdata   := New(newmaxlen * SIZEOF LONG)
                -> Copy data up to (including) write position to end of buffer
                -> (Correct, consider new index for max i)
                offset := newmaxlen - self.w - 1
                FOR i := 0 TO self.w
                        newdata[offset + i] := self.data[i]
                ENDFOR
                -> Copy data after write pointer (ie, long before) to before ↙
                    ↳ that
```

```
                    -> (Correct, consider new index for max i, and above index at i ↙
                        ↳ = 0)
85                  offset := newmaxlen - self.w - 1 - self.maxlen
                    FOR i := self.w + 1 TO self.maxlen - 1
                            newdata[offset + i] := self.data[i]
                    ENDFOR
                    -> Clear rest of buffer
90                  FOR i := 0 TO newmaxlen - self.w - self.maxlen - 2
                            newdata[i] := 0.0
                    ENDFOR
                    -> Replace old with new
                    Dispose(self.data)
95                  self.maxlen := newmaxlen
                    self.data   := newdata
                    self.r      := newmaxlen - 1 ! - length
                    self.w      := newmaxlen - 1
                    self.l      := length
100         ENDIF
    ENDPROC

    /*-------------------------------------------------------------------------*/

105 PROC read() OF cbuffer                  -> using linear interpolation
        DEF x0, x1, y0, y1, dx, dy, y
        assert(self.data, 'cbuffer.read.data')
        x0 := ! wrapf(Ffloor(self.r), self.maxlen !) !
        x1 := ! wrapf(Fceil (self.r), self.maxlen !) !
110     y0 := self.data[x0]
        y1 := self.data[x1]
        dx := ! self.r - Ffloor(self.r)
        dy := ! y1 - y0
        y  := ! y0 + (! dx * dy)
115 ENDPROC y

    /*-------------------------------------------------------------------------*/

    PROC readrel(offset) OF cbuffer         -> using linear interpolation
120     DEF r, x0, x1, y0, y1, dx, dy, y
        assert((!-self.l<=offset) AND (!offset<=0.0),'cbuffer.readrel.offset')
        assert(self.data, 'cbuffer.readrel.data')
        r  := ! self.r - offset
        x0 := ! wrapf(Ffloor(r), self.maxlen !) !
125     x1 := ! wrapf(Fceil (r), self.maxlen !) !
        y0 := self.data[x0]
        y1 := self.data[x1]
        dx := ! r - Ffloor(r)
        dy := ! y1 - y0
130     y  := ! y0 + (! dx * dy)
    ENDPROC y

    /*-------------------------------------------------------------------------*/

135 PROC write(data) OF cbuffer
        self.data[self.w] := data
    ENDPROC

    /*-------------------------------------------------------------------------*/
```

```
140
PROC next() OF cbuffer
        self.w := wrapi(self.w + 1, self.maxlen)
        self.r := wrapf(! self.r + 1.0, self.maxlen !)
        assert((0 <= self.w) AND (self.w < self.maxlen), 'cbuffer.next.w')
145     assert((!0.0 <= self.r) AND (!self.r < (self.maxlen!)), 'cbuffer.next.r↙
            ↳ ')
ENDPROC

/*-----------------------------------------------------------------------*/

150 PROC clear() OF cbuffer
        DEF i
        assert(self.data, 'cbuffer.clear')
        FOR i := 0 TO self.maxlen - 1 DO self.data[i] := 0.0
ENDPROC
155
/*-----------------------------------------------------------------------*/

-> Wrap a float to between 0.0 and length (length > 0.0)
PROC wrapf(x, length)
160     assert(! length > 0.0, 'cbuffer.wrapf.length')
        WHILE ! x >= length
                x := ! x - length
        ENDWHILE
        WHILE ! x < 0.0
165             x := ! x + length
        ENDWHILE
        assert((! 0.0 <= x) AND (! x < length), 'cbuffer.wrapf.x')
ENDPROC x

170 /*-----------------------------------------------------------------------*/

-> Wrap an integer to between 0 and length (length > 0)
PROC wrapi(x, length)
        assert(length > 0, 'cbuffer.wrapi.length')
175     WHILE x >= length
                x := x - length
        ENDWHILE
        WHILE x < 0
                x := x + length
180     ENDWHILE
        assert((0 <= x) AND (x < length), 'cbuffer.wrapi.x')
ENDPROC x

/*-----------------------------------------------------------------------+
185 | END: cbuffer.e                                                      |
+=======================================================================*/
```

12 Source/classnode.e

```
/*=======================================================================+
| classnode.e                                                            |
| Node structure for list of classes                                     |
|                                                                        |
  5 | classnode.newf    address of a function with arguments "list, name" that  |
|                   returns a new object of that class, linked into list |
```

```
+-----------------------------------------------------------------------------*/

OPT MODULE

MODULE '*list'

EXPORT OBJECT classnode OF node
        newf    : LONG
ENDOBJECT

/*
        classnode.newf := {eg}
        PROC eg(list, name, o = NIL : PTR TO eg) IS NEW o.new(list, name)
*/

/*-----------------------------------------------------------------------------+
| END: classnode.e                                                             |
+=============================================================================*/
```

13 Source/cli.e

```
/*=============================================================================+
| cli.e                                                                        |
| Command line interface                                                       |
|                                                                              |
| cli.new(in, out, err)     constructor; in, out, err are files for text IO    |
| cli.end()                 destructor (called via END)                        |
| cli.parse(str)            parse and execute a command string, terminated     |
|                           in "\n"; the string is modified; returns TRUE      |
|                           if command was quit                                |
|                                                                              |
| The following should not be called from outside the class                    |
|                                                                              |
| cli.parsenew(str)         \   parse the arguments (str) for the command      |
| cli.parsedelete(str)      |   and execute it                                 |
| cli.parselink(str)        |                                                  |
| cli.parseset(str)         |                                                  |
| cli.parserun(str)         |                                                  |
| cli.parselist(str)        /                                                  |
| cli.getident(str)         (destructively) get an identifier, returning       |
|                           the identifier and the new location                |
| cli.error(err, info = 0)  print an error message for an error id             |
| cli.readargs(template, args, str, n)                                         |
|                           interface to dos.library/ReadArgs(), assumes       |
|                           n simple strings to be copied (uses String() to    |
|                           allocate so free with DisposeLink()                |
|                                                                              |
+-----------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

RAISE "MEM" IF String() = NIL

MODULE '*defs', '*kernel', '*list', '*string', '*value', '*effect', '*hack',
       'dos/dos', 'dos/rdargs', 'tools/ctype', '*debug'

/*-----------------------------------------------------------------------------*/
```

```
EXPORT OBJECT cli
PRIVATE
        kernel  : PTR TO kernel
ENDOBJECT

/*------------------------------------------------------------------------*/

PROC new() OF cli
        NEW self.kernel.new()
ENDPROC

/*------------------------------------------------------------------------*/

PROC end() OF cli
        IF self.kernel THEN END self.kernel
ENDPROC

/*------------------------------------------------------------------------*/

PROC parse(s : PTR TO CHAR) OF cli HANDLE
        DEF com
        -> Strip white space
        WHILE isspace(s[]) DO s++
        -> Check for empty command
        IF (s[0] = 0) OR (s[0] = "\n") THEN RETURN FALSE
        -> Get command (in length order for efficiency)
        com, s := self.getident(s)
        IF      strcmp('new',    com);   self.parsenew(s)
        ELSEIF strcmp('run',    com);   self.parserun(s)
        ELSEIF strcmp('set',    com);   self.parseset(s)
        ELSEIF strcmp('link',   com);   self.parselink(s)
        ELSEIF strcmp('list',   com);   self.parselist(s)
        ELSEIF strcmp('quit',   com);   RETURN TRUE
        ELSEIF strcmp('reset',  com);   self.kernel.reset()
        ELSEIF strcmp('delete', com);   self.parsedelete(s)
        ELSE;  Throw(ERR_UNKNOWN_COMMAND, com)
        ENDIF
EXCEPT
        self.error(exception, exceptioninfo)
ENDPROC FALSE

/*------------------------------------------------------------------------*/

PROC parsenew(s : PTR TO CHAR) OF cli
        DEF args[2] : ARRAY OF LONG
        args[0] := NIL
        args[1] := NIL
        self.readargs('TYPE,NAME', args, s, 2)
        self.kernel.new_(args[0], args[1])
ENDPROC

/*------------------------------------------------------------------------*/

PROC parsedelete(s : PTR TO CHAR) OF cli
        DEF args[1] : ARRAY OF LONG
        args[0] := NIL
```

```
        self.readargs('NAME', args, s, 1)
        self.kernel.delete(args[0])
ENDPROC

/*-----------------------------------------------------------------------*/

PROC parselink(s : PTR TO CHAR) OF cli
        DEF args[4] : ARRAY OF LONG
        args[0] := NIL
        args[1] := NIL
        args[2] := NIL
        args[3] := NIL
        self.readargs('FROM,OUTPUT,TO,INPUT', args, s, 4)
    self.kernel.link(args[0], args[1], args[2], args[3])
ENDPROC

/*-----------------------------------------------------------------------*/

PROC parseset(s : PTR TO CHAR) OF cli
        DEF args[3] : ARRAY OF LONG, type, num, len, val : value,
            op : value_objpart, opargs[2] : ARRAY OF LONG, str : PTR TO CHAR
        args[0] := NIL
        args[1] := NIL
        args[2] := NIL
        self.readargs('NAME,PARAM,VAL/F', args, s, 3)     -> /F = rest of line
        val.type := self.kernel.paramtype(args[0], args[1])
        type := val.type
        SELECT type
        CASE TYPE_NUMBER
                num, len := RealVal(args[2])
                IF len <> StrLen(args[2]) THEN Throw(ERR_BAD_NUMBER, args[2])
                val.data := num
        CASE TYPE_STRING
                val.data := args[2]
        CASE TYPE_OBJPART
                -> put \n at end of string
                str := StrCopy(String(StrLen(args[2]) + 2), args[2])
                str[StrLen(args[2]) + 1] := 0
                str[StrLen(args[2])    ] := "\n"
                opargs[0] := NIL
                opargs[1] := NIL
                self.readargs('NAME,PART', opargs, str, 2)
                op.obj := opargs[0]
                op.pid := opargs[1]
                val.data := op
        DEFAULT
                Throw(ERR_BAD_PARAM_TYPE, type)
        ENDSELECT
        self.kernel.set_(args[0], args[1], val)
ENDPROC

/*-----------------------------------------------------------------------*/

PROC parserun(s : PTR TO CHAR) OF cli
        WHILE isspace(s[0]) AND (s[0] <> "\n") DO s++
        IF (s[0] <> "\n") AND (s[0] <> 0) THEN Throw(ERR_BAD_ARGS, '')
    self.kernel.run()
```

```
ENDPROC

/*-----------------------------------------------------------------------*/

PROC parselist(str) OF cli      -> compiler warning, unreferenced str
        DEF list : PTR TO list, node : PTR TO node, args[1] : ARRAY OF LONG,
            fx : PTR TO effect
        args[0] := NIL
        self.readargs('NAME', args, str, 1)
        IF strcmp('.', args[0])
                PrintF('** Effect types:\n')
                list := self.kernel.classes
                node := list.head.next
                WHILE node.next
                        PrintF('**\t\s\n', node.name)
                        node := node.next
                ENDWHILE
        ELSEIF strcmp('*', args[0])
                PrintF('** Effect objects:\n')
                list := self.kernel.objects
                node := list.head.next
                WHILE node.next
                        PrintF('**\t\s\n', node.name)
                        node := node.next
                ENDWHILE
        ELSE
                -> List object parameters / type properties ?
                node := find(self.kernel.objects, args[0])
                IF node
                        fx := __node2effect(node)
                        PrintF('** Name: "\s"\n** Type: "\s"\n', node.name, fx.↙
                            ↳ class())
                ENDIF
        ENDIF
ENDPROC

/*-----------------------------------------------------------------------*/

PROC getident(s : PTR TO CHAR) OF cli
        DEF ident
        ident := s
        IF isalpha(s[0])
                s++
                WHILE isalnum(s[0]) OR (s[0] = "_") DO s++
                IF isspace(s[0]) OR (s[0] = 0) OR (s[0] = "\n")
                        s[0] := 0
                        s++
                        RETURN ident, s
                ENDIF
                -> s is at bad char
        ENDIF
        Throw(ERR_BAD_COMMAND_LINE, s)
ENDPROC

/*-----------------------------------------------------------------------*/

-> NB: dos.library/ReadArgs() expects the string to be \n terminated
```

```
    -> The data *must* be copied from the structure returned before freeing
    PROC readargs(template, args : PTR TO LONG, string, n) OF cli
            DEF rdargs : PTR TO rdargs, i
210         IF rdargs := AllocDosObject(DOS_RDARGS, NIL)
                rdargs.source.buffer := string
            rdargs.source.length := StrLen(string)
            IF ReadArgs(template, args, rdargs)
                            FOR i := 0 TO n - 1     -> copy string
215                             args[i] := StrCopy(String(StrLen(args[i]) + 1), ↙
                                    ↳ args[i])
                            ENDFOR
                            FreeArgs(rdargs)
                    FreeDosObject(DOS_RDARGS, rdargs)
                    ELSE
220                 FreeDosObject(DOS_RDARGS, rdargs)
                    self.error(ERR_BAD_ARGS, template)
            ENDIF
        ELSE
                    Raise(ERR_ALLOC_RDARGS)
225     ENDIF
    ENDPROC

    /*-----------------------------------------------------------------------*/

230 PROC error(error, data = NIL : PTR TO CHAR) OF cli
            DEF prompt, datal : PTR TO LONG, fx : PTR TO effect, id
            prompt := '** Error: '
            SELECT error
            CASE ERR_OK
235                 -> No error
            CASE ERR_NO_SUCH_CLASS
                    PrintF('\seffect type "\s" not found\n', prompt, data)
            CASE ERR_NO_SUCH_NAME
                    PrintF('\seffect object "\s" not found\n', prompt, data)
240         CASE ERR_NAME_ALREADY_USED
                    PrintF('\seffect object "\s" already exists\n', prompt, data)
            CASE ERR_NO_SUCH_INPUT
                    PrintF('\sinput "\s" not found\n', prompt, data)
            CASE ERR_NO_SUCH_OUTPUT
245                 PrintF('\soutput "\s" not found\n', prompt, data)
            CASE ERR_NO_SUCH_PARAM
                    PrintF('\sparameter "\s" not found\n', prompt, data)
            CASE ERR_UNKNOWN_COMMAND
                    PrintF('\sunknown command "\s"\n', prompt, data)
250         CASE ERR_BAD_ARGS
                    PrintF('\sbad arguments for "\s"\n', prompt, data)
            CASE ERR_BAD_RANGE
                    PrintF('\sparameter out of allowed range\n', prompt)
            CASE ERR_PARAM_NOT_NUMBER
255                 PrintF('\sparameter "\s" is not numeric\n', prompt, data)
            CASE ERR_BAD_COMMAND_LINE
                    -> Print one character as unexpected
                    PrintF('\sunexpected character "\s"\n', prompt, [data[0],0] : ↙
                        ↳ CHAR)
            CASE ERR_BAD_NUMBER
260                 PrintF('\snumber expected "\s"\n', prompt, data)
            CASE ERR_CHECK
```

```
                PrintF('\sprocessing failed, check linkage\n', prompt)
cli_error_check_container_recurse:      -> fake recursion for container
                datal := data
                error := datal[0]
                fx := datal[1]
                id := datal[2]
                SELECT error
                CASE CHECK_OK
                        PrintF('\sno error?\n', prompt)
                CASE CHECK_OUTPUT_NOT_CONNECTED
                        PrintF('\soutput "\s" of "\s" not connected\n', prompt,
                                                fx.id2output(id), fx.node.↙
                                                    ↳ name)
                CASE CHECK_INPUT_NOT_CONNECTED
                        PrintF('\sinput "\s" of "\s" not connected\n', prompt,
                                                fx.id2input(id), fx.node.↙
                                                    ↳ name)
                CASE CHECK_INVALID_PARAM
                        PrintF('\sparameter "\s" of "\s" is invalid\n', prompt,
                                                fx.id2param(id), fx.node.↙
                                                    ↳ name)
                CASE CHECK_BAD_INPUT_RATE
                        PrintF('\sinput "\s" of "\s" has a bad rate\n', prompt,
                                                fx.id2input(id), fx.node.↙
                                                    ↳ name)
                CASE CHECK_INTERNAL_ERROR
                        PrintF('\sinternal error\n', prompt)
                CASE CHECK_CONTAINER
                        PrintF('\serror in "\s":\n', prompt, fx.node.name)
                        data := id                      -> fake recursion
                        JUMP cli_error_check_container_recurse
                DEFAULT
                        PrintF('\sunknown\n')
                ENDSELECT
        DEFAULT
                Throw(error, data)
        ENDSELECT
ENDPROC

/*-----------------------------------------------------------------------+
| END: cli.e                                                             |
+=======================================================================*/
```

14 Source/constant.e

```
/*=======================================================================+
| constant.e                                                             |
| Effect class "constant", outputs a constant, set by parameter "value"  |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*in0out1', '*string', '*value'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT constant OF in0out1
```

```
PRIVATE
        value    : LONG
ENDOBJECT

PROC class() OF constant IS 'constant'

/*-----------------------------------------------------------------------*/

PROC new(list, name) OF constant
        SUPER self.new(list, name)
        self.value := 0.0
ENDPROC

/*-----------------------------------------------------------------------*/

PROC param2id(str) OF constant
ENDPROC IF strcmp(IDS_VALUE, str) THEN ID_VALUE ELSE SUPER self.param2id(str)

PROC id2param(id) OF constant
ENDPROC IF id = ID_VALUE THEN IDS_VALUE ELSE SUPER self.id2param(id)

PROC paramtype(id) OF constant
ENDPROC IF id = ID_VALUE THEN TYPE_NUMBER ELSE SUPER self.paramtype(id)

PROC get(id) OF constant
ENDPROC IF id = ID_VALUE THEN self.value ELSE SUPER self.get(id)

/*-----------------------------------------------------------------------*/

PROC set(id, data) OF constant
        SELECT id
        CASE ID_VALUE;   self.value := data
        DEFAULT;         SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-----------------------------------------------------------------------*/

PROC process() OF constant
        SUPER self.process()
        self.output(ID_MAIN, self.value)
ENDPROC

/*-----------------------------------------------------------------------+
| END: constant.e                                                        |
+=======================================================================*/
```

15 Source/container.e

```
/*=======================================================================+
| container.e                                                            |
| Effect base class "container"                                          |
|                                                                        |
| Base class for effects built up from others. An internal list is kept of |
| the effect objects making up the compound effect. No inputs, outputs or |
| parameters are defined. Objects are the responsibility of the container, |
| which handles most functions. Derived classes add objects and delegate |
```

```
    | inputs, outputs and parameters.                                          |
10  |                                                                          |
    | See echo.e and notch.e for examples.                                     |
    |                                                                          |
    | Members are added using the public list as an argument for new.          |
    |                                                                          |
15  | Note: the fix in kernel to move toparam objects to the end of the list is |
    | not present, so they must be added last.                                 |
    |                                                                          |
    | container.new(list, name)     initialises the internal list of members   |
    | container.end()               ends all members                           |
20  | container.clear()             clears all members                         |
    | container.reset()             resets all members                         |
    | container.process()           called if issource is set, calls process   |
    |                               for all members that have issource set     |
    | container.check()             checks all members that have issource set  |
25  |                                                                          |
    +--------------------------------------------------------------------------*/

    OPT MODULE, PREPROCESS

30  MODULE '*defs', '*effect', '*hack', '*list'

    /*--------------------------------------------------------------------------*/

    EXPORT OBJECT container OF effect
35  PUBLIC
        list : list     -> list of member objects (needed by derived classes)
    ENDOBJECT

    PROC class() OF container IS 'container'
40
    /*--------------------------------------------------------------------------*/

    PROC new(list, name) OF container
        SUPER self.new(list, name)
45      newlist(self.list)
    ENDPROC

    /*--------------------------------------------------------------------------*/

50  PROC end() OF container
        DEF n : PTR TO node, fx : PTR TO effect, t
        n := self.list.head.next
        WHILE (t := n.next) <> NIL
            fx := __node2effect(n)
55          END fx
            n := t
        ENDWHILE
        SUPER self.end()
    ENDPROC
60
    /*--------------------------------------------------------------------------*/

    PROC process() OF container
        DEF n : PTR TO node, fx : PTR TO effect
65      SUPER self.process()
```

```
        n := self.list.head.next
        WHILE n.next <> NIL
                fx := __node2effect(n)
                IF fx.issource() THEN fx.process()
                n := n.next
        ENDWHILE
ENDPROC

/*-----------------------------------------------------------------------*/

PROC clear() OF container
        DEF n : PTR TO node, fx : PTR TO effect
        SUPER self.clear()
        n := self.list.head.next
        WHILE n.next <> NIL
                fx := __node2effect(n)
                fx.clear()
                n := n.next
        ENDWHILE
ENDPROC

/*-----------------------------------------------------------------------*/

PROC reset() OF container
        DEF n : PTR TO node, fx : PTR TO effect
        SUPER self.reset()
        n := self.list.head.next
        WHILE n.next <> NIL
                fx := __node2effect(n)
                fx.reset()
                n := n.next
        ENDWHILE
ENDPROC

/*-----------------------------------------------------------------------*/

PROC check() OF container HANDLE    -> internal objects should not be visible
        DEF n : PTR TO node, fx : PTR TO effect, ok
        IF SUPER self.check() = FALSE THEN RETURN FALSE
        ok := TRUE
        n := self.list.head.next
        WHILE n.next <> NIL
                fx := __node2effect(n)
                IF fx.issource() THEN ok := ok AND fx.check()
                n := n.next
        ENDWHILE
EXCEPT          -> catch errors, pass on with extra information
        IF exception = ERR_CHECK
                Throw(ERR_CHECK, [ CHECK_CONTAINER, self, exceptioninfo ])
        ENDIF
        ReThrow()    -> throw exception if it is <> 0
ENDPROC ok

/*-----------------------------------------------------------------------+
| END: echo.e                                                            |
+=======================================================================*/
```

16 Source/copy.e

```
/*=============================================================================+
| copy.e                                                                       |
| Effect class "copy", copies input to output                                  |
+-----------------------------------------------------------------------------*/

OPT MODULE

MODULE '*defs', '*in1out1'

EXPORT OBJECT copy OF in1out1
ENDOBJECT

PROC class() OF copy IS 'copy'

PROC process() OF copy
        SUPER self.process()
        self.output(ID_MAIN, self._main())
ENDPROC

/*-----------------------------------------------------------------------------+
| END: copy.e                                                                  |
+=============================================================================*/
```

17 Source/debug.e

```
/*=============================================================================+
| debug.e                                                                      |
| Debugging macros                                                             |
|                                                                              |
| DEBUG             this is defined to enable debugging, as a suitable         |
|                   prefix for printing debug info (use                        |
|                   "PrintF(DEBUG 'string',args)")                             |
| debug(x)          this expands to x when debugging is enabled, otherwise     |
|                   to nothing, neater than #ifdef ... #endif                  |
| ASSERT            defined to enable assertion                                |
| assert(x, str)    when assertion is enabled, throws an "asrt" exception      |
|                   with str as info if x is false                             |
|                                                                              |
+-----------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS
OPT EXPORT

/*-----------------------------------------------------------------------------*/

-> Comment [uncomment] next lines to remove [insert] debugging code
->#define DEBUG '++ Debug: '+
->#define ASSERT

/*-----------------------------------------------------------------------------*/

#ifdef DEBUG
#define debug(x) x
#define realf(x) RealF(String(16), x, 8)
```

```
#endif

#ifndef DEBUG
#define debug(x)
#endif

/*-------------------------------------------------------------------------*/

#ifdef ASSERT
#define assert(x,str) IF (x) = FALSE THEN Throw("asrt", str)
#endif

#ifndef ASSERT
#define assert(x,str)
#endif

/*-------------------------------------------------------------------------+
| END: debug.e                                                             |
+=========================================================================*/
```

18 Source/defs.e

```
/*=========================================================================+
| defs.e                                                                   |
| Definitions required throughout the system                               |
|                                                                          |
| ERR_#?     error IDs                                                     |
| ID_#?      IDs for inputs, outputs and parameters                        |
| IDS_#?     names for inputs, outputs and parameters                      |
| PI, PI2    mathematical constants                                        |
| DEF_RATE   default sample rate                                           |
|                                                                          |
+-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS
OPT EXPORT      -> exports all, only way to export macros

/*-------------------------------------------------------------------------*/

ENUM ERR_OK = 0,
      ERR_ALLOC_RDARGS,          ERR_BAD_ARGS,          ERR_BAD_COMMAND_LINE,
      ERR_BAD_FILE_FORMAT,       ERR_BAD_NUMBER,        ERR_BAD_PARAM_TYPE,
      ERR_BAD_RANGE,             ERR_CANT_OPEN_FILE,    ERR_CHECK,
      ERR_NAME_ALREADY_USED,     ERR_NO_SUCH_CLASS,     ERR_NO_SUCH_INPUT,
      ERR_NO_SUCH_INPUTID,       ERR_NO_SUCH_NAME,      ERR_NO_SUCH_OUTPUT,
      ERR_NO_SUCH_OUTPUTID,      ERR_NO_SUCH_PARAM,     ERR_NO_SUCH_PARAMID,
      ERR_PARAM_NOT_NUMBER,      ERR_UNKNOWN_COMMAND

/*-------------------------------------------------------------------------*/

ENUM CHECK_OK = 0,
      CHECK_BAD_INPUT_RATE, CHECK_CONTAINER,       CHECK_INPUT_NOT_CONNECTED,
      CHECK_INTERNAL_ERROR, CHECK_INVALID_PARAM, CHECK_OUTPUT_NOT_CONNECTED

/*-------------------------------------------------------------------------*/

ENUM ID_INVALID = 0,
```

```
        ID_AMPLITUDE,       ID_BANDWIDTH,       ID_DECAY,           ID_DELAY,
        ID_DEPTH,           ID_FILE,            ID_FREQUENCY,       ID_GAIN,
        ID_INPUTS,          ID_LEFT,            ID_MAIN,            ID_NORMALIZE,
        ID_OUTPUTS,         ID_PHASE,           ID_POLES,           ID_RATE,
40      ID_RIGHT,           ID_SIDECHAIN,       ID_THRESHOLD,       ID_TO,
        ID_VALUE,           ID_WIDTH,           ID_ZEROS,

        ID_MULTI_IN      = $1000,  ID_MULTI_OUT     = $2000,
        ID_MULTI_ZERO_R  = $3000,  ID_MULTI_ZERO_F  = $4000,
45      ID_MULTI_POLE_R  = $5000,  ID_MULTI_POLE_F  = $6000

    /*--------------------------------------------------------------------------*/

    #define IDS_AMPLITUDE    'amplitude'
50  #define IDS_BANDWIDTH    'bandwidth'
    #define IDS_DECAY        'decay'
    #define IDS_DELAY        'delay'
    #define IDS_DEPTH        'depth'
    #define IDS_FILE         'file'
55  #define IDS_FREQUENCY    'frequency'
    #define IDS_GAIN         'gain'
    #define IDS_INPUTS       'inputs'
    #define IDS_LEFT         'left'
    #define IDS_MAIN         'main'
60  #define IDS_NORMALIZE    'normalize'
    #define IDS_OUTPUTS      'outputs'
    #define IDS_PHASE        'phase'
    #define IDS_POLES        'poles'
    #define IDS_RATE         'rate'
65  #define IDS_RIGHT        'right'
    #define IDS_SIDECHAIN    'sidechain'
    #define IDS_THRESHOLD    'threshold'
    #define IDS_TO           'to'
    #define IDS_VALUE        'value'
70  #define IDS_WIDTH        'width'
    #define IDS_ZEROS        'zeros'

    #define IDS_IN           'in'
    #define IDS_OUT          'out'
75  #define IDS_POLE         'pole'
    #define IDS_ZERO         'zero'

    #define IDS_ON           'on'
    #define IDS_OFF          'off'
80
    /*--------------------------------------------------------------------------*/

    #define PI  3.14159265
    #define PI2 6.28318531
85
    #define DEF_RATE 44100.0

    /*--------------------------------------------------------------------------+
    | END: defs.e                                                               |
90  +==========================================================================*/
```

19 Source/delay.e

```
/*=============================================================================+
| delay.e                                                                      |
| Effect class "delay", delays input by parameter "delay"                      |
+-----------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*cbuffer', '*defs', '*in1out1', '*link', '*string', '*value',
       '*debug'

/*-----------------------------------------------------------------------------*/

EXPORT OBJECT delay OF in1out1
PUBLIC      -> fbdelay needs access
        cb  : PTR TO cbuffer
        dt  : LONG
ENDOBJECT

PROC class() OF delay IS 'delay'

/*-----------------------------------------------------------------------------*/

PROC new(list, name) OF delay
        SUPER self.new(list, name)
        self.cb := NIL      -> deferred recalc() => cbuffer not allocated now
        self.set(ID_DELAY, 0.05)
ENDPROC

/*-----------------------------------------------------------------------------*/

PROC end() OF delay
        END self.cb
ENDPROC SUPER self.end()

/*-----------------------------------------------------------------------------*/

PROC param2id(str) OF delay
ENDPROC IF strcmp(IDS_DELAY, str) THEN ID_DELAY ELSE SUPER self.param2id(str)

PROC id2param(id) OF delay
ENDPROC IF id = ID_DELAY THEN IDS_DELAY ELSE SUPER self.id2param(id)

PROC paramtype(id) OF delay
ENDPROC IF id = ID_DELAY THEN TYPE_NUMBER ELSE SUPER self.paramtype(id)

PROC get(id) OF delay
ENDPROC IF id = ID_DELAY THEN self.dt ELSE SUPER self.get(id)

/*-----------------------------------------------------------------------------*/

PROC set(id, data) OF delay
        SELECT id
        CASE ID_DELAY
             self.dt := data
             self.setrecalc()
```

```
        DEFAULT; SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-----------------------------------------------------------------------*/

PROC recalc() OF delay
        DEF in  : PTR TO link
        SUPER self.recalc()
        in := self.getinput(ID_MAIN)
        IF self.cb
                self.cb.setlength(! self.dt * in.rate)
        ELSE
                NEW self.cb.new(! self.dt * in.rate)
        ENDIF
ENDPROC

/*-----------------------------------------------------------------------*/

PROC reset() OF delay
        SUPER self.reset()
        self.cb.clear()     -> reset buffer to 0
ENDPROC

/*-----------------------------------------------------------------------*/

PROC process() OF delay
        SUPER self.process()
        self.cb.next()
        self.cb.write(self._main())
        self.output(ID_MAIN, self.cb.read())
ENDPROC

/*-----------------------------------------------------------------------+
| END: delay.e                                                           |
+=======================================================================*/
```

20 Source/e2txt.script

```
Echo  >/Docs/Source.txt "" NOLINE
NL -s #?.e -E"Execute e2txt_sub.script %n"
More /Docs/Source.txt
```

21 Source/e2txt_sub.script

```
.key NAME/A
.bra {
.ket }
Type {NAME} >>/Docs/Source.txt
Echo "*n*n" >>/Docs/Source.txt
```

22 Source/echo.e

```
/*=======================================================================+
| echo.e                                                                 |
| Effect class "echo"                                                    |
```

```
    |                                                                              |
    | Simple echo effect, made up of one each of add, split, fbdelay, amp, and     |
    | feedback.                                                                    |
    |                                                                              |
    | Features: input "main", output "main", parameters "decay" and "delay"        |
    |                                                                              |
    +------------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*add', '*amp', '*container', '*defs', '*fbdelay', '*feedback',
       '*split', '*string', '*link'

/*------------------------------------------------------------------------------*/

EXPORT OBJECT echo OF container
PRIVATE
        add       : PTR TO add          -> for delegation purposes only
        split     : PTR TO split
        fbdelay   : PTR TO fbdelay
        amp       : PTR TO amp
ENDOBJECT

PROC class()    OF echo IS 'echo'
PROC issource() OF echo IS TRUE     -> contains source objects

/*------------------------------------------------------------------------------*/

PROC new(list, name) OF echo
        DEF fb : PTR TO feedback
        SUPER self.new(list, name)
        -> create parts
        NEW self.add    .new(self.list, 'echo_add')
        NEW self.split  .new(self.list, 'echo_split')
        NEW self.fbdelay.new(self.list, 'echo_fbdelay')
        NEW self.amp    .new(self.list, 'echo_amp')
        NEW fb          .new(self.list, 'echo_feedback')
        -> set parameters
        self.add    .set(ID_INPUTS,  2.0)
        self.split  .set(ID_OUTPUTS, 2.0)
        self.fbdelay.set(ID_DELAY,   0.1)
        self.amp    .set(ID_GAIN,    0.5)
        -> link
        link(self.add,     ID_MAIN,          self.split,   ID_MAIN)
        link(self.split,   ID_MULTI_OUT + 2, self.fbdelay, ID_MAIN)
        link(self.fbdelay, ID_MAIN,          self.amp,     ID_MAIN)
        link(self.amp,     ID_MAIN,          fb,           ID_MAIN)
        link(fb,           ID_MAIN,          self.add,     ID_MULTI_IN + 2)
ENDPROC

/*------------------------------------------------------------------------------*/

PROC input2id(str) OF echo
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.input2id(str)

PROC output2id(str) OF echo
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.output2id(str)
```

```
PROC id2input(id) OF echo
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2input(id)

PROC id2output(id) OF echo
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2output(id)

/*-------------------------------------------------------------------------*/

PROC param2id(str) OF echo
        IF      strcmp(IDS_DECAY, str); RETURN ID_DECAY
        ELSEIF strcmp(IDS_DELAY, str); RETURN ID_DELAY
        ENDIF
ENDPROC SUPER self.param2id(str)

/*-------------------------------------------------------------------------*/

PROC id2param(id) OF echo
        SELECT id
        CASE ID_DECAY; RETURN self.amp.id2param(ID_GAIN)
        CASE ID_DELAY; RETURN self.fbdelay.id2param(ID_DELAY)
        ENDSELECT
ENDPROC SUPER self.id2param(id)

/*-------------------------------------------------------------------------*/

PROC paramtype(id) OF echo
        SELECT id
        CASE ID_DECAY; RETURN self.amp.paramtype(ID_GAIN)
        CASE ID_DELAY; RETURN self.fbdelay.paramtype(ID_DELAY)
        ENDSELECT
ENDPROC SUPER self.paramtype(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF echo
        SELECT id
        CASE ID_DECAY; self.amp.set(ID_GAIN, data)
        CASE ID_DELAY; self.fbdelay.set(ID_DELAY, data)
        DEFAULT;       SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC get(id) OF echo
        SELECT id
        CASE ID_DECAY; RETURN self.amp.get(ID_GAIN)
        CASE ID_DELAY; RETURN self.fbdelay.get(ID_DELAY)
        ENDSELECT
ENDPROC SUPER self.get(id)

/*-------------------------------------------------------------------------*/

PROC getinput(id) OF echo
        SELECT id
        CASE ID_MAIN; RETURN self.add.getinput(ID_MULTI_IN + 1)
```

```
        ENDSELECT
ENDPROC SUPER self.getinput(id)

/*-----------------------------------------------------------------------*/

PROC setoutput(id, link) OF echo
        SELECT id
        CASE ID_MAIN; RETURN self.split.setoutput(ID_MULTI_OUT + 1, link)
        ENDSELECT
ENDPROC SUPER self.setoutput(id, link)

/*-----------------------------------------------------------------------+
| END: echo.e                                                            |
+=======================================================================*/
```

23 Source/effect.e

```
/*=======================================================================+
| effect.e                                                               |
| Effect base class                                                      |
|                                                                        |
| Derived classes implementing methods that are passed an id (or a string)  |
| must call SUPER if it is not recognised; the default behaviour is raising |
| an exception (ERR_NO_SUCH_#?ID, id) unless otherwise specified         |
|                                                                        |
| Derived classes may add new methods, these should not be called from   |
| outside the class (or classes derived from it)                         |
|                                                                        |
| Implementation of inputs:  see in1out0.e                               |
| Implementation of outputs: see in0out1.e                               |
| Implementation of params:  see constant.e                              |
| Implementation of recalc:  see delay.e                                 |
|                                                                        |
| effect.new(list, name)        constructor, link named node into list   |
|                               derived classes must call SUPER first    |
| effect.end()                  destructor, remove node from list        |
|                               derived classes must call SUPER last     |
| effect.class()                returns the name of the class, for run time |
|                               inquiries                                |
| effect.input2id(str)          \  get id of str (def = ID_INVALID)      |
| effect.output2id(str)         |                                        |
| effect.param2id(str)          /                                        |
| effect.id2input(id)           \  get str of id (def = NIL)             |
| effect.id2output(id)          |                                        |
| effect.id2param(id)           /                                        |
| effect.getinput(id)           get the link structure of an input        |
| effect.setoutput(id, link)    set an output link pointer               |
| effect.set(id, value)         set a parameter, call setrecalc if the   |
|                               change requires recalculation            |
| effect.get(id)                get a parameter                          |
| effect.paramtype(id)          get the type of a parameter              |
| effect.setrecalc()            signal that parameters have changed that |
|                               require recalculation of internal data,  |
|                               should not be changed in derived classes |
| effect.recalc()               recalculate internal data (def = reset   |
|                               recalc flag), derived classes must call  |
|                              SUPER first                               |
```

```
    | effect.check()                check if set up enough to run, derived      |
    |                               classes should check own data before SUPER  |
    |                               (for efficiency)                            |
    | effect.isready()              check whether enough input to process       |
45  |                               (def = first), derived classes should check |
    |                               own inputs before SUPER (for efficiency)    |
    | effect.issource()             the effect is a source (def = FALSE)        |
    | effect.clear()                clear data, ready for next sample instance, |
    |                               should call SUPER first                     |
50  | effect.reset()                clear data, ready to restart processing,    |
    |                               should call SUPER first, calls clear        |
    | effect.output(id, sample)     send output to destination, should call     |
    |                               outputlink on internal link structure, this |
    |                               method should only be called from process   |
55  | effect.process()              perform processing, default behaviour calls |
    |                               recalc if necessary, derived classes should |
    |                               call SUPER first; the kernel calls process  |
    |                               for effects with issource = TRUE during run |
    |                                                                           |
60  +--------------------------------------------------------------------------*/

    OPT MODULE, PREPROCESS

    MODULE '*defs', '*list', '*debug'
65
    /*--------------------------------------------------------------------------*/

    EXPORT OBJECT effect
    PUBLIC
70          node    : node      -> offset = 4, as class link is at 0 (see hack.e)
    PRIVATE
            recalcf : LONG      -> recalc needs to be called
    ENDOBJECT

75  /*--------------------------------------------------------------------------*/

    -> Construction / destruction
    PROC new(list, name)        OF effect
            self.node.name := name
80          add(list, self.node)
    ENDPROC
    PROC end()                  OF effect IS remove(self.node)
    PROC class()                OF effect IS 'effect'

85  -> String interface
    PROC input2id (str)         OF effect IS ID_INVALID
    PROC output2id(str)         OF effect IS ID_INVALID
    PROC param2id (str)         OF effect IS ID_INVALID
    PROC id2input (id)          OF effect IS NIL
90  PROC id2output(id)          OF effect IS NIL
    PROC id2param (id)          OF effect IS NIL

    -> Linking
    PROC getinput(id)           OF effect IS Throw(ERR_NO_SUCH_INPUTID, id)
95  PROC setoutput(id, link)    OF effect IS Throw(ERR_NO_SUCH_OUTPUTID, id)

    -> Parameters
```

```
PROC set(id, data)          OF effect IS Throw(ERR_NO_SUCH_PARAMID, id)
PROC get(id)                OF effect IS Throw(ERR_NO_SUCH_PARAMID, id)
PROC paramtype(id)          OF effect IS Throw(ERR_NO_SUCH_PARAMID, id)

-> Recalculation
PROC setrecalc()            OF effect;  self.recalcf := TRUE;  ENDPROC
PROC recalc()               OF effect;  self.recalcf := FALSE; ENDPROC

-> Processing control
PROC check()                OF effect; self.setrecalc();  ENDPROC TRUE
PROC isready()              OF effect IS TRUE
PROC issource()             OF effect IS FALSE
PROC clear()                OF effect IS EMPTY
PROC reset()                OF effect IS self.clear()

-> Processing
PROC output(id, data)       OF effect IS Throw(ERR_NO_SUCH_OUTPUTID, id)
PROC process()              OF effect
    IF self.recalcf THEN self.recalc()
ENDPROC

/*-------------------------------------------------------------------------+
| END: effect.e                                                            |
+=========================================================================*/
```

24 Source/exp.e

```
/*=========================================================================+
| exp.e                                                                    |
| Effect class "exp", exponential oscillator, varies between 0 and 1       |
+-------------------------------------------------------------------------*/

OPT MODULE

MODULE '*osc'

EXPORT OBJECT exp OF osc
ENDOBJECT

PROC oscillator(time) OF exp IS ! Fexp(! Flog(2.0) * time) - 1.0

PROC class() OF exp IS 'exp'

/*-------------------------------------------------------------------------+
| END: exp.e                                                               |
+=========================================================================*/
```

25 Source/fbdelay.e

```
/*=========================================================================+
| fbdelay.e                                                                |
| Effect class "fbdelay", delay for use in feedback loops                  |
+-------------------------------------------------------------------------*/

OPT MODULE
```

```
MODULE '*cbuffer', '*defs', '*delay', '*link'

EXPORT OBJECT fbdelay OF delay
ENDOBJECT

PROC class() OF fbdelay IS 'fbdelay'

PROC recalc() OF fbdelay
        DEF in  : PTR TO link
        SUPER self.recalc()             -> does normal delay time, and NEWs cb
        in := self.getinput(ID_MAIN)
        -> reduce delay time by 1 sample
        self.cb.setlength(! self.dt * in.rate - 1.0)
ENDPROC

/*-----------------------------------------------------------------------+
| END: fbdelay.e                                                         |
+=======================================================================*/
```

26 Source/feedback.e

```
/*=======================================================================+
| feedback.e                                                             |
| Effect class "feedback"                                                |
| Allow feedback loops                                                   |
|                                                                        |
| Inherits from in1out1, adding to methods process and clear, and setting|
| issource to TRUE and isready to FALSE                                  |
|                                                                        |
| isready() is FALSE, so that process is not called when input is recieved|
| issource() is TRUE, so process is called by kernel in run              |
| clear() stores data for next sample                                    |
|                                                                        |
| Due to feedback, input cannot reach the feedback object until after its|
| process has been called.                                               |
|                                                                        |
| --object--feedback--\     object needs data from feedback to send input|
|     \---------------/     to feedback, so nothing happens until run calls|
|                           feedback                                     |
|                                                                        |
+-----------------------------------------------------------------------*/

OPT MODULE

MODULE '*defs', '*in1out1'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT feedback OF in1out1
PRIVATE
        last       : LONG     -> previous input
        checking   : LONG     -> prevent loops in check
        processing : LONG     -> prevent loops in process
ENDOBJECT

PROC class()    OF feedback IS 'feedback'
PROC issource() OF feedback IS TRUE
```

```
PROC isready()  OF feedback IS FALSE

/*-------------------------------------------------------------------------*/

PROC new(list , name) OF feedback
        SUPER self.new(list , name)
        self.last := 0.0
        self.checking := FALSE
        self.processing := FALSE
ENDPROC

/*-------------------------------------------------------------------------*/

PROC clear() OF feedback
        self.last := self._main()
        SUPER self.clear()
ENDPROC

/*-------------------------------------------------------------------------*/

PROC reset() OF feedback
        SUPER self.reset()
        self.last := 0.0
ENDPROC

/*-------------------------------------------------------------------------*/

PROC process() OF feedback
        IF self.processing = FALSE  -> prevent loop
                self.processing := TRUE
                SUPER self.process()
                self.output(ID_MAIN, self.last)
                self.processing := FALSE
        ENDIF
ENDPROC

/*-------------------------------------------------------------------------*/

PROC check() OF feedback
        IF self.checking = FALSE    -> prevent loop
                self.checking := TRUE
                IF SUPER self.check() = FALSE THEN RETURN FALSE
                self.checking := FALSE
        ENDIF
ENDPROC TRUE

/*-------------------------------------------------------------------------+
| END: feedback.e                                                          |
+=========================================================================*/
```

27 Source/file.e

```
/*=========================================================================+
| file.e                                                                   |
| Functions for reading / writing / converting integers                    |
|                                                                          |
| Types: ( s | u ) ( byte | word | long ) ( | _x ), where                  |
```

```
    | - s = signed (2's complement), u = unsigned                                |
    | - byte = 8 bits, word = 16 bits, long = 32 bits                            |
    | - (def) = msb first (eg $01234567), _x = lsb first (eg $67452301)          |
    | NB: all longs in E are signed, so ulongs are really slongs                 |
    |                                                                            |
    | read / write return success, others may raise ("file", fh) on errors       |
    |                                                                            |
    +----------------------------------------------------------------------------*/

OPT MODULE

/*----------------------------------------------------------------------------*/

-> Read functions
EXPORT PROC read_sbyte(fh)   IS sbyte(      _read(fh, 1) )
EXPORT PROC read_ubyte(fh)   IS             _read(fh, 1)
EXPORT PROC read_sword(fh)   IS sword(      _read(fh, 2) )
EXPORT PROC read_uword(fh)   IS             _read(fh, 2)
EXPORT PROC read_slong(fh)   IS             _read(fh, 4)
EXPORT PROC read_ulong(fh)   IS             _read(fh, 4)
EXPORT PROC read_sword_x(fh) IS sword(xword(_read(fh, 2)))
EXPORT PROC read_uword_x(fh) IS       xword(_read(fh, 2))
EXPORT PROC read_slong_x(fh) IS       xlong(_read(fh, 4))
EXPORT PROC read_ulong_x(fh) IS       xlong(_read(fh, 4))

/*----------------------------------------------------------------------------*/

-> Write functions
EXPORT PROC write_sbyte(fh, data)   IS _write(fh, 1, data)
EXPORT PROC write_ubyte(fh, data)   IS _write(fh, 1, data)
EXPORT PROC write_sword(fh, data)   IS _write(fh, 2, data)
EXPORT PROC write_uword(fh, data)   IS _write(fh, 2, data)
EXPORT PROC write_slong(fh, data)   IS _write(fh, 4, data)
EXPORT PROC write_ulong(fh, data)   IS _write(fh, 4, data)
EXPORT PROC write_sword_x(fh, data) IS _write(fh, 2, xword(data))
EXPORT PROC write_uword_x(fh, data) IS _write(fh, 2, xword(data))
EXPORT PROC write_slong_x(fh, data) IS _write(fh, 4, xlong(data))
EXPORT PROC write_ulong_x(fh, data) IS _write(fh, 4, xlong(data))

/*----------------------------------------------------------------------------*/

-> Conversion functions
EXPORT PROC sbyte(ubyte) IS IF ubyte >= 128 THEN ubyte - 256 ELSE ubyte
EXPORT PROC ubyte(sbyte) IS sbyte AND $FF
EXPORT PROC sword(uword) IS IF uword >= 32768 THEN uword - 65536 ELSE uword
EXPORT PROC uword(sword) IS sword AND $FFFF
EXPORT PROC xword(word)  IS Shr(word AND $FF00, 8) OR Shl(word AND $00FF, 8)
EXPORT PROC xlong(long)  IS Shr(long AND $FF000000, 24) OR
                            Shr(long AND $00FF0000,  8) OR
                            Shl(long AND $0000FF00,  8) OR
                            Shl(long AND $000000FF, 24)

/*----------------------------------------------------------------------------*/

EXPORT PROC read (fh, buf, len)
ENDPROC Read (fh, buf, len) = len
```

```
EXPORT PROC write(fh, buf, len)
ENDPROC Write(fh, buf, len) = len

/*-----------------------------------------------------------------------*/

PROC _read (f, b); DEF d = 0      -> Read b bytes from f into lsb of d
ENDPROC IF read (f, {d} + 4 - b, b) THEN d    ELSE Throw("file", f)

PROC _write(f, b, d)              -> Write b bytes to f from lsb of d
ENDPROC IF write(f, {d} + 4 - b, b) THEN TRUE ELSE Throw("file", f)

/*-----------------------------------------------------------------------+
| END: file.e                                                            |
+=======================================================================*/
```

28 Source/filter.e

```
/*=======================================================================+
| filter.e                                                               |
| Effect base class "filter"                                             |
| Base class for lowpass, highpass, bandpass, bandreject                 |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*in1out1', '*string', '*value'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT filter OF in1out1
PUBLIC                -> need to be used in derived.recalc
        freq : LONG     -> frequency
        a0   : LONG     -> filter coefficients
        a1   : LONG
        a2   : LONG
        b1   : LONG
        b2   : LONG
PRIVATE
        in1  : LONG     -> previous input / output samples
        in2  : LONG     -> (not cbuffer, as small amount of data)
        out1 : LONG
        out2 : LONG
ENDOBJECT

PROC class() OF filter IS 'filter'

-> No constructor, as freq default set in derived, coefficients set
-> in recalc, previous samples set in reset

/*-----------------------------------------------------------------------*/

PROC reset() OF filter
        SUPER self.reset()
        self.in1  := 0.0
        self.in2  := 0.0
        self.out1 := 0.0
        self.out2 := 0.0
```

```
ENDPROC

/*-------------------------------------------------------------------------*/

PROC process() OF filter
        DEF in, out
        SUPER self.process()
        -> calculate
        in := self._main()
        out := ! (! self.a0 * in) + (! self.a1 * self.in1) +
                 (! self.a2 * self.in2) + (! self.b1 * self.out1) +
                 (! self.b2 * self.out2)
        -> shift to next sample
        self.out2 := self.out1
        self.out1 := out
        self.in2  := self.in1
        self.in1  := in
        -> output
        self.output(ID_MAIN, out)
ENDPROC

/*-------------------------------------------------------------------------*/

PROC param2id(str) OF filter
        IF strcmp(IDS_FREQUENCY, str); RETURN ID_FREQUENCY
        ENDIF
ENDPROC SUPER self.param2id(str)

/*-------------------------------------------------------------------------*/

PROC id2param(id) OF filter
        SELECT id
        CASE ID_FREQUENCY; RETURN IDS_FREQUENCY
        ENDSELECT
ENDPROC SUPER self.id2param(id)

/*-------------------------------------------------------------------------*/

PROC paramtype(id) OF filter
        SELECT id
        CASE ID_FREQUENCY; RETURN TYPE_NUMBER
        ENDSELECT
ENDPROC SUPER self.paramtype(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF filter
        SELECT id
        CASE ID_FREQUENCY
                self.freq := data
                self.setrecalc()
        DEFAULT; SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/
```

```
PROC get(id) OF filter
        SELECT id
100     CASE ID_FREQUENCY; RETURN self.freq
        ENDSELECT
ENDPROC SUPER self.get(id)

/*-------------------------------------------------------------------------+
105 | END: filter.e                                                         |
+=========================================================================*/
```

29 Source/hack.e

```
/*=========================================================================+
| hack.e                                                                   |
| Get an effect from the node contained in it                              |
| __node2effect(node) => effect                                            |
5 +-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS
OPT EXPORT

10 /*-------------------------------------------------------------------------*/

-> WARNING: HACK
-> This function assumes that the node is at offset 4 into the object
-> Offsets are shown by ShowModule for public fields
15 #define __node2effect(node) (node-4)

/*-------------------------------------------------------------------------+
| END: hack.e                                                              |
+=========================================================================*/
```

30 Source/halfrectify.e

```
/*=========================================================================+
| halfrectify.e                                                            |
| Effect class "halfrectify", half rectifies input                         |
+-------------------------------------------------------------------------*/
5

OPT MODULE

MODULE '*defs', '*in1out1'

10 EXPORT OBJECT halfrectify OF in1out1
ENDOBJECT

PROC class() OF halfrectify IS 'halfrectify'

15 PROC process() OF halfrectify
        SUPER self.process()
        self.output(ID_MAIN, IF ! self._main() > 0.0 THEN self._main() ELSE 0.0)
ENDPROC

20 /*-------------------------------------------------------------------------+
| END: halfrectify.e                                                       |
+=========================================================================*/
```

31 Source/highpass.e

```
/*============================================================================+
| highpass.e                                                                  |
| Effect class "highpass", high pass filter                                   |
+----------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*filter', '*link'

EXPORT OBJECT highpass OF filter
ENDOBJECT

PROC new(list, name) OF highpass
        SUPER self.new(list, name)
        self.freq := 2000.0
        self.setrecalc()
        self.reset()
ENDPROC

PROC class() OF highpass IS 'highpass'

PROC recalc() OF highpass
        DEF c, in : PTR TO link
        SUPER self.recalc()
        in := self.getinput(ID_MAIN)
        c := Ftan(! PI * self.freq  / in.rate)
        self.a0 := ! 1.0 / (! 1.0 + (! (! c + Fsqrt(2.0)) * c))
        self.a1 := ! -2.0 * self.a0
        self.a2 := self.a0
        self.b1 := ! self.a1 * (! c * c - 1.0)
        self.b2 := ! -self.a0 * (! 1.0 + (! (! c - Fsqrt(2.0)) * c))
ENDPROC

/*----------------------------------------------------------------------------+
| END: highpass.e                                                             |
+============================================================================*/
```

32 Source/iff8svx_ff.e

```
/*============================================================================+
| iff8svx_ff.e                                                                |
| IFF 8SVX file format definitions                                            |
+----------------------------------------------------------------------------*/

OPT MODULE
OPT EXPORT

/*----------------------------------------------------------------------------*/

OBJECT iff8svx_vhdr
        hioctsamples : LONG
        repeatstart  : LONG
        repeatlength : LONG
        rate         : INT      -> unsigned, but signed in E
```

```
        octaves       : CHAR
        compression   : CHAR
        volume        : LONG      -> 65536 maps to 1.0
ENDOBJECT

/*-----------------------------------------------------------------------*/

CONST MAGIC_FORM = "FORM", MAGIC_8SVX = "8SVX",
      MAGIC_VHDR = "VHDR", MAGIC_BODY = "BODY"

/*-----------------------------------------------------------------------+
| END: iff8svx_ff.e                                                      |
+=======================================================================*/
```

33 Source/in0out1.e

```
/*=======================================================================+
| in0out1.e                                                              |
| Effect base class in0out1                                              |
| Features: output "main", parameter "rate" = output sample rate         |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*effect', '*defs', '*link', '*string', '*value'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT in0out1 OF effect
PUBLIC
        rate : LONG          -> Needed by some derived classes
PRIVATE
        out  : PTR TO link
ENDOBJECT

PROC class()    OF in0out1 IS 'in0out1'
PROC issource() OF in0out1 IS TRUE

/*-----------------------------------------------------------------------*/

PROC new(list, name) OF in0out1
        SUPER self.new(list, name)
        self.out := NIL
        self.rate := DEF_RATE
ENDPROC

/*-----------------------------------------------------------------------*/

PROC end() OF in0out1
        IF self.out
                unlink(self.out)
                self.out := NIL
        ENDIF
ENDPROC SUPER self.end()

/*-----------------------------------------------------------------------*/
```

```
PROC output2id(str) OF in0out1
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.output2id(str)

PROC id2output(id) OF in0out1
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2output(id)

PROC param2id(str) OF in0out1
ENDPROC IF strcmp(IDS_RATE, str) THEN ID_RATE ELSE SUPER self.param2id(str)

PROC id2param(id) OF in0out1
ENDPROC IF id = ID_RATE THEN IDS_RATE ELSE SUPER self.id2param(id)

PROC paramtype(id) OF in0out1
ENDPROC IF id = ID_RATE THEN TYPE_NUMBER ELSE SUPER self.paramtype(id)

PROC get(id) OF in0out1
ENDPROC IF id = ID_RATE THEN self.rate ELSE SUPER self.get(id)

/*-------------------------------------------------------------------------*/

PROC setoutput(id, link) OF in0out1
        SELECT id
        CASE ID_MAIN;    self.out := link
        DEFAULT;         SUPER self.setoutput(id, link)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC output(id, data)    OF in0out1
        SELECT id
        CASE ID_MAIN;    outputlink(self.out, data)
        DEFAULT;         SUPER self.output(id, data)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF in0out1
        SELECT id
        CASE ID_RATE
                IF ! data <= 0.0 THEN Throw(ERR_BAD_RANGE, id)
                self.rate := data
        DEFAULT;        SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC check() OF in0out1
        IF SUPER self.check() = FALSE THEN RETURN FALSE
        IF self.out = NIL
                Throw(ERR_CHECK, [ CHECK_OUTPUT_NOT_CONNECTED, self, ID_MAIN ])
        ENDIF
        self.out.rate := self.rate
ENDPROC self.out.to.check()
```

```
/*-------------------------------------------------------------------------+
| END: in0out1.e                                                           |
+=========================================================================*/
```

34 Source/in1out0.e

```
/*=========================================================================+
| in1out0.e                                                                |
| Effect base class in1out0                                                |
| Defines an input "main"                                                  |
| in1out0._main()       get input sample (for efficiency in process)       |
+-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*effect', '*defs', '*link', '*string'

/*-------------------------------------------------------------------------*/

EXPORT OBJECT in1out0 OF effect
PRIVATE
        in   : link
ENDOBJECT

PROC class() OF in1out0 IS 'in1out0'
PROC _main() OF in1out0 IS self.in.data

/*-------------------------------------------------------------------------*/

PROC new(list, name) OF in1out0
        SUPER self.new(list, name)
        newlink(self.in, self, ID_MAIN)
ENDPROC

/*-------------------------------------------------------------------------*/

PROC end() OF in1out0
        unlink(self.in)
ENDPROC SUPER self.end()

/*-------------------------------------------------------------------------*/

PROC input2id (str) OF in1out0
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.input2id(str)

PROC id2input (id) OF in1out0
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2input(id)

PROC getinput(id) OF in1out0
ENDPROC IF id = ID_MAIN THEN self.in ELSE SUPER self.getinput(id)

PROC clear() OF in1out0 IS clearlink(self.in) BUT SUPER self.clear()

PROC isready() OF in1out0
ENDPROC IF self.in.ready THEN SUPER self.isready() ELSE FALSE

/*-------------------------------------------------------------------------*/
```

```
PROC check() OF in1out0
        IF SUPER self.check() = FALSE THEN RETURN FALSE
        IF self.in.from = NIL
                Throw(ERR_CHECK, [ CHECK_INPUT_NOT_CONNECTED, self, ID_MAIN ])
        ENDIF
ENDPROC TRUE

/*-------------------------------------------------------------------------+
| END: in1out0.e                                                           |
+=========================================================================*/
```

35 Source/in1out1.e

```
/*=========================================================================+
| in1out1.e                                                                |
| Effect base class in1out1                                                |
| Defines an input "main" and an output "main"                             |
| in1out1._main()       get input sample (for efficiency in process)       |
+-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*effect', '*defs', '*link', '*string', '*debug'

/*-------------------------------------------------------------------------*/

EXPORT OBJECT in1out1 OF effect
PRIVATE
        in  : link
        out : PTR TO link
ENDOBJECT

PROC class() OF in1out1 IS 'in1out1'
PROC _main() OF in1out1 IS self.in.data

/*-------------------------------------------------------------------------*/

PROC new(list, name) OF in1out1
        SUPER self.new(list, name)
        newlink(self.in, self, ID_MAIN)
        self.out := NIL
ENDPROC

/*-------------------------------------------------------------------------*/

PROC end() OF in1out1
        unlink(self.in)
        unlink(self.out)
        self.out := NIL
ENDPROC SUPER self.end()

/*-------------------------------------------------------------------------*/

PROC input2id (str)     OF in1out1
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.input2id(str)
```

```
PROC output2id(str)      OF in1out1
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.output2id(str)

PROC id2input (id)       OF in1out1
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2input(id)

PROC id2output(id)       OF in1out1
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2output(id)

PROC getinput(id)        OF in1out1
ENDPROC IF id = ID_MAIN THEN self.in ELSE SUPER self.getinput(id)

PROC isready()           OF in1out1
ENDPROC IF self.in.ready THEN SUPER self.isready() ELSE FALSE

/*-------------------------------------------------------------------------*/

PROC setoutput(id, link)    OF in1out1
        SELECT id
        CASE ID_MAIN;    self.out := link
        DEFAULT;         SUPER self.setoutput(id, link)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC clear()             OF in1out1
        clearlink(self.in)
ENDPROC SUPER self.clear()

/*-------------------------------------------------------------------------*/

PROC output(id, data)    OF in1out1
        SELECT id
        CASE ID_MAIN;    outputlink(self.out, data)
        DEFAULT;         SUPER self.output(id, data)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC check() OF in1out1
        IF SUPER self.check() = FALSE THEN RETURN FALSE
        IF self.in.from = NIL
                Throw(ERR_CHECK, [ CHECK_INPUT_NOT_CONNECTED, self, ID_MAIN ])
        ENDIF
        IF self.out = NIL
                Throw(ERR_CHECK, [ CHECK_OUTPUT_NOT_CONNECTED, self, ID_MAIN ])
        ENDIF
        self.out.rate := self.in.rate
ENDPROC self.out.to.check()

/*-------------------------------------------------------------------------+
| END: in1out1.e                                                           |
+=========================================================================*/
```

36 Source/in1outm.e

```
/*=============================================================================+
| in1outm.e                                                                    |
| Effect base class in1outm                                                    |
|                                                                              |
| Multiple outputs, single input.  The number of outputs can be set once,      |
| after which attempts to change it fail.                                     |
|                                                                              |
| Defines an input "main" and outputs "outX", X = 1, 2, ... "outputs" param    |
|                                                                              |
| in1outm._outputs()      get number of inputs (for efficiency in process)     |
| in1outm._out(x)         get output id                                        |
| in1outm._in(x)          get input sample (for efficiency in process)         |
|                                                                              |
+-----------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*effect', '*defs', '*link', '*string', '*value', '*debug'

RAISE "MEM" IF New() = NIL

/*-----------------------------------------------------------------------------*/

EXPORT OBJECT in1outm OF effect
PRIVATE
        outs : LONG
        out  : PTR TO LONG   -> ptr to ptr to link
        in   : link
ENDOBJECT

PROC class()     OF in1outm IS 'in1outm'
PROC _outputs() OF in1outm IS self.outs
PROC _out(x)     OF in1outm IS ID_MULTI_OUT + x
PROC _in()       OF in1outm IS self.in.data

/*-----------------------------------------------------------------------------*/

PROC new(list, name) OF in1outm
        SUPER self.new(list, name)
        self.outs := 0
        self.out  := NIL
        newlink(self.in, self, ID_MAIN)
ENDPROC

/*-----------------------------------------------------------------------------*/

PROC end() OF in1outm
        DEF i
        IF self.out
                FOR i := 0 TO self.outs - 1
                        IF self.out[i] THEN unlink(self.out[i])
                ENDFOR
                Dispose(self.out)
                self.out := NIL
        ENDIF
```

```
        unlink(self.in)
ENDPROC SUPER self.end()

/*-------------------------------------------------------------------------*/

PROC input2id (str)     OF in1outm
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.input2id(str)

PROC id2input (id)      OF in1outm
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2input(id)

PROC getinput(id)       OF in1outm
ENDPROC IF id = ID_MAIN THEN self.in ELSE SUPER self.getinput(id)

PROC get(id) OF in1outm
ENDPROC IF id = ID_OUTPUTS THEN self.outs ELSE SUPER self.get(id)

PROC isready()          OF in1outm
ENDPROC IF self.in.ready THEN SUPER self.isready() ELSE FALSE

/*-------------------------------------------------------------------------*/

PROC output2id(str) OF in1outm
        DEF id, len
        IF strncmp(IDS_OUT, str, 3)
                id, len := Val(str + 3)
                IF ((len + 3) = StrLen(str)) AND (0 < id) AND (id <= self.outs)
                        RETURN ID_MULTI_OUT + id
                ENDIF
        ENDIF
ENDPROC SUPER self.output2id(str)

/*-------------------------------------------------------------------------*/

PROC param2id(str) OF in1outm
        IF strcmp(IDS_OUTPUTS, str); RETURN ID_OUTPUTS
        ENDIF
ENDPROC SUPER self.param2id(str)

/*-------------------------------------------------------------------------*/

PROC id2output(id)      OF in1outm
        DEF outid
        outid := id - ID_MULTI_OUT
        IF (0 < outid) AND (outid <= self.outs)
                RETURN StringF(String(8), 'out\d', outid)
        ENDIF
ENDPROC SUPER self.id2output(id)

/*-------------------------------------------------------------------------*/

PROC id2param(id) OF in1outm
        SELECT id
        CASE ID_OUTPUTS; RETURN IDS_OUTPUTS
        ENDSELECT
ENDPROC SUPER self.id2param(id)
```

```
/*-----------------------------------------------------------------------*/

PROC paramtype(id) OF in1outm
        SELECT id
        CASE ID_OUTPUTS; RETURN TYPE_NUMBER
        ENDSELECT
ENDPROC SUPER self.paramtype(id)

/*-----------------------------------------------------------------------*/

PROC set(id, data) OF in1outm
        DEF i
        SELECT id
        CASE ID_OUTPUTS
                IF ! data ! <= 0 THEN Throw(ERR_BAD_RANGE, id)
                -> Cannot change number of outputs after first set (requires ↙
                    ↳ either
                -> relinking of all existing links or keeping all existing links↙
                    ↳  and
                -> adding)
                IF self.out THEN RETURN FALSE
                self.out := New(! data ! * SIZEOF LONG)
                self.outs := ! data !
                -> Clear links
                FOR i := 0 TO self.outs - 1
                        self.out[i] := NIL
                ENDFOR
                RETURN TRUE
        ENDSELECT
ENDPROC SUPER self.set(id, data)

/*-----------------------------------------------------------------------*/

PROC setoutput(id, link)    OF in1outm
        DEF outid
        outid := id - ID_MULTI_OUT
        IF (0 < outid) AND (outid <= self.outs)
                self.out[outid - 1] := link
                RETURN TRUE
        ENDIF
ENDPROC SUPER self.setoutput(id, link)

/*-----------------------------------------------------------------------*/

PROC clear() OF in1outm
        clearlink(self.in)
ENDPROC SUPER self.clear()

/*-----------------------------------------------------------------------*/

PROC output(id, data)   OF in1outm
        DEF outid
        outid := id - ID_MULTI_OUT
        IF (0 < outid) AND (outid <= self.outs)
                RETURN outputlink(self.out[outid - 1], data)
        ENDIF
ENDPROC SUPER self.output(id, data)
```

```
/*-------------------------------------------------------------------------*/

PROC check() OF in1outm
        DEF i, ok, l : PTR TO link
        IF SUPER self.check() = FALSE THEN RETURN FALSE
        IF self.in.from = NIL
                Throw(ERR_CHECK, [ CHECK_INPUT_NOT_CONNECTED, self, ID_MAIN ])
        ENDIF
        IF self.out = NIL
                Throw(ERR_CHECK, [ CHECK_INTERNAL_ERROR, self, 'no outputs' ])
        ENDIF
        FOR i := 0 TO self.outs - 1
                l := self.out[i]
                IF l = NIL
                        Throw(ERR_CHECK,
                                [ CHECK_OUTPUT_NOT_CONNECTED, self, i + 1 + ↙
                                    ↳ ID_MULTI_OUT])
                ENDIF
                l.rate := self.in.rate
        ENDFOR
        ok := TRUE
        FOR i := 0 TO self.outs - 1
                l := self.out[i]
                ok := ok AND l.to.check()
        ENDFOR
ENDPROC ok

/*-------------------------------------------------------------------------+
| END: in1outm.e                                                           |
+=========================================================================*/
```

37 Source/inmout1.e

```
/*=========================================================================+
| inmout1.e                                                                |
| Effect base class inmout1                                                |
|                                                                          |
| Multiple inputs, single output.  The number of inputs can be set once,   |
| after which attempts to change it fail.                                  |
|                                                                          |
| Defines an output "main" and inputs "inX", X = 1, 2, ... "inputs" param  |
|                                                                          |
| inmout1._inputs()     get number of inputs (for efficiency in process)   |
| inmout1._in(x)        get input sample (for efficiency in process)       |
|                                                                          |
+-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*effect', '*defs', '*link', '*string', '*value', '*debug'

RAISE "MEM" IF New() = NIL

/*-------------------------------------------------------------------------*/

EXPORT OBJECT inmout1 OF effect
```

```
PRIVATE
        ins : LONG
        in : PTR TO link
        out : PTR TO link
ENDOBJECT

PROC class()    OF inmout1 IS 'inmout1'
PROC _inputs() OF inmout1 IS self.ins
PROC _in(x)     OF inmout1 IS self.in[x - 1].data

/*------------------------------------------------------------------------*/

PROC new(list, name) OF inmout1
        SUPER self.new(list, name)
        self.ins := 0
        self.in  := NIL
        self.out := NIL
ENDPROC

/*------------------------------------------------------------------------*/

PROC end() OF inmout1
        DEF i
        IF self.in
                FOR i := 0 TO self.ins - 1
                        IF self.in[i] THEN unlink(self.in[i])
                ENDFOR
                Dispose(self.in)
                self.in := NIL
        ENDIF
        IF self.out
                unlink(self.out)
                self.out := NIL
        ENDIF
ENDPROC SUPER self.end()

/*------------------------------------------------------------------------*/

PROC input2id (str) OF inmout1
        DEF id, len
        IF strncmp(IDS_IN, str, 2)
                id, len := Val(str + 2)
                IF ((len + 2) = StrLen(str)) AND (0 < id) AND (id <= self.ins)
                        RETURN ID_MULTI_IN + id
                ENDIF
        ENDIF
ENDPROC SUPER self.input2id(str)

/*------------------------------------------------------------------------*/

PROC param2id(str) OF inmout1
        IF strcmp(IDS_INPUTS, str) THEN RETURN ID_INPUTS
ENDPROC SUPER self.param2id(str)

/*------------------------------------------------------------------------*/

PROC id2input (id) OF inmout1
```

```
        DEF inid
        inid := id - ID_MULTI_IN
        IF (0 < inid) AND (inid <= self.ins)
                RETURN StringF(String(8), 'in\d', inid)
        ENDIF
ENDPROC SUPER self.id2input(id)

/*-----------------------------------------------------------------------*/

PROC output2id(str)     OF inmout1
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.output2id(str)

PROC id2output(id)      OF inmout1
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2output(id)

PROC id2param(id) OF inmout1
ENDPROC IF id = ID_INPUTS THEN IDS_INPUTS ELSE SUPER self.id2param(id)

PROC paramtype(id) OF inmout1
ENDPROC IF id = ID_INPUTS THEN TYPE_NUMBER ELSE SUPER self.paramtype(id)

PROC get(id) OF inmout1
ENDPROC IF id = ID_INPUTS THEN self.ins ELSE SUPER self.get(id)

/*-----------------------------------------------------------------------*/

PROC set(id, data) OF inmout1
        DEF i
        SELECT id
        CASE ID_INPUTS
                IF ! data ! <= 0 THEN Throw(ERR_BAD_RANGE, id)
                -> Cannot change number of inputs after first set (requires ↙
                    ↳ either
                -> relinking of all existing links (which requires access to ↙
                    ↳ kernel)
                -> or keeping all existing links and adding (which is tricky))
                IF self.in THEN RETURN FALSE
                self.in := New(! data ! * SIZEOF link)
                self.ins := ! data !
                -> Clear links
                FOR i := 0 TO self.ins - 1
                        newlink(self.in[i], self, ID_MULTI_IN + i + 1)
                ENDFOR
                RETURN TRUE
        ENDSELECT
ENDPROC SUPER self.set(id, data)

/*-----------------------------------------------------------------------*/

PROC getinput(id)       OF inmout1
        DEF inid
        inid := id - ID_MULTI_IN
        IF (0 < inid) AND (inid <= self.ins)
                RETURN self.in[inid - 1]
        ENDIF
ENDPROC SUPER self.getinput(id)
```

```
/*-------------------------------------------------------------------------*/

PROC setoutput(id, link)    OF inmout1
        SELECT id
140     CASE ID_MAIN;    self.out := link
        DEFAULT;         SUPER self.setoutput(id, link)
        ENDSELECT
ENDPROC

145 /*-------------------------------------------------------------------------*/

PROC clear() OF inmout1
        DEF i
        FOR i := 0 TO self.ins - 1
150             clearlink(self.in[i])
        ENDFOR
ENDPROC SUPER self.clear()

/*-------------------------------------------------------------------------*/
155
PROC isready()          OF inmout1
        DEF ready = TRUE, i
        FOR i := 0 TO self.ins - 1
        EXIT ready = FALSE
160             ready := ready AND self.in[i].ready
        ENDFOR
        IF ready
                RETURN SUPER self.isready()
        ENDIF
165 ENDPROC FALSE

/*-------------------------------------------------------------------------*/

PROC output(id, data)   OF inmout1
170     SELECT id
        CASE ID_MAIN;    outputlink(self.out, data)
        DEFAULT;         SUPER self.output(id, data)
        ENDSELECT
ENDPROC
175
/*-------------------------------------------------------------------------*/

PROC check() OF inmout1
        DEF i
180     IF SUPER self.check() = FALSE THEN RETURN FALSE
        IF self.out = NIL
                Throw(ERR_CHECK, [ CHECK_OUTPUT_NOT_CONNECTED, self, ID_MAIN ])
        ENDIF
        self.out.rate := self.in[0].rate
185     IF self.out.to.check() = FALSE THEN RETURN FALSE
        IF self.in = NIL
                Throw(ERR_CHECK, [ CHECK_INTERNAL_ERROR, self, 'no inputs' ])
        ENDIF
        FOR i := 0 TO self.ins - 1
190             IF self.in[i].from = NIL
                        Throw(ERR_CHECK,
                              [ CHECK_INPUT_NOT_CONNECTED, self, i + 1 + ↙
```

```
                                        ↳ ID_MULTI_IN ])
                ENDIF
        ENDFOR
195     FOR i := 0 TO self.ins - 1
                IF ! self.in[i].rate <> self.in[0].rate
debug(PrintF(DEBUG'inmout1.check bad rate: \s <> \s\n',realf(self.in[i].rate), ↙
    ↳ realf(self.in[0].rate)))
/*                      Throw(ERR_CHECK,
                                [ CHECK_BAD_INPUT_RATE, self, i + 1 + ID_MULTI_IN ↙
                                    ↳ ])
200 */              ENDIF
        ENDFOR
ENDPROC TRUE

/*-------------------------------------------------------------------------+
205 | END: inmout1.e                                                        |
+=========================================================================*/
```

38 Source/invert.e

```
/*=========================================================================+
| invert.e                                                                 |
| Effect class "invert", inverts input                                     |
+-------------------------------------------------------------------------*/
5

OPT MODULE

MODULE '*defs', '*in1out1'

10  EXPORT OBJECT invert OF in1out1
    ENDOBJECT

    PROC class() OF invert IS 'invert'

15  PROC process() OF invert
        SUPER self.process()
        self.output(ID_MAIN, ! - self._main())
    ENDPROC

20  /*-------------------------------------------------------------------------+
    | END: invert.e                                                          |
    +=========================================================================*/
```

39 Source/kernel.e

```
/*=========================================================================+
| kernel.e                                                                 |
| System kernel                                                            |
|                                                                          |
5   | All arguments are strings unless otherwise specified.  Errors are        |
    | reported by raising exceptions (see defs.e).                             |
    |                                                                          |
    | kernel.new()                  constructor                                |
    | kernel.end()                  destructor                                 |
10  | kernel.new_(class, name)      create a new effect object                 |
    | kernel.delete(name)           delete a new effect object                 |
```

```
    | kernel.link(source, output, dest, input)                                  |
    |                              link effect objects                          |
    | kernel.set_(name, param, value)                                           |
15  |                              set a parameter, value is a value structure  |
    | kernel.paramtype(name, param)                                             |
    |                              get the type of a parameter                  |
    | kernel.run()                 perform processing                           |
    |                                                                           |
20  | kernel.runonce()             perform processing on one sample             |
    | #?new(list, name)            classnode functions                          |
    |                                                                           |
    +---------------------------------------------------------------------------*/

25  OPT MODULE, PREPROCESS

    /*---------------------------------------------------------------------------*/

    MODULE '*classnode', '*defs', '*effect', '*hack', '*link', '*link_', '*list',
30         '*string', '*value', '*debug',
           '*add', '*amp', '*bandpass', '*bandreject', '*constant', '*copy',
           '*delay', '*echo', '*fbdelay', '*feedback', '*halfrectify',
           '*highpass', '*invert', '*lowpass', '*mul', '*notch', '*print',
           '*pulse', '*ramp', '*read8svx', '*readslab', '*rectify', '*sine',
35         '*split', '*toparam', '*triangle', '*vox', '*whitenoise',
           '*write8svx', '*writeslab', '*zfilter',
           '*exp', '*scale'

    /*---------------------------------------------------------------------------*/
40
    EXPORT OBJECT kernel
    PUBLIC      -> CLI likes to list objects and classes
           classes : list
           objects : list
45  PRIVATE
           rate    : LONG      -> float sample rate to use for global calcs
           runtime : LONG      -> float time to run for
           runsmps : LONG      -> integer number of samples to run for
    ENDOBJECT
50
    /*---------------------------------------------------------------------------*/

    -> Reduce verbosity of kernel.new
    #define __addclass(name,func)\
55  add(self.classes, NEW [ NIL, NIL, 'name', {func} ] : classnode)

    #define _addclass(name)\
    add(self.classes, NEW [ NIL, NIL, 'name', {name} ] : classnode)

60  PROC new() OF kernel
           -> Initialise classes
           newlist(self.classes)
           __addclass(add, add_);
           _addclass(exp);          _addclass(scale);    ->_addclass(bandreject);
65         _addclass(amp);          _addclass(bandpass);    _addclass(bandreject);
           _addclass(constant);     _addclass(copy);        _addclass(delay);
           _addclass(echo);         _addclass(fbdelay);     _addclass(feedback);
           _addclass(halfrectify); _addclass(highpass);     _addclass(invert);
```

```
        _addclass(lowpass);     _addclass(mul);         _addclass(notch);
        _addclass(print);       _addclass(pulse);       _addclass(ramp);
        _addclass(read8svx);    _addclass(readslab);    _addclass(rectify);
        _addclass(sine);        _addclass(split);       _addclass(toparam);
        _addclass(triangle);    _addclass(vox);         _addclass(whitenoise);
        _addclass(write8svx);   _addclass(writeslab);   _addclass(zfilter);
        -> Initialise object list
        newlist(self.objects)
        -> Initialise parameters
        self.rate := DEF_RATE
        self.runtime := 0.5
ENDPROC

/*-------------------------------------------------------------------------*/

PROC end() OF kernel
        DEF s : PTR TO effect, n : PTR TO node, c : PTR TO classnode, t
        -> Free objects
        n := self.objects.head.next
        WHILE (t := n.next) <> NIL
                s := __node2effect(n)
                END s
                n := t
        ENDWHILE
        -> Free classes
        c := self.classes.head.next
        WHILE (t := c.next) <> NIL
                END c
                c := t
        ENDWHILE
ENDPROC

/*-------------------------------------------------------------------------*/

-> Note "_", new() is used for constructors
PROC new_(class, name) OF kernel
        DEF c : PTR TO classnode, f
        IF (c := find(self.classes, class)) = NIL
                Throw(ERR_NO_SUCH_CLASS, class)
        ENDIF
        IF find(self.objects, name) THEN Throw(ERR_NAME_ALREADY_USED, name)
        f := c.newf
        -> compiler warning, variable used as function
        f(self.objects, name, NIL)      -> last variable is dummy for "_func"
ENDPROC

/*-------------------------------------------------------------------------*/

PROC delete(name) OF kernel
        DEF s   : PTR TO effect
        IF (s := __node2effect(find(self.objects, name))) = __node2effect(NIL)
                Throw(ERR_NO_SUCH_NAME, name)
        ENDIF
        END s
ENDPROC

/*-------------------------------------------------------------------------*/
```

```
PROC link(source, output, dest, input) OF kernel
        DEF s : PTR TO effect, sid, d : PTR TO effect, did
        IF (s := __node2effect(find(self.objects, source))) = __node2effect(NIL)
                Throw(ERR_NO_SUCH_NAME,    source)
        ENDIF
        IF (sid := s.output2id(output)) = ID_INVALID
                Throw(ERR_NO_SUCH_OUTPUT, output)
        ENDIF
        IF (d := __node2effect(find(self.objects, dest))) = __node2effect(NIL)
                Throw(ERR_NO_SUCH_NAME, dest)
        ENDIF
        IF (did := d.input2id(input)) = ID_INVALID
                Throw(ERR_NO_SUCH_INPUT, input)
        ENDIF
        link_(s, sid, d, did)       -> workaround for bug in EC
ENDPROC

/*-----------------------------------------------------------------------*/

-> Note "_", set() is to set parameters (for consistency with effect classes)
PROC set_(name, param, value : PTR TO value) OF kernel
        DEF s : PTR TO effect, id, obj
        IF (s := __node2effect(find(self.objects, name))) = __node2effect(NIL)
                Throw(ERR_NO_SUCH_NAME,    name)
        ENDIF
        IF (id := s.param2id(param)) = ID_INVALID
                Throw(ERR_NO_SUCH_PARAM, param)
        ENDIF
        IF value.type <> s.paramtype(id) THEN Throw(ERR_BAD_PARAM_TYPE, value)
        IF value.type = TYPE_OBJPART    -> get object pointer from string
                obj := find(self.objects, value.data::value_objpart.obj)
                IF obj = NIL
                        Throw(ERR_NO_SUCH_NAME, value.data::value_objpart.obj)
                ENDIF
                value.data::value_objpart.obj := __node2effect(obj)
        ENDIF
        s.set(id, value.data)
ENDPROC

/*-----------------------------------------------------------------------*/

PROC paramtype(name, param) OF kernel
        DEF s : PTR TO effect, id
        IF (s := __node2effect(find(self.objects, name))) = __node2effect(NIL)
                Throw(ERR_NO_SUCH_NAME, name)
        ENDIF
        IF (id := s.param2id(param)) = ID_INVALID
                Throw(ERR_NO_SUCH_PARAM, param)
        ENDIF
ENDPROC s.paramtype(id)

/*-----------------------------------------------------------------------*/

PROC reset() OF kernel
        DEF n : PTR TO node, s : PTR TO effect
        n := self.objects.head.next
```

```
        WHILE n.next <> NIL
                s := __node2effect(n)
                s.reset()
                n := n.next
        ENDWHILE
ENDPROC

/*-----------------------------------------------------------------------*/

PROC run() OF kernel
        DEF n : PTR TO node, s : PTR TO effect, t
        -> Move any toparam objects to end of list
        n := self.objects.tail.prev
        WHILE (t := n.prev) <> NIL
                s := __node2effect(n)
                IF strcmp('toparam', s.class())
                        remove(n)
                        add(self.objects, n)
                ENDIF
                n := t
        ENDWHILE
        -> Check sources
        n := self.objects.head.next
        WHILE n.next <> NIL
                s := __node2effect(n)
                IF s.issource() THEN s.check()
                n := n.next
        ENDWHILE
/************************************************************************/
        -> How many samples to run for
->      self.rate := 44100 ->maxsamplerate(self.objects.head.next)
->      self.runsmps := ! self.runtime * self.rate !
        self.runsmps := 22050
/************************************************************************/
        -> Run
        FOR t := 1 TO self.runsmps DO self.runonce()
ENDPROC

/*-----------------------------------------------------------------------*/

PROC runonce() OF kernel
        DEF n : PTR TO node, s : PTR TO effect
        -> call process of sources
        n := self.objects.head.next
        WHILE n.next <> NIL
                s := __node2effect(n)
                IF s.issource() THEN s.process()
                n := n.next
        ENDWHILE
        -> clear all
        n := self.objects.head.next
        WHILE n.next <> NIL
                s := __node2effect(n)
                s.clear()
                n := n.next
        ENDWHILE
ENDPROC
```

```
240
    /*-----------------------------------------------------------------------*/
    /*************************************************************************/

    PROC maxsamplerate(node : PTR TO node)
245         DEF s : PTR TO effect, rate = 0.0, newrate
            WHILE node.next <> NIL
                    s := __node2effect(node)
                    IF s.issource()
                            newrate := s.get(ID_RATE)
250                         IF ! newrate > rate THEN rate := newrate
                    ENDIF
                    node := node.next
            ENDWHILE
    ENDPROC
255
    /*************************************************************************/
    /*-----------------------------------------------------------------------*/

    -> classnode functions, generic macro (macros in E are limited to one line
260 -> without ";", so a dummy argument is used instead of DEF)
    #define _func(type)\
    PROC type(list, name, o = NIL : PTR TO type) IS NEW o.new(list, name)

    -> already a function add() in list.e, so use add_()
265 PROC add_(list, name, o = NIL : PTR TO add) IS NEW o.new(list, name)

    _func(exp);         _func(scale);    ->_func(bandreject);  _func(constant);
    _func(amp);         _func(bandpass);    _func(bandreject);  _func(constant);
    _func(copy);        _func(delay);       _func(echo);        _func(fbdelay);
270 _func(feedback);    _func(halfrectify); _func(highpass);    _func(invert);
    _func(lowpass);     _func(mul);         _func(notch);       _func(print);
    _func(pulse);       _func(ramp);        _func(read8svx);    _func(readslab);
    _func(rectify);     _func(sine);        _func(split);       _func(toparam);
    _func(triangle);    _func(vox);         _func(whitenoise);  _func(write8svx);
275 _func(writeslab);   _func(zfilter);

    /*-----------------------------------------------------------------------+
    | END: kernel.e                                                         |
    +=======================================================================*/
```

40 Source/link.e

```
    /*=======================================================================+
    | link.e                                                                |
    | Effect interconection structure and functions                         |
    |                                                                       |
  5 | For examples of usage, see in1out1.e                                  |
    |                                                                       |
    | newlink(link, effect, input)      initialise link, giving destination |
    | link(source, output, dest, input) link effects                        |
    | clearlink(link)                   reset data and ready before next sample |
 10 | outputlink(link, data)            calls input of the destination      |
    | unlink(link)                      unlinks the input from both ends    |
    |                                                                       |
    +-----------------------------------------------------------------------*/
```

```
OPT MODULE, PREPROCESS
OPT EXPORT

MODULE '*effect', '*defs', '*debug'

/*-----------------------------------------------------------------------*/

OBJECT link
PUBLIC
        to      : PTR TO effect     -> owner of input
        tid     : LONG              -> input id
        from    : PTR TO effect     -> source of output
        fid     : LONG              -> output id
        data    : LONG              -> sample data
        ready   : LONG              -> input set this time
        rate    : LONG              -> link sample rate
ENDOBJECT

/*-----------------------------------------------------------------------*/

PROC newlink(link : PTR TO link, effect : PTR TO effect, input)
        assert(link, 'link.newlink')
        link.to   := effect
        link.tid  := input
        link.rate := DEF_RATE
        link.from := NIL
        link.fid  := ID_INVALID
        clearlink(link)
ENDPROC

/*-----------------------------------------------------------------------*/

PROC link(source : PTR TO effect, output, dest : PTR TO effect, input)
        DEF link = NIL : PTR TO link
        -> allow using link() to set input / output to nothing
        IF dest   THEN link := dest.getinput(input)     -> exception if invalid
        IF source THEN source.setoutput(output, link)   -> exception if invalid
        IF link                                         -> valid, change data
                link.from := source
                link.fid := output
        ENDIF
ENDPROC

/*-----------------------------------------------------------------------*/

PROC clearlink(link : PTR TO link)
        assert(link, 'link.clearlink')
        link.data  := 0.0
        link.ready := FALSE
ENDPROC

/*-----------------------------------------------------------------------*/

PROC outputlink(link : PTR TO link, data)
        assert(link,     'link.outputlink.link')
        assert(link.to, 'link.outputlink.to')
        link.data := data
```

```
        link.ready := TRUE
        IF link.to.isready() THEN link.to.process()
ENDPROC

/*-----------------------------------------------------------------------*/

PROC unlink(link : PTR TO link)
        IF link
                IF link.from <> NIL
                        link.from.setoutput(link.fid, NIL)
                        link.from := NIL
                        link.fid := ID_INVALID
                ENDIF
                clearlink(link)
        ENDIF
ENDPROC

/*-----------------------------------------------------------------------+
| END: link.e                                                            |
+=======================================================================*/
```

41 Source/link_.e

```
/*=======================================================================+
| link_.e                                                                |
| Workaround for bug in EC                                               |
+-----------------------------------------------------------------------*/

OPT MODULE

MODULE '*link'

EXPORT PROC link_(a,b,c,d) IS link(a,b,c,d)

/*-----------------------------------------------------------------------+
| END: link_.e                                                           |
+=======================================================================*/
```

42 Source/list.e

```
/*=======================================================================+
| list.e                                                                 |
| Low level list type, with named nodes (case insensitive, see string.e) |
|                                                                        |
| At each end the list is terminated with a node that is part of the list |
| structure.  See find for an example of traversing a list.              |
|                                                                        |
| newlist(list)          prepare a list for use                          |
| add(list, node)        add a node to (the tail of) the list            |
| remove(node)           remove a node from a list (safe to remove twice) |
| find(list, name)       find a named node, returns the node or NIL      |
|                                                                        |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS
OPT EXPORT
```

```
MODULE '*string', '*debug'

/*-----------------------------------------------------------------------*/

OBJECT node
        next    : PTR TO node
        prev    : PTR TO node
        name    : PTR TO CHAR
ENDOBJECT

/*-----------------------------------------------------------------------*/

OBJECT list
        head    : node
        tail    : node
ENDOBJECT

/*-----------------------------------------------------------------------*/

PROC newlist(list : PTR TO list)
        assert(list, 'list.newlist.list')
        list.head.next := list.tail
        list.head.prev := NIL
        list.tail.next := NIL
        list.tail.prev := list.head
ENDPROC

/*-----------------------------------------------------------------------*/

PROC add(list : PTR TO list, node : PTR TO node)
        assert(list, 'list.add.list')
        assert(node, 'list.add.node')
        node.next := list.tail
        node.prev := list.tail.prev
        node.prev.next := node
        node.next.prev := node
ENDPROC

/*-----------------------------------------------------------------------*/

PROC remove(node : PTR TO node)
        assert(node, 'list.remove.node')
        IF (node.prev <> NIL) AND (node.next <> NIL)
                node.prev.next := node.next
                node.next.prev := node.prev
                node.next := NIL    -> make safe to remove() twice
                node.prev := NIL
        ENDIF
ENDPROC

/*-----------------------------------------------------------------------*/

PROC find(list : PTR TO list, name)
        DEF n : PTR TO node
        assert(list, 'list.find.list')
        n := list.head.next     -> first "real node
```

```
        WHILE n.next <> NIL      -> "fake" last node has next = NIL
                IF strcmp(n.name, name) THEN RETURN n
                n := n.next
        ENDWHILE
ENDPROC NIL

/*-----------------------------------------------------------------------+
| END: list.e                                                            |
+=======================================================================*/
```

43 Source/lowpass.e

```
/*=======================================================================+
| lowpass.e                                                              |
| Effect class "lowpass", low pass filter                                |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*filter', '*link'

EXPORT OBJECT lowpass OF filter
ENDOBJECT

PROC new(list, name) OF lowpass
        SUPER self.new(list, name)
        self.freq := 250.0
        self.setrecalc()
        self.reset()
ENDPROC

PROC class() OF lowpass IS 'lowpass'

PROC recalc() OF lowpass
        DEF c, in : PTR TO link
        SUPER self.recalc()
        in := self.getinput(ID_MAIN)
        c := ! 1.0 / Ftan(! PI * self.freq / in.rate)
        self.a0 := ! 1.0 / (! 1.0 + (! (! c + Fsqrt(2.0)) * c))
        self.a1 := ! 2.0 * self.a0
        self.a2 := self.a0
        self.b1 := ! self.a1 * (! c * c - 1.0)
        self.b2 := ! -self.a0 * (! 1.0 + (! (! c - Fsqrt(2.0)) * c))
ENDPROC

/*-----------------------------------------------------------------------+
| END: lowpass.e                                                         |
+=======================================================================*/
```

44 Source/main.e

```
/*=======================================================================+
| main.e                                                                 |
| System entry point                                                     |
|                                                                        |
| E automatically opens exec.library, dos.library, mathieeesingbas.library |
```

```
    | and mathieeesingtrans.library                                                |
    |                                                                             |
    +-----------------------------------------------------------------------------*/

10  OPT PREPROCESS

    /*-----------------------------------------------------------------------------*/

    MODULE '*cli', '*rnd', '*string'
15
    /*-----------------------------------------------------------------------------*/

    -> workaround for not being able to use EXIT in LOOP
    #define _EXIT(x) IF (x) THEN JUMP _exit_endloop
20
    PROC main() HANDLE

            DEF cli = NIL : PTR TO cli, instr = NIL : PTR TO CHAR, allocout = FALSE

25          -> Initialise
            IF stdout = NIL     -> not started from system CLI (eg, from Workbench)
                    stdout := Open('KCON:////SLab/CLOSE/WAIT', NEWFILE)
                    IF stdout = NIL THEN CleanUp(20)     -> DOS return code FAIL
                    allocout := TRUE                     -> don't close if not ours
30          ENDIF
            initseed($AF7642B9)     -> random number seed
            NEW cli.new()

            -> Main loop
35          instr := String(1024)   -> allow long input
            LOOP
                    PrintF('>> ')       -> prompt
                    IF Fgets(stdout, instr, 1023) -> bug in OS, sometimes writes ↙
                        ↳ past end
                            _EXIT( cli.parse(instr) )
40                  ELSE
                            -> EOF, ie close window or control-\
                            PrintF('\n')    -> no return entered by user, neat ↙
                                ↳ output
                            _EXIT( TRUE )
                    ENDIF
45          ENDLOOP
    _exit_endloop:

            PrintF('** Bye!\n')     -> exit message

50  EXCEPT DO

            -> Clean up
            IF cli THEN END cli
            IF allocout
55                  Close(stdout)   -> will be open, CleanUp()ed above if not
                    stdout := NIL
            ENDIF

            -> Report fatal errors
60          SELECT exception
```

```
        CASE 0;      RETURN 0     -> no error
        CASE "MEM"; PrintF('** Error: out of memory\n')
        CASE "NEW"; PrintF('** Error: out of memory\n')
        -> Should never be seen
        CASE "asrt"
                PrintF('** Error: assertion failed in "\s"\n', exceptioninfo)
        DEFAULT
                PrintF('** Error: \z\h[8] \z\h[8]\n', exception, exceptioninfo)
        ENDSELECT

ENDPROC 5    -> DOS return code WARN

/*-----------------------------------------------------------------------+
| END: main.e                                                            |
+=======================================================================*/
```

45 Source/mul.e

```
/*=======================================================================+
| mul.e                                                                  |
| Effect class "mul", multiplies all inputs                              |
+-----------------------------------------------------------------------*/

OPT MODULE

MODULE '*defs', '*inmout1', '*debug'

EXPORT OBJECT mul OF inmout1
ENDOBJECT

PROC class() OF mul IS 'mul'

PROC process() OF mul
        DEF i, o = 0.0
        SUPER self.process()
        FOR i := 1 TO self._inputs() DO o := ! o * self._in(i)
        self.output(ID_MAIN, o)
ENDPROC

/*-----------------------------------------------------------------------+
| END: mul.e                                                             |
+=======================================================================*/
```

46 Source/notch.e

```
/*=======================================================================+
| notch.e                                                                |
| Effect class "notch"                                                   |
|                                                                        |
| Simple notch filter, using a zfilter with 2 poles and 2 zeros. The zeros |
| are at the "frequency" specified with radius 1, the poles are at radius |
| "depth". The closer "depth" is to 1, the narrower the notch.           |
|                                                                        |
| Features: input "main", output "main", parameters "depth" and "frequency" |
|                                                                        |
+-----------------------------------------------------------------------*/
```

```
OPT MODULE, PREPROCESS

MODULE '*container', '*defs', '*string', '*zfilter'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT notch OF container
PRIVATE
        filter : PTR TO zfilter         -> for delegation purposes only
ENDOBJECT

PROC class() OF notch IS 'notch'

/*-----------------------------------------------------------------------*/

PROC new(list, name) OF notch
        SUPER self.new(list, name)
        -> create parts
        NEW self.filter.new(self.list, 'notch_zfilter')
        -> set parameters
        self.filter.set(ID_POLES, 2.0)
        self.filter.set(ID_ZEROS, 2.0)
        self.filter.set(ID_MULTI_ZERO_R + 1, 1.0)
        self.filter.set(ID_MULTI_ZERO_R + 2, 1.0)
        self.set(ID_DEPTH, 0.9)
        self.set(ID_FREQUENCY, 440.0)
ENDPROC

/*-----------------------------------------------------------------------*/

PROC input2id(str) OF notch
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.input2id(str)

PROC output2id(str) OF notch
ENDPROC IF strcmp(IDS_MAIN, str) THEN ID_MAIN ELSE SUPER self.output2id(str)

PROC id2input(id) OF notch
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2input(id)

PROC id2output(id) OF notch
ENDPROC IF id = ID_MAIN THEN IDS_MAIN ELSE SUPER self.id2output(id)

/*-----------------------------------------------------------------------*/

PROC param2id(str) OF notch
        IF     strcmp(IDS_DEPTH,     str); RETURN ID_DEPTH
        ELSEIF strcmp(IDS_FREQUENCY, str); RETURN ID_FREQUENCY
        ENDIF
ENDPROC SUPER self.param2id(str)

/*-----------------------------------------------------------------------*/

PROC id2param(id) OF notch
        SELECT id
        CASE ID_DEPTH;     RETURN IDS_DEPTH
        CASE ID_FREQUENCY; RETURN IDS_FREQUENCY
```

```
        ENDSELECT
ENDPROC SUPER self.id2param(id)

/*-------------------------------------------------------------------------*/

PROC paramtype(id) OF notch
        SELECT id
        CASE ID_DEPTH;     RETURN self.filter.paramtype(ID_MULTI_POLE_R + 1)
        CASE ID_FREQUENCY; RETURN self.filter.paramtype(ID_MULTI_POLE_F + 1)
        ENDSELECT
ENDPROC SUPER self.paramtype(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF notch
        SELECT id
        CASE ID_DEPTH
                self.filter.set(ID_MULTI_POLE_R + 1, data)
                self.filter.set(ID_MULTI_POLE_R + 2, data)
                self.setrecalc()
        CASE ID_FREQUENCY
                self.filter.set(ID_MULTI_POLE_F + 1, data)
                self.filter.set(ID_MULTI_POLE_F + 2, ! -data)
                self.filter.set(ID_MULTI_ZERO_F + 1, data)
                self.filter.set(ID_MULTI_ZERO_F + 2, ! -data)
                self.setrecalc()
        DEFAULT; SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC get(id) OF notch
        SELECT id
        CASE ID_DEPTH;     RETURN self.filter.get(ID_MULTI_POLE_R + 1)
        CASE ID_FREQUENCY; RETURN self.filter.get(ID_MULTI_POLE_F + 1)
        ENDSELECT
ENDPROC SUPER self.get(id)

/*-------------------------------------------------------------------------*/

PROC getinput(id) OF notch
        SELECT id
        CASE ID_MAIN; RETURN self.filter.getinput(id)
        ENDSELECT
ENDPROC SUPER self.getinput(id)

/*-------------------------------------------------------------------------*/

PROC setoutput(id, link) OF notch
        SELECT id
        CASE ID_MAIN; RETURN self.filter.setoutput(id, link)
        ENDSELECT
ENDPROC SUPER self.setoutput(id, link)

/*-------------------------------------------------------------------------+
| END: notch.e                                                             |
```

```
+=============================================================================*/
```

47 Source/osc.e

```
/*=============================================================================+
| osc.e                                                                        |
| Effect base class "osc"                                                      |
|                                                                              |
| Oscillator, controlled by parameters "frequency", "amplitude", "phase"       |
|                                                                              |
| osc.oscillator(time)  implemented by derived classes, this is called with   |
|                       0.0 <= time < 1.0, calculated from the parameters      |
|                       and the current sample time                            |
|                                                                              |
+-----------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*in0out1', '*string', '*value', '*debug'

/*-----------------------------------------------------------------------------*/

EXPORT OBJECT osc OF in0out1
PRIVATE
        freq    : LONG   -> frequency
        ampl    : LONG   -> amplitude
        phase   : LONG   -> phase
        time    : LONG   -> "time"
        dt      : LONG   -> "time" step
ENDOBJECT

PROC class() OF osc IS 'osc'
PROC oscillator(time) OF osc IS 0.0     -> compiler: unreferenced "time"

/*-----------------------------------------------------------------------------*/

PROC new(list, name) OF osc
        SUPER self.new(list, name)
        self.rate   := DEF_RATE
        self.freq   := 440.0    -> concert pitch A
        self.ampl   := 1.0
        self.phase  := 0.0
        self.time   := 0.0
        self.setrecalc()        -> dt needs to be recalculated
ENDPROC

/*-----------------------------------------------------------------------------*/

PROC param2id(str) OF osc
        IF      strcmp(IDS_FREQUENCY, str); RETURN ID_FREQUENCY
        ELSEIF strcmp(IDS_AMPLITUDE, str); RETURN ID_AMPLITUDE
        ELSEIF strcmp(IDS_PHASE,     str); RETURN ID_PHASE
        ENDIF
ENDPROC SUPER self.param2id(str)

/*-----------------------------------------------------------------------------*/
```

```
PROC id2param(id) OF osc
        SELECT id
        CASE ID_FREQUENCY; RETURN IDS_FREQUENCY
        CASE ID_AMPLITUDE; RETURN IDS_AMPLITUDE
        CASE ID_PHASE;     RETURN IDS_PHASE
        ENDSELECT
ENDPROC SUPER self.id2param(id)

/*-------------------------------------------------------------------------*/

PROC paramtype(id) OF osc
        SELECT id
        CASE ID_FREQUENCY; RETURN TYPE_NUMBER
        CASE ID_AMPLITUDE; RETURN TYPE_NUMBER
        CASE ID_PHASE;     RETURN TYPE_NUMBER
        ENDSELECT
ENDPROC SUPER self.paramtype(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF osc
        SELECT id
        CASE ID_FREQUENCY; self.freq  := data; self.setrecalc()
        CASE ID_AMPLITUDE; self.ampl  := data
        CASE ID_PHASE;     self.phase := data
        DEFAULT;           SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC get(id) OF osc
        SELECT id
        CASE ID_FREQUENCY; RETURN self.freq
        CASE ID_AMPLITUDE; RETURN self.ampl
        CASE ID_PHASE;     RETURN self.phase
        ENDSELECT
ENDPROC SUPER self.get(id)

/*-------------------------------------------------------------------------*/

PROC recalc() OF osc
        SUPER self.recalc()
        self.dt := ! self.freq / self.rate
ENDPROC

/*-------------------------------------------------------------------------*/

PROC reset() OF osc
        SUPER self.reset()
        self.time := 0.0
ENDPROC

/*-------------------------------------------------------------------------*/

PROC process() OF osc
        DEF time
```

```
        SUPER self.process()
        time      := ! self.time + self.dt              -> next t
        self.time := ! time - Ffloor(time)              -> 0 <= t < 1
        time      := ! self.time + self.phase
        time      := ! time - Ffloor(time)              -> 0 <= t < 1
        self.output(ID_MAIN, ! self.ampl * self.oscillator(time))
ENDPROC

/*-----------------------------------------------------------------------+
| END: osc.e                                                             |
+=======================================================================*/
```

48 Source/print.e

```
/*=======================================================================+
| print.e                                                                |
| Effect class "print"                                                   |
| Used in debugging only, as it prints the object name and the input to the |
| screen each time process is called.                                    |
+-----------------------------------------------------------------------*/

OPT MODULE
OPT EXPORT

MODULE '*defs', '*in1out0', '*list'

RAISE "MEM" IF String() = NIL

/*-----------------------------------------------------------------------*/

OBJECT print OF in1out0
PRIVATE
        buffer  : PTR TO CHAR       -> E-String
ENDOBJECT

PROC class() OF print IS 'print'

/*-----------------------------------------------------------------------*/

PROC new(list, name) OF print
        SUPER self.new(list, name)
        self.buffer := String(32)
ENDPROC

/*-----------------------------------------------------------------------*/

PROC end() OF print
        DisposeLink(self.buffer)
        self.buffer := NIL
ENDPROC SUPER self.end()

/*-----------------------------------------------------------------------*/

PROC process() OF print
        DEF n : PTR TO node
        SUPER self.process()
        n := self.node
```

```
        PrintF('\s \s\n', n.name, RealF(self.buffer, self._main(), 8))
ENDPROC

/*-------------------------------------------------------------------------+
| END: print.e                                                             |
+=========================================================================*/
```

49 Source/pulse.e

```
/*=========================================================================+
| pulse.e                                                                  |
| Effect class "pulse"                                                     |
| A pulse wave oscillator, with parameter "width"                          |
+-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS
OPT EXPORT

MODULE '*defs', '*osc', '*string', '*value'

/*-------------------------------------------------------------------------*/

OBJECT pulse OF osc
PRIVATE
        width : LONG
ENDOBJECT

PROC class() OF pulse IS 'pulse'
PROC oscillator(time) OF pulse IS IF ! time > self.width THEN -1.0 ELSE 1.0

/*-------------------------------------------------------------------------*/

PROC new(list, name) OF pulse
        SUPER self.new(list, name)
        self.width := 0.5
ENDPROC

/*-------------------------------------------------------------------------*/

PROC param2id(str) OF pulse
ENDPROC IF strcmp(IDS_WIDTH, str) THEN ID_WIDTH ELSE SUPER self.param2id(str)

PROC id2param(id) OF pulse
ENDPROC IF id = ID_WIDTH THEN IDS_WIDTH ELSE SUPER self.id2param(id)

PROC paramtype(id) OF pulse
ENDPROC IF id = ID_WIDTH THEN TYPE_NUMBER ELSE SUPER self.paramtype(id)

PROC get(id) OF pulse
ENDPROC IF id = ID_WIDTH THEN self.width ELSE SUPER self.get(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF pulse
        SELECT id
        CASE ID_WIDTH; self.width := data
        DEFAULT;       SUPER self.set(id, data)
```

```
        ENDSELECT
ENDPROC

/*------------------------------------------------------------------------+
| END: pulse.e                                                            |
+=======================================================================*/
```

50 Source/ramp.e

```
/*=======================================================================+
| ramp.e                                                                  |
| Effect class "ramp", ramp oscillator                                    |
+------------------------------------------------------------------------*/

OPT MODULE

MODULE '*osc'

EXPORT OBJECT ramp OF osc
ENDOBJECT

PROC oscillator(time) OF ramp IS ! 2.0 * time - 1.0

PROC class() OF ramp IS 'ramp'

/*------------------------------------------------------------------------+
| END: ramp.e                                                             |
+=======================================================================*/
```

51 Source/read8svx.e

```
/*=======================================================================+
| read8svx.e                                                              |
| Effect class "read8svx", read from an IFF 8SVX sample file              |
+------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*file', '*read', '*iff8svx_ff', '*string', '*value',
       '*debug', 'dos/dos'

/*------------------------------------------------------------------------*/

EXPORT OBJECT read8svx OF read
PRIVATE
        vhdr : iff8svx_vhdr -> data
        len  : LONG         -> number of samples
        pos  : LONG         -> current sample position (starting at 0)
ENDOBJECT

PROC class() OF read8svx IS 'read8svx'

/*------------------------------------------------------------------------*/

PROC new(list, name) OF read8svx
        SUPER self.new(list, name)
```

```
ENDPROC

/*------------------------------------------------------------------------*/

-> Workaround for deficiency in E
#define _EXIT(x) IF (x) THEN JUMP _exit_endloop

PROC startread() OF read8svx
        DEF fh, id, len, gotvhdr = FALSE
        IF fh := self._fh()
                IF read_ulong(fh) <> MAGIC_FORM THEN Throw(ERR_BAD_FILE_FORMAT, ↙
                    ↳ fh)
                read_ulong(fh)  -> file length - 4
                IF read_ulong(fh) <> MAGIC_8SVX THEN Throw(ERR_BAD_FILE_FORMAT, ↙
                    ↳ fh)
                LOOP    -> up to BODY chunk
                        id  := read_ulong(fh)   -> Exception exits loop if EOF
                        len := read_ulong(fh)
                        SELECT id
                        CASE MAGIC_VHDR         -> voice header
                                gotvhdr := TRUE
                                IF len <> SIZEOF iff8svx_vhdr
                                        Throw(ERR_BAD_FILE_FORMAT, fh)
                                ENDIF
                                -> len = SIZEOF iff8svx_vhdr
                                IF read(fh, self.vhdr, len) = FALSE
                                        Throw(ERR_BAD_FILE_FORMAT, fh)
                                ENDIF
                                -> check parameters
                                IF (self.vhdr.octaves <> 1) OR      -> no ↙
                                    ↳ multioctave
                                   (self.vhdr.compression <> 0)     -> no ↙
                                       ↳ compression
                                        Throw(ERR_BAD_FILE_FORMAT, fh)
                                ENDIF
                        CASE MAGIC_BODY
                                IF gotvhdr = FALSE THEN Throw(↙
                                    ↳ ERR_BAD_FILE_FORMAT, fh)
                                -> At start of sample data
                                self.len := len
                                _EXIT(TRUE)
                        DEFAULT
                                -> Skip unknown chunk
                                Seek(fh, len, OFFSET_CURRENT)
                        ENDSELECT
                ENDLOOP
_exit_endloop:
        ENDIF
        self.pos := 0
ENDPROC

/*------------------------------------------------------------------------*/

PROC process() OF read8svx
        DEF smp = 0.0
        SUPER self.process()
        IF (self.pos < self.len) AND self._fh()
```

```
                smp := read_sbyte(self._fh()) ! / 128.0
        ENDIF
        self.pos := self.pos + 1
ENDPROC self.output(ID_MAIN, smp)

/*-----------------------------------------------------------------------+
| END: read8svx.e                                                        |
+=======================================================================*/
```

52 Source/read.e

```
/*=======================================================================+
| read.e                                                                 |
| Effect base class "read"                                               |
| Read from a file, set by parameter "file"                              |
|                                                                        |
| read.startread()      sort out header info etc, called when file changed |
| read.stopread()       clean up, called when file changed               |
|                       both the above return success, new versions should |
|                       call SUPER, both should handle fh being NIL      |
| read._fh()            get file handle                                  |
| read.setfile(name)    set file to name, calls stopread and startread    |
|                                                                        |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS
OPT EXPORT

MODULE '*defs', '*in0out1', '*string', '*value'

/*-----------------------------------------------------------------------*/

OBJECT read OF in0out1
PRIVATE
        fname : PTR TO CHAR     -> file name
        fh    : LONG            -> file handle
ENDOBJECT

PROC class()     OF read IS 'read'
PROC _fh()       OF read IS self.fh
PROC startread() OF read IS TRUE      -> read header info
PROC stopread()  OF read IS TRUE      -> consistency checks ?

/*-----------------------------------------------------------------------*/

PROC setfile(fname) OF read
        self.stopread()
        IF self.fh THEN Close(self.fh)
        self.fname := fname
        self.fh := IF fname THEN Open(fname, OLDFILE) ELSE NIL
        self.startread()
ENDPROC self.fh <> NIL

/*-----------------------------------------------------------------------*/

PROC new(list, name) OF read
        SUPER self.new(list, name)
```

```
        self.setfile(NIL)
ENDPROC

/*-------------------------------------------------------------------------*/

PROC end() OF read
        self.setfile(NIL)
ENDPROC SUPER self.end()

/*-------------------------------------------------------------------------*/

PROC param2id(str) OF read
ENDPROC IF strcmp(IDS_FILE, str) THEN ID_FILE ELSE SUPER self.param2id(str)

PROC id2param(id) OF read
ENDPROC IF id = ID_FILE THEN IDS_FILE ELSE SUPER self.id2param(id)

PROC paramtype(id) OF read
ENDPROC IF id = ID_FILE THEN TYPE_STRING ELSE SUPER self.paramtype(id)

PROC set(id, data) OF read
ENDPROC IF id = ID_FILE THEN self.setfile(data) ELSE SUPER self.set(id, data)

PROC get(id) OF read
ENDPROC IF id = ID_FILE THEN self.fname ELSE SUPER self.get(id)

/*-------------------------------------------------------------------------*/

PROC reset() OF read
        SUPER self.reset()
        self.setfile(self.fname)    -> reset file to start
ENDPROC

/*-------------------------------------------------------------------------+
| END: read.e                                                              |
+=========================================================================*/
```

53 Source/readslab.e

```
/*=========================================================================+
| readslab.e                                                               |
| Effect class "readslab"                                                  |
| Read from a SLab sample file (see slab.txt)                              | ↙
    ↳                                                                        ↙
    ↳ |
| Parameter "normalize" = "on" / "off" normalizes output                   |
+-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS
OPT EXPORT

MODULE '*defs', '*file', '*read', '*slab_ff', '*string', '*value', '*debug'

/*-------------------------------------------------------------------------*/

OBJECT readslab OF read
PRIVATE
```

```
        info : slab_info    -> data
        norm : LONG         -> normalize?
        len  : LONG         -> number of samples
        pos  : LONG         -> current sample position (starting at 0)
ENDOBJECT

PROC class() OF readslab IS 'readslab'

/*-----------------------------------------------------------------------*/

PROC new(list, name) OF readslab
        SUPER self.new(list, name)
        self.norm := TRUE
ENDPROC

/*-----------------------------------------------------------------------*/

PROC startread() OF readslab
        DEF fh
        IF fh := self._fh()
                IF read_ulong(fh) <> MAGIC_SLab THEN Throw(ERR_BAD_FILE_FORMAT, fh)
                read_ulong(fh)  -> file length - 4
                IF read_ulong(fh) <> MAGIC_Info THEN Throw(ERR_BAD_FILE_FORMAT, fh)
                IF read_ulong(fh) <> 12         THEN Throw(ERR_BAD_FILE_FORMAT, fh)
                self.info.rate := read_ulong(fh)
                self.info.bias := read_ulong(fh)
                self.info.ampl := read_ulong(fh)
                IF read_ulong(fh) <> MAGIC_Data THEN Throw(ERR_BAD_FILE_FORMAT, fh)
                self.len := Div(read_ulong(fh), 4)   -> standard */ are only 16 bit
        ENDIF
        self.pos := 0
ENDPROC

/*-----------------------------------------------------------------------*/

PROC param2id(str) OF readslab
        IF strcmp(IDS_NORMALIZE, str); RETURN ID_NORMALIZE
        ENDIF
ENDPROC SUPER self.param2id(str)

/*-----------------------------------------------------------------------*/

PROC id2param(id) OF readslab
        SELECT id
        CASE ID_NORMALIZE; RETURN IDS_NORMALIZE
        ENDSELECT
ENDPROC SUPER self.id2param(id)

/*-----------------------------------------------------------------------*/

PROC paramtype(id) OF readslab
        SELECT id
```

```
        CASE ID_NORMALIZE; RETURN TYPE_STRING
        ENDSELECT
ENDPROC SUPER self.paramtype(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF readslab
        SELECT id
        CASE ID_NORMALIZE
                IF strcmp(IDS_ON, data)
                        self.norm := TRUE
                ELSEIF strcmp(IDS_OFF, data)
                        self.norm := FALSE
                ELSE
                        Throw(ERR_BAD_RANGE, data)
                ENDIF
                RETURN TRUE
        ENDSELECT
ENDPROC SUPER self.set(id, data)

/*-------------------------------------------------------------------------*/

PROC get(id) OF readslab
        SELECT id
        CASE ID_NORMALIZE; RETURN IF self.norm THEN IDS_ON ELSE IDS_OFF
        ENDSELECT
ENDPROC SUPER self.get(id)

/*-------------------------------------------------------------------------*/

PROC process() OF readslab
        DEF smp = 0.0
        SUPER self.process()
        IF (self.pos < self.len) AND self._fh()
                smp := read_ulong(self._fh())
                IF self.norm THEN smp := ! (! smp - self.info.bias) / self.info.↙
                    ↳ ampl
        ENDIF
        self.pos := self.pos + 1
ENDPROC self.output(ID_MAIN, smp)

/*-------------------------------------------------------------------------+
| END: readslab.e                                                          |
+=========================================================================*/
```

54 Source/rectify.e

```
/*=========================================================================+
| rectify.e                                                                |
| Effect class "rectify", rectifies input                                  |
+-------------------------------------------------------------------------*/

OPT MODULE

MODULE '*defs', '*in1out1'

EXPORT OBJECT rectify OF in1out1
```

```
ENDOBJECT

PROC class() OF rectify IS 'rectify'

PROC process() OF rectify
        SUPER self.process()
        self.output(ID_MAIN, Fabs(self._main()))
ENDPROC

/*-----------------------------------------------------------------------+
| END: rectify.e                                                         |
+=======================================================================*/
```

55 Source/rnd.e

```
/*=======================================================================+
| rnd.e                                                                  |
| Generate random numbers                                                |
|                                                                        |
| initseed(newseed)     initialise the seed                              |
| frnd()                returns a random float between -1 and 1          |
|                                                                        |
+-----------------------------------------------------------------------*/

OPT MODULE

DEF seed    -> E doesn't allow globals in modules to be initialised

EXPORT PROC initseed(newseed) IS seed := newseed

-> RndQ gives a full-range 32 bit value, not really very random though
EXPORT PROC frnd() IS (seed := RndQ(seed)) ! / (Shl(1, 31) !)

/*=======================================================================+
| END: rnd.e                                                             |
+-----------------------------------------------------------------------*/
```

56 Source/scale.e

```
/*=======================================================================+
| scale.e                                                                |
| Effect class "scale", scales input to fit a new range                  |
+-----------------------------------------------------------------------*/

OPT MODULE

MODULE '*defs', '*in1out1'

EXPORT OBJECT scale OF in1out1
ENDOBJECT

PROC class() OF scale IS 'scale'

PROC process() OF scale
        SUPER self.process()
        self.output(ID_MAIN, scale(0.0, 1.0, 20.0, 2000.0, self._main()))
```

```
ENDPROC

PROC scale(min0, max0, min1, max1, x)
        DEF shift, scale
        shift := ! max1 - max0
        scale := ! (! max1 - min1) / (! max0 - min0)
ENDPROC ! (! x + shift) * scale

/*-----------------------------------------------------------------------+
| END: scale.e                                                           |
+=======================================================================*/
```

57 Source/sine.e

```
/*=======================================================================+
| sine.e                                                                 |
| Effect class "sine", sine oscillator                                   |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*osc'

EXPORT OBJECT sine OF osc
ENDOBJECT

PROC oscillator(time) OF sine IS Fsin(! PI2 * time)  -> 2 PI * time

PROC class() OF sine IS 'sine'

/*-----------------------------------------------------------------------+
| END: sine.e                                                            |
+=======================================================================*/
```

58 Source/slab_ff.e

```
/*=======================================================================+
| slab_ff.e                                                              |
| SLab file format definitions (see slab.txt)                            |
+-----------------------------------------------------------------------*/

OPT MODULE
OPT EXPORT

/*-----------------------------------------------------------------------*/

OBJECT slab_info
        rate : LONG     -> really floats
        bias : LONG
        ampl : LONG
ENDOBJECT

/*-----------------------------------------------------------------------*/

CONST MAGIC_SLab = "SLab", MAGIC_Info = "Info", MAGIC_Data = "Data"
```

```
/*-------------------------------------------------------------------------+
| END: slab_ff.e                                                           |
+=========================================================================*/
```

59 Source/split.e

```
/*=========================================================================+
| split.e                                                                  |
| Effect class "split", sends input to all outputs                         |
+-------------------------------------------------------------------------*/

OPT MODULE

MODULE '*defs', '*in1outm', '*debug'

EXPORT OBJECT split OF in1outm
ENDOBJECT

PROC class() OF split IS 'split'

PROC process() OF split
        DEF i
        SUPER self.process()
        FOR i := 1 TO self._outputs() DO self.output(self._out(i), self._in())
ENDPROC

/*-------------------------------------------------------------------------+
| END: split.e                                                             |
+=========================================================================*/
```

60 Source/string.e

```
/*=========================================================================+
| string.e                                                                 |
| String functions                                                         |
|                                                                          |
| strcmp(str1, str2)        returns TRUE if equal (case insensitive)       |
| strncmp(str1, str2, n)    as strcmp, with number of chars to compare     |
|                                                                          |
+-------------------------------------------------------------------------*/

OPT MODULE
OPT EXPORT

/*-------------------------------------------------------------------------*/

PROC strcmp(s1 : PTR TO CHAR, s2 : PTR TO CHAR)
        IF (s1 = NIL) OR (s2 = NIL) THEN RETURN FALSE
        WHILE s1[0] = s2[0]
                IF s1[0] = 0 THEN RETURN TRUE
                INC s1
                INC s2
        ENDWHILE
ENDPROC FALSE

/*-------------------------------------------------------------------------*/
```

```
25
PROC strncmp(s1 : PTR TO CHAR, s2 : PTR TO CHAR, n)
        DEF i
        IF (s1 = NIL) OR (s2 = NIL) THEN RETURN FALSE
        FOR i := 0 TO n - 1
                IF s1[i] <> s2[i] THEN RETURN FALSE
        ENDFOR
ENDPROC TRUE

/*-----------------------------------------------------------------------+
35 | END: string.e                                                        |
+=======================================================================*/
```

61 Source/testcbuffer.e

```
/*=======================================================================+
| testcbuffer.e                                                         |
| Test circular buffer class                                            |
+-----------------------------------------------------------------------*/
5
MODULE '*cbuffer'

ENUM ERR_OK = 0, ERR_MATHLIB

10 RAISE "MEM" IF String() = NIL

PROC main() HANDLE

        DEF a   = NIL   : PTR TO cbuffer,
15          b   = NIL   : PTR TO cbuffer,
            c   = NIL   : PTR TO cbuffer,
            in  = 0.0   : LONG,
            i   = 0     : LONG,
            bin = NIL   : PTR TO CHAR,      -> estring buffers for RealF()
20          ba  = NIL   : PTR TO CHAR,
            bb  = NIL   : PTR TO CHAR,
            bc  = NIL   : PTR TO CHAR,
            bd  = NIL   : PTR TO CHAR

25      NEW a.new(5.0)      -> check integer length
        NEW b.new(5.5)      -> check fractional length
        NEW c.new(1.0)      -> check short length

        bin := String(16)
30      ba  := String(16)
        bb  := String(16)
        bc  := String(16)
        bd  := String(16)

35      PrintF('i\tin\ta\tb\tc\ta(-3.2)\n')

        FOR i := 1 TO 80
                IF i = 10 THEN c.setlength(50.0)    -> check reallocation
                IF i = 20 THEN a.setlength(8.0)     -> check increase
40              IF i = 40 THEN b.setlength(3.0)     -> check decrease
                IF i = 50 THEN a.clear()            -> check clear
                IF i = 60 THEN c.setlength(0.0)     -> check zero length
```

```
                in := i ! * .1
                a.write(in)                         -> check write
                b.write(in)
                c.write(in)
                RealF(bin, in, 4)
                RealF(ba, a.read(), 4)              -> check read
                RealF(bb, b.read(), 4)
                RealF(bc, c.read(), 4)
                RealF(bd, a.readrel(-3.2), 4)       -> check relative read
                PrintF('\d[2]\t\s\t\s\t\s\t\s\t\s\t\s\n', i, bin, ba, bb, bc, bd)
                a.next()                            -> check next
                b.next()
                c.next()
        ENDFOR

EXCEPT DO

        IF bc  THEN Dispose(bc)
        IF bb  THEN Dispose(bb)
        IF ba  THEN Dispose(ba)
        IF bin THEN Dispose(bin)

        IF c THEN END c
        IF b THEN END b
        IF a THEN END a

        SELECT exception
        CASE ERR_OK
                RETURN 0
        CASE ERR_MATHLIB
                PrintF('** Error: couldn''t open "mathieeesingbas.library"\n')
        CASE "asrt"
                PrintF('** Error: assertion failed in "\s"\n', exceptioninfo)
        DEFAULT
                PrintF('** Unknown error: \z\h[8] \z\h[8]\n',
                                exception, exceptioninfo)
        ENDSELECT

ENDPROC 5

/*-----------------------------------------------------------------------+
| END: testcbuffer.e                                                     |
+=======================================================================*/
```

62 Source/testrnd.e

```
/*=======================================================================+
| testrnd.e                                                              |
| Test random numbers                                                    |
+-----------------------------------------------------------------------*/

OPT REG = 5     -> use register variables

MODULE '*rnd'

PROC main()
```

```
        DEF num = 0.0, sum = 0.0, avg = 0.0, abssum = 0.0, absavg = 0.0,
            hicnt = 0, locnt = 0, hieqcnt = 0, loeqcnt = 0, buf[16] : STRING, i

        initseed($3F44A8B3)

        FOR i := 1 TO 100000
                num := frnd()
                sum := ! sum + num
                abssum := ! abssum + Fabs(num)
                IF ! num >  1.0 THEN INC hicnt
                IF ! num < -1.0 THEN INC locnt
                IF ! num =  1.0 THEN INC hieqcnt
                IF ! num = -1.0 THEN INC loeqcnt
        ENDFOR
        avg := ! sum / 100000.0
        absavg := ! abssum / 100000.0

        PrintF('avg     = \s, expected 0.0\n', RealF(buf, avg,    8))
        PrintF('absavg  = \s, expected 0.5\n', RealF(buf, absavg, 8))
        PrintF('hicnt   = \d, should be 0\n',  hicnt)
        PrintF('locnt   = \d, should be 0\n',  locnt)
        PrintF('hieqcnt = \d, should be 0\n',  hieqcnt)
        PrintF('loeqcnt = \d, should be 0\n',  loeqcnt)

ENDPROC

/*-----------------------------------------------------------------------+
| END: testrnd.e                                                         |
+=======================================================================*/
```

63 Source/toparam.e

```
/*=======================================================================+
| toparam.e                                                              |
| Effect class "toparam"                                                 |
| Set parameters according to sample data                                |
|                                                                        |
| isready() is FALSE, so that process is not called when input is recieved |
| issource() is TRUE, so process is called by kernel in run              |
|                                                                        |
| The run command moves all toparam objects to the end of the list, so that |
| their input will have arrived by the time process is called.  For this |
| reason class() should not be replaced in any derived classes.          |
|                                                                        |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*effect', '*in1out0', '*string', '*value', '*debug'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT toparam OF in1out0
PRIVATE
        obj   : PTR TO effect
        pid   : LONG
        last  : LONG
```

```
ENDOBJECT

PROC class()    OF toparam IS 'toparam'
PROC issource() OF toparam IS TRUE
PROC isready()  OF toparam IS FALSE

/*-------------------------------------------------------------------------*/

PROC new(list, name) OF toparam
        SUPER self.new(list, name)
        self.obj  := NIL
        self.pid  := ID_INVALID
        self.last := 0.0
ENDPROC

/*-------------------------------------------------------------------------*/

PROC param2id(str) OF toparam
ENDPROC IF strcmp(IDS_TO, str) THEN ID_TO ELSE SUPER self.param2id(str)

PROC id2param(id) OF toparam
ENDPROC IF id = ID_TO THEN IDS_TO ELSE SUPER self.id2param(id)

PROC paramtype(id) OF toparam
ENDPROC IF id = ID_TO THEN TYPE_OBJPART ELSE SUPER self.paramtype(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF toparam
        DEF op : PTR TO value_objpart
        SELECT id
        CASE ID_TO
                op := data
                self.obj := op.obj  -> (valid) pointer, but op.pid is string
                self.pid := self.obj.param2id(op.pid)
                IF self.pid = ID_INVALID THEN Throw(ERR_NO_SUCH_PARAM, op.pid)
                IF self.obj.paramtype(self.pid) <> TYPE_NUMBER
                        Throw(ERR_PARAM_NOT_NUMBER, op.pid)
                ENDIF
                self.last := 0.0
        DEFAULT; SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC get(id) OF toparam
        SELECT id
        CASE ID_TO; RETURN [ self.obj, self.pid ] : value_objpart
        ENDSELECT
ENDPROC SUPER self.get(id)

/*-------------------------------------------------------------------------*/

PROC process() OF toparam
        SUPER self.process()
        IF (! self.last <> self._main()) AND self.obj
```

```
                self.obj.set(self.pid, self._main())
        ENDIF
ENDPROC

/*-----------------------------------------------------------------------*/

PROC clear() OF toparam
        self.last := self._main()
        SUPER self.clear()
ENDPROC

/*-----------------------------------------------------------------------*/

PROC reset() OF toparam
        SUPER self.reset()
        self.last := 0.0
ENDPROC

/*-----------------------------------------------------------------------+
| END: feedback.e                                                       |
+=======================================================================*/
```

64 Source/triangle.e

```
/*=======================================================================+
| triangle.e                                                            |
| Effect class "triangle", triangle oscillator                          |
+-----------------------------------------------------------------------*/

OPT MODULE

MODULE '*osc'

EXPORT OBJECT triangle OF osc
ENDOBJECT

PROC oscillator(time) OF triangle
        IF ! time < 0.25 THEN RETURN ! 4.0 * time
        IF ! time < 0.75 THEN RETURN ! 4.0 * (! 0.5 - time)
        IF ! time < 1.00 THEN RETURN ! 4.0 * (! time - 1.0)
ENDPROC 0.0

PROC class() OF triangle IS 'triangle'

/*-----------------------------------------------------------------------+
| END: triangle.e                                                       |
+=======================================================================*/
```

65 Source/value.e

```
/*=======================================================================+
| value.e                                                               |
| Value structure                                                       |
+-----------------------------------------------------------------------*/

OPT MODULE
```

```
OPT EXPORT

/*-----------------------------------------------------------------------*/

OBJECT value
PUBLIC
        type    : LONG      -> TYPE_#?
        data    : LONG
ENDOBJECT

/*-----------------------------------------------------------------------*/

OBJECT value_objpart
PUBLIC
        obj     : LONG      -> pointer to effect object (or name)
        pid     : LONG      -> effect part (input, output, param) ID (or name)
ENDOBJECT

/*-----------------------------------------------------------------------*/

ENUM TYPE_INVALID = 0,
     TYPE_NUMBER,       -> float
     TYPE_STRING,       -> character string
     TYPE_OBJPART       -> value_objpart

/*-----------------------------------------------------------------------+
| END: value.e                                                           |
+=======================================================================*/
```

66 Source/vox.e

```
/*=======================================================================+
| vox.e                                                                  |
| Effect class "vox"                                                     |
|                                                                        |
| Sends out input only once it has exceeded parameter "threshold"        |
|                                                                        |
| WARNING: effects linked (even indirectly) to one vox can have no inputs|
| not from that vox without unpredictable results (removing a certain vox|
| should cause the network of effects to fall into two unconnected parts).|
|                                                                        |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*in1out1', '*string', '*value'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT vox OF in1out1
PRIVATE
        threshold : LONG    -> cutoff level
        max       : LONG    -> current maximum attained (threshold may change)
ENDOBJECT

PROC class() OF vox IS 'vox'
```

```
/*-------------------------------------------------------------------------*/

PROC new(list , name) OF vox
        SUPER self.new(list , name)
        self.threshold := 0.0001
        self.reset()
ENDPROC

/*-------------------------------------------------------------------------*/

PROC param2id(str) OF vox
        IF strcmp(IDS_THRESHOLD, str); RETURN ID_THRESHOLD
        ENDIF
ENDPROC SUPER self.param2id(str)

/*-------------------------------------------------------------------------*/

PROC id2param(id) OF vox
        SELECT id
        CASE ID_THRESHOLD; RETURN IDS_THRESHOLD
        ENDSELECT
ENDPROC SUPER self.id2param(id)

/*-------------------------------------------------------------------------*/

PROC paramtype(id) OF vox
        SELECT id
        CASE ID_THRESHOLD; RETURN TYPE_NUMBER
        ENDSELECT
ENDPROC SUPER self.paramtype(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF vox
        SELECT id
        CASE ID_THRESHOLD; self.threshold := data
        DEFAULT;           SUPER self.set(id, data)
        ENDSELECT
ENDPROC

/*-------------------------------------------------------------------------*/

PROC get(id) OF vox
        SELECT id
        CASE ID_THRESHOLD; RETURN self.threshold
        ENDSELECT
ENDPROC SUPER self.get(id)

/*-------------------------------------------------------------------------*/

PROC reset() OF vox
        SUPER self.reset()
        self.max := 0.0
ENDPROC

/*-------------------------------------------------------------------------*/
```

```
PROC process() OF vox
        SUPER self.process()
        IF ! Fabs(self._main()) > self.max THEN self.max := Fabs(self._main())
        IF ! self.max > self.threshold THEN self.output(ID_MAIN, self._main())
ENDPROC

/*-----------------------------------------------------------------------+
| END: vox.e                                                             |
+=======================================================================*/
```

67 Source/whitenoise.e

```
/*=======================================================================+
| whitenoise.e                                                           |
| Effect class "whitenoise", white noise (random values, between -1 and 1)  |
+-----------------------------------------------------------------------*/

OPT MODULE

MODULE '*defs', '*in0out1', '*rnd'

EXPORT OBJECT whitenoise OF in0out1
ENDOBJECT

PROC class() OF whitenoise IS 'whitenoise'

PROC process() OF whitenoise
        SUPER self.process()
        self.output(ID_MAIN, frnd())
ENDPROC

/*-----------------------------------------------------------------------+
| END: whitenoise.e                                                      |
+=======================================================================*/
```

68 Source/write8svx.e

```
/*=======================================================================+
| write8svx.e                                                            |
| Effect class "write8svx", write to an IFF 8SVX sample file             |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*file', '*iff8svx_ff', '*rnd', '*string', '*value',
       '*write', 'dos/dos'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT write8svx OF write
PRIVATE
        vhdr : iff8svx_vhdr
        pos  : LONG         -> current sample position (starting at 0)
ENDOBJECT

PROC class() OF write8svx IS 'write8svx'
```

```
20
    /*------------------------------------------------------------------------*/

    PROC new(list, name) OF write8svx
            SUPER self.new(list, name)
25          self.vhdr.hioctsamples := 0         -> fixed in stopwrite
            self.vhdr.repeatstart  := 0
            self.vhdr.repeatlength := 0
            self.vhdr.rate         := 44100     -> fixed in stopwrite
            self.vhdr.octaves      := 1
30          self.vhdr.compression  := 0
            self.vhdr.volume       := 65536
    ENDPROC

    /*------------------------------------------------------------------------*/
35
    PROC startwrite() OF write8svx
            DEF fh
            IF fh := self._fh()
                    write_ulong(fh, MAGIC_FORM)
40                  write_ulong(fh, 0)          -> fixed in stopwrite
                    write_ulong(fh, MAGIC_8SVX)
                    write_ulong(fh, MAGIC_VHDR)
                    write_ulong(fh, SIZEOF iff8svx_vhdr)
                    IF write(fh, self.vhdr, SIZEOF iff8svx_vhdr) = FALSE
45                          Throw("file", fh)
                    ENDIF
                    write_ulong(fh, MAGIC_BODY)
                    write_ulong(fh, 0)          -> fixed in stopwrite
            ENDIF
50          self.pos := 0
    ENDPROC

    /*------------------------------------------------------------------------*/

55  PROC stopwrite() OF write8svx
            DEF fh
            IF fh := self._fh()
                    Seek(fh,  4, OFFSET_BEGINNING)      -> FORM.length
                    write_ulong(fh, 20 + SIZEOF iff8svx_vhdr + self.pos)
60                  Seek(fh, 20, OFFSET_BEGINNING)      -> VHDR.hisamples
                    write_ulong(fh, self.pos)
                    Seek(fh, 24 + SIZEOF iff8svx_vhdr, OFFSET_BEGINNING)
                    write_ulong(fh, self.pos)           -> BODY.length
            ENDIF
65  ENDPROC

    /*------------------------------------------------------------------------*/

    PROC process() OF write8svx
70          DEF smp
            SUPER self.process()
            smp := ! self._main() * 128.0 !
            IF self._fh() THEN write_sbyte(self._fh(), Bounds(smp, -128, 127))
            self.pos := self.pos + 1
75  ENDPROC
```

```
/*-------------------------------------------------------------------------+
| END: write8svx.e                                                         |
+=========================================================================*/
```

69 Source/write.e

```
/*=========================================================================+
| write.e                                                                  |
| Effect base class "write"                                                |
| Write to a file, set by parameter "file"                                 |
|                                                                          |
| WARNING: write.reset() causes existing output file to be wiped           |
|                                                                          |
| write.startwrite()    sort out header info etc, called when file changed  |
| write.stopwrite()     clean up, called when file changed                 |
|                       both the above return success, new versions should |
|                       call SUPER, both should handle fh being NIL        |
| write._fh()           get file handle                                    |
| write.setfile(name)   set file to name, calls stopwrite and startwrite   |
+-------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*in1out0', '*string', '*value'

/*-------------------------------------------------------------------------*/

EXPORT OBJECT write OF in1out0
PRIVATE
        fname : PTR TO CHAR     -> file name
        fh    : LONG            -> file handle
ENDOBJECT

PROC class()       OF write IS 'write'
PROC _fh()         OF write IS self.fh
PROC startwrite() OF write IS TRUE      -> write header structure
PROC stopwrite()  OF write IS TRUE      -> write header data (eg length)

/*-------------------------------------------------------------------------*/

PROC setfile(fname) OF write
        self.stopwrite()
        IF self.fh THEN Close(self.fh)
        self.fname := fname
        self.fh := IF fname THEN Open(fname, NEWFILE) ELSE NIL
        self.startwrite()
ENDPROC self.fh <> NIL

/*-------------------------------------------------------------------------*/

PROC new(list, name) OF write
        SUPER self.new(list, name)
        self.setfile(NIL)
ENDPROC

/*-------------------------------------------------------------------------*/
```

```
PROC end() OF write
        self.setfile(NIL)
ENDPROC SUPER self.end()

/*-----------------------------------------------------------------------*/

PROC param2id(str) OF write
ENDPROC IF strcmp(IDS_FILE, str) THEN ID_FILE ELSE SUPER self.param2id(str)

PROC id2param(id) OF write
ENDPROC IF id = ID_FILE THEN IDS_FILE ELSE SUPER self.id2param(id)

PROC paramtype(id) OF write
ENDPROC IF id = ID_FILE THEN TYPE_STRING ELSE SUPER self.paramtype(id)

PROC set(id, data) OF write
ENDPROC IF id = ID_FILE THEN self.setfile(data) ELSE SUPER self.set(id, data)

PROC get(id) OF write
ENDPROC IF id = ID_FILE THEN self.fname ELSE SUPER self.get(id)

/*-----------------------------------------------------------------------*/

PROC reset() OF write
        SUPER self.reset()
        self.setfile(self.fname)
ENDPROC

/*-----------------------------------------------------------------------+
| END: write.e                                                           |
+=======================================================================*/
```

70 Source/writeslab.e

```
/*=======================================================================+
| writeslab.e                                                            |
| Effect class "writeslab", write to a SLab sample file                  |
+-----------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*defs', '*file', '*slab_ff', '*string', '*value', '*write', 'dos/dos'

/*-----------------------------------------------------------------------*/

EXPORT OBJECT writeslab OF write
PRIVATE
        info : slab_info    -> data
        pos  : LONG         -> current sample position (starting at 0)
ENDOBJECT

PROC class() OF writeslab IS 'writeslab'

/*-----------------------------------------------------------------------*/

PROC new(list, name) OF writeslab
        SUPER self.new(list, name)
```

```
        self.info.rate := 44100.0
        self.info.bias := 0.0
        self.info.ampl := 0.0  -> changed in process to real max
ENDPROC

/*-----------------------------------------------------------------------*/

PROC startwrite() OF writeslab
        DEF fh
        IF fh := self._fh()
                write_ulong(fh, MAGIC_SLab)
                write_ulong(fh, 0)          -> fixed in stopwrite
                write_ulong(fh, MAGIC_Info)
                write_ulong(fh, 12)
                write_ulong(fh, 0)          -> fixed in stopwrite
                write_ulong(fh, 0)          -> fixed in stopwrite
                write_ulong(fh, 0)          -> fixed in stopwrite
                write_ulong(fh, MAGIC_Data)
                write_ulong(fh, 0)          -> fixed in stopwrite
        ENDIF
        self.pos := 0
ENDPROC

/*-----------------------------------------------------------------------*/

PROC stopwrite() OF writeslab
        DEF fh
        IF fh := self._fh()
                Seek(fh,  4, OFFSET_BEGINNING)
                write_ulong(fh, 24 + (self.pos * 4))
                Seek(fh, 16, OFFSET_BEGINNING)
                write_ulong(fh, self.info.rate)
                write_ulong(fh, ! self.info.bias / (self.pos !))
                write_ulong(fh, IF !self.info.ampl=0.0 THEN 1.0 ELSE self.info.↙
                    ↳ ampl)
                write_ulong(fh, MAGIC_Data)
                Seek(fh, 32, OFFSET_BEGINNING)
                write_ulong(fh, Mul(self.pos, 4))       -> normal * / is only 16↙
                    ↳  bit
        ENDIF
ENDPROC

/*-----------------------------------------------------------------------*/

PROC process() OF writeslab
        DEF smp
        smp := self._main()
        SUPER self.process()
        IF self._fh() THEN write_ulong(self._fh(), smp)
        self.info.bias := ! self.info.bias + smp
        IF ! self.info.ampl < Fabs(smp) THEN self.info.ampl := Fabs(smp)
        self.pos := self.pos + 1
ENDPROC

/*-----------------------------------------------------------------------+
| END: writeslab.e                                                       |
+=======================================================================*/
```

71 Source/zfilter.e

```
/*=============================================================================+
| zfilter.e                                                                    |
| Effect class "zfilter"                                                       |
| z-plane filters, see mathematical appendix                                   |
|                                                                              |
| Parameters:                                                                  |
| "poles"       0 <= number of poles  <= 31     \ limit due to number of       |
| "zeros"       0 <= number of zeros  <= 31     / unsigned bits in a LONG      |
| "poleXr"              0 <= pole X radius    <  1                             |
| "poleXf"      - rate / 2 <  pole X frequency <= rate / 2                     |
| "zeroXr"              0 <= zero X radius    <  1                             |
| "zeroXf"      - rate / 2 <  zero X frequency <= rate / 2                     |
+-----------------------------------------------------------------------------*/

OPT MODULE, PREPROCESS

MODULE '*cbuffer', '*defs', '*in1out1', '*link', '*string', '*value',
       '*debug'

/*-----------------------------------------------------------------------------*/

-> Static arrays make all objects large, but make implementation much simpler
EXPORT OBJECT zfilter OF in1out1
PRIVATE
        p       : LONG              -> poles (integer)
        z       : LONG              -> zeros (integer)
        pf[32]  : ARRAY OF LONG     -> pole freq     \ polar form, as set
        pr[32]  : ARRAY OF LONG     -> pole radius   |
        zf[32]  : ARRAY OF LONG     -> zero freq     |
        zr[32]  : ARRAY OF LONG     -> zero radius   /
        px[32]  : ARRAY OF LONG     -> pole x        \ cartesian form, for calc
        py[32]  : ARRAY OF LONG     -> pole y        |
        zx[32]  : ARRAY OF LONG     -> zero x        |
        zy[32]  : ARRAY OF LONG     -> zero y        /
        rx[32]  : ARRAY OF LONG     -> recursion x
        ry[32]  : ARRAY OF LONG     -> recursion y (ry[0] not used)
        x       : PTR TO cbuffer    -> previous input samples
        y       : PTR TO cbuffer    -> previous output samples
ENDOBJECT

PROC class() OF zfilter IS 'zfilter'

/*-----------------------------------------------------------------------------*/

PROC new(list, name) OF zfilter
        SUPER self.new(list, name)
        self.p := 0     -> empty filter, out = in
        self.z := 0
        -> arrays set before use, no need to clear
        NEW self.x.new(32.0)    -> maximum length
        NEW self.y.new(32.0)
ENDPROC

/*-----------------------------------------------------------------------------*/

```

```
PROC end() OF zfilter
        END self.y              -> delete sample buffers
        END self.x
ENDPROC SUPER self.end()

/*-----------------------------------------------------------------------*/

PROC reset() OF zfilter
        SUPER self.reset()
        assert(self.x, 'zfilter.reset.x')
        assert(self.y, 'zfilter.reset.y')
        self.x.clear()          -> clear sample buffers
        self.y.clear()
ENDPROC

/*-----------------------------------------------------------------------*/

PROC param2id(str : PTR TO CHAR) OF zfilter
        DEF id, len
        IF      strcmp(IDS_POLES, str); RETURN ID_POLES  -> these must be before..
        ELSEIF strcmp(IDS_ZEROS, str); RETURN ID_ZEROS
        ELSEIF strncmp(IDS_POLE, str, 4)                -> .. these
                -> poleXr, poleXf
                id, len := Val(str + 4)
                IF (len > 0) AND (0 < id) AND (id <= self.p)
                        IF      strcmp('r', str + 4 + len); RETURN ID_MULTI_POLE_R + id
                        ELSEIF strcmp('f', str + 4 + len); RETURN ID_MULTI_POLE_F + id
                        ENDIF
                ENDIF
        ELSEIF strncmp(IDS_ZERO, str, 4)
                -> zeroXr, zeroXf
                id, len := Val(str + 4)
                IF (len > 0) AND (0 < id) AND (id <= self.z)
                        IF      strcmp('r', str + 4 + len); RETURN ID_MULTI_ZERO_R + id
                        ELSEIF strcmp('f', str + 4 + len); RETURN ID_MULTI_ZERO_F + id
                        ENDIF
                ENDIF
        ENDIF
ENDPROC SUPER self.param2id(str)

/*-----------------------------------------------------------------------*/

PROC id2param(id) OF zfilter
        IF      (ID_MULTI_ZERO_R < id) AND (id <= (ID_MULTI_ZERO_R + self.z))
                RETURN StringF(String(8), 'zero\dr', id - ID_MULTI_ZERO_R)
        ELSEIF (ID_MULTI_ZERO_F < id) AND (id <= (ID_MULTI_ZERO_F + self.z))
                RETURN StringF(String(8), 'zero\df', id - ID_MULTI_ZERO_F)
        ELSEIF (ID_MULTI_POLE_R < id) AND (id <= (ID_MULTI_POLE_R + self.p))
                RETURN StringF(String(8), 'pole\dr', id - ID_MULTI_POLE_R)
        ELSEIF (ID_MULTI_POLE_F < id) AND (id <= (ID_MULTI_POLE_F + self.p))
                RETURN StringF(String(8), 'pole\df', id - ID_MULTI_POLE_F)
        ENDIF
```

```
ENDPROC SUPER self.id2param(id)

/*-------------------------------------------------------------------------*/

PROC paramtype(id) OF zfilter
        IF ((ID_MULTI_ZERO_R < id) AND (id <= (ID_MULTI_ZERO_R + self.z))) OR
           ((ID_MULTI_ZERO_F < id) AND (id <= (ID_MULTI_ZERO_F + self.z))) OR
           ((ID_MULTI_POLE_R < id) AND (id <= (ID_MULTI_POLE_R + self.p))) OR
           ((ID_MULTI_POLE_F < id) AND (id <= (ID_MULTI_POLE_F + self.p))) OR
           (id = ID_POLES) OR (id = ID_ZEROS)
                RETURN TYPE_NUMBER
        ENDIF
ENDPROC SUPER self.paramtype(id)

/*-------------------------------------------------------------------------*/

PROC set(id, data) OF zfilter
        IF     (ID_MULTI_ZERO_R < id) AND (id <= (ID_MULTI_ZERO_R + self.z))
                self.zr[id - 1 - ID_MULTI_ZERO_R] := data
                self.setrecalc()
        ELSEIF (ID_MULTI_ZERO_F < id) AND (id <= (ID_MULTI_ZERO_F + self.z))
                self.zf[id - 1 - ID_MULTI_ZERO_F] := data
                self.setrecalc()
        ELSEIF (ID_MULTI_POLE_R < id) AND (id <= (ID_MULTI_POLE_R + self.p))
                self.pr[id - 1 - ID_MULTI_POLE_R] := data
                self.setrecalc()
        ELSEIF (ID_MULTI_POLE_F < id) AND (id <= (ID_MULTI_POLE_F + self.p))
                self.pf[id - 1 - ID_MULTI_POLE_F] := data
                self.setrecalc()
        ELSEIF id = ID_POLES
                self.p := ! data !
                self.setrecalc()
        ELSEIF id = ID_ZEROS
                self.z := ! data !
                self.setrecalc()
        ELSE
                SUPER self.set(id, data)
        ENDIF
ENDPROC

/*-------------------------------------------------------------------------*/

PROC get(id) OF zfilter
        IF     (ID_MULTI_ZERO_R < id) AND (id <= (ID_MULTI_ZERO_R + self.z))
                RETURN self.zr[id - 1 - ID_MULTI_ZERO_R]
        ELSEIF (ID_MULTI_ZERO_F < id) AND (id <= (ID_MULTI_ZERO_F + self.z))
                RETURN self.zf[id - 1 - ID_MULTI_ZERO_F]
        ELSEIF (ID_MULTI_POLE_R < id) AND (id <= (ID_MULTI_POLE_R + self.p))
                RETURN self.pr[id - 1 - ID_MULTI_POLE_R]
        ELSEIF (ID_MULTI_POLE_F < id) AND (id <= (ID_MULTI_POLE_F + self.p))
                RETURN self.pf[id - 1 - ID_MULTI_POLE_F]
        ELSEIF id = ID_POLES
                RETURN self.p !
        ELSEIF id = ID_ZEROS
                RETURN self.z !
        ENDIF
ENDPROC SUPER self.get(id)
```

```
165
    /*------------------------------------------------------------------------*/

    PROC process() OF zfilter
            DEF out, i
170         SUPER self.process()
            -> move to next in buffer
            self.x.next()
            self.y.next()
            self.x.write(self._main())
175         -> calculate
            out := 0.0
            IF self.z
                    FOR i := 0 TO self.z
                            out := ! out + (! self.rx[i] * self.x.readrel(i - self.z↙
                                ↳ !))
180                 ENDFOR
            ENDIF
            IF self.p
                    FOR i := 1 TO self.p
                            out := ! out - (! self.ry[i] * self.y.readrel(i - self.p↙
                                ↳ !))
185                 ENDFOR
                    out := ! out / self.ry[0]
            ENDIF
            -> output
            self.y.write(out)
190         self.output(ID_MAIN, out)
    ENDPROC

    /*------------------------------------------------------------------------*/

195 PROC recalc() OF zfilter
            DEF i, link : PTR TO link, t
            SUPER self.recalc()
            link := self.getinput(ID_MAIN)
            t := ! PI2 / link.rate
200         FOR i := 0 TO self.p - 1        -> sort out poles
                    self.px[i] := ! self.pr[i] * Fcos(! self.pf[i] * t)
                    self.py[i] := ! self.pr[i] * Fsin(! self.pf[i] * t)
            ENDFOR
            FOR i := 0 TO self.z - 1        -> sort out zeros
205                 self.zx[i] := ! self.zr[i] * Fcos(! self.zf[i] * t)
                    self.zy[i] := ! self.zr[i] * Fsin(! self.zf[i] * t)
            ENDFOR
            -> make recursion formulae
            IF self.p THEN make_r(self.p, self.px, self.py, self.ry)
210         IF self.z THEN make_r(self.z, self.zx, self.zy, self.rx)
            -> set delays
            self.x.setlength(self.z !)
            self.y.setlength(self.p !)

215         -> debugging code (dumps all relevant state info)
    #ifdef DEBUG
            PrintF(DEBUG'zfilter.recalc zeros = \d\n', self.z)
            PrintF(DEBUG'\ti\tzr\tzf\tzx\tzy\n')
            FOR i := 0 TO self.z - 1 DO PrintF(DEBUG'\t\d\t\s\t\s\t\s\t\s\t\s\n', i,
```

```
                realf(self.zr[i]), realf(self.zf[i]),
                realf(self.zx[i]), realf(self.zy[i]))
        PrintF(DEBUG'zfilter.recalc poles = \d\n', self.p)
        PrintF(DEBUG'\ti\tpr\tpf\tpx\tpy\n')
        FOR i := 0 TO self.p - 1 DO PrintF(DEBUG'\t\d\t\s\t\s\t\s\t\s\t\s\n', i,
                realf(self.pr[i]), realf(self.pf[i]),
                realf(self.px[i]), realf(self.py[i]))
        PrintF(DEBUG'zfilter.recalc recurse\n')
        PrintF(DEBUG'\ti\trx\try\n')
        FOR i := 0 TO IF self.z > self.p THEN self.z ELSE self.p
            PrintF(DEBUG'\t\d\t\s\t\s\n', i,
                    IF i <= self.z THEN realf(self.rx[i]) ELSE '-',
                    IF i <= self.p THEN realf(self.ry[i]) ELSE '-')
        ENDFOR
#endif
ENDPROC

/*-----------------------------------------------------------------------*/

-> Multiply out a set of linear factors, assuming result is real polynomial
-> Polynomial is d[0]z^n + d[1]z^(n-1) + ... + d[n]
-> This function is a candidate for assembly language optimisation, as the
-> number of loops is high (O(factors * 2 ^ factors)) and only simple
-> maths is done (no large function calls)
-> fx, fy are length factors, d is length factors + 1
PROC make_r(factors, fx : PTR TO LONG, fy : PTR TO LONG, d : PTR TO LONG)
        DEF i : REG, j : REG, k : REG, l : REG, m : REG, -> put ints in ↙
            ↳ registers
            x, y, xx, yy, a, b
        assert(factors, 'zfilter.make_r.factors')
        assert(fx,      'zfilter.make_r.fx')
        assert(fy,      'zfilter.make_r.fy')
        assert(d,       'zfilter.make_r.d')
    FOR i := 0 TO factors DO d[i] := 0.0    -> clear
        FOR i := 0 TO Shl(1, factors) - 1       -> all permutations (large)
            k := i                                  -> copy of i to modify
            m := 0                                  -> number of numbers ↙
                ↳ multiplied
            x := 1.0                                -> multiply => start at 1
            y := 0.0
            FOR j := 0 TO factors - 1
                -> extract bits in order
                l, k := Mod(k, 2)  -> l = multiply by a number (1) or z ↙
                    ↳ (0)
                -> k := k / 2      -> truncated integer division in Mod
                m := m + l
                IF l = 1
                    a := ! -fx[j]       -> multiply (x,y) by (fx,fy)↙
                        ↳ [j]
                    b := ! -fy[j]
                    xx := ! (! a * x) - (! b * y)
                    yy := ! (! a * y) + (! b * x)
                    x := xx
                    y := yy
                -> ELSE                 -> multiply by "z"
                ENDIF
            ENDFOR
```

```
                d[m] := ! d[m] + x        -> add to polynomial coefficient
        ENDFOR
ENDPROC

/*-------------------------------------------------------------------------+
| END: zfilter.e                                                           |
+=========================================================================*/
```