Axoris Group AL3101 Application Generator
Users Guide & Writing Scripts

G. Soyez

1 Introduction

The command line Application Generator (codename *AgALag*) takes as argument a project script describing the structure to build. We describe here what kind of structure we are able to generate and how to write project scripts.

2 Description

Given a structure to build, the application generator collect the source code for the requested modules and merge it into a single assembler file. If the project script is called `project.axp`, the output assembler code will be `project.asm`.

3 Structures

Inside the word *structure*, we tried to make an abstract representation as close as possible to the one you have when you plug sound effects in serie or in parallel. Basically, this mean implementing two kinds of objects

- *modules*, corresponding to some effect, and
- *links* between them.

A structure is therefore a succession of modules, associated in serie or parallel with links, starting from an input module and going to an output module.

4 Scripts file format

4.1 General structure

The scripts are expected to have the extension `.axp`. They should the following general structure

```
<Number of modules>
1. <module 1>
2. <module 2>
...
n. <module n>
<Number of links>
1. <link 1>
2. <link 2>
...
m. <link m>
```

We will describe in details the `module` and `link` entries in the next sections.

4.2 Module list

After giving the number of modules, you need to give to list of modules you use. Each line of this list describes a module and has the following syntax

```
i.␣ModuleName
```

where `i` is the module index, starting from 1, and `ModuleName` is

- the module filename with its path.
- `IN` or `OUT` for the input and output modules.

Note that, **every** structure **must** contain an input **and** an output module!

4.3 Links list

Once the list of modules finished, you give the number of links and the link list. Each link connect one output channel of a module to an input channel of another module. Each line of the link list describes a link and has the syntax[1]

```
i.␣Out(ChO)␣->␣In(ChI)
```

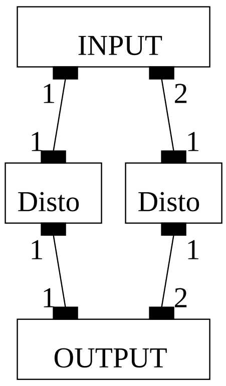
where `i` is the link index, starting from 1. This describes a link starting from channel `ChO` of module `Out` and going to channel `ChI` of module `In`.

[1]Pay attention to spaces.

5 Examples

5.1 Example 1: Stereo Distorsion

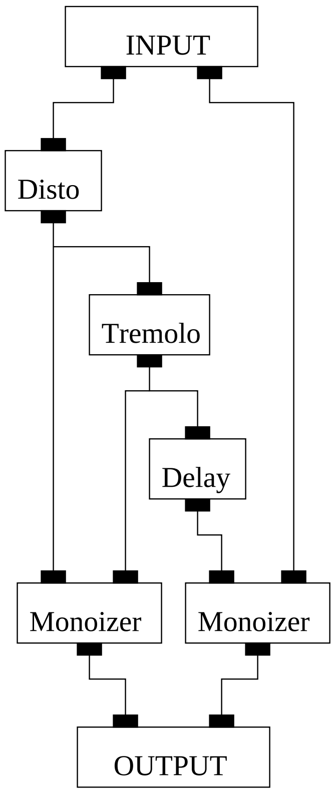
Having some mono distorsion, we want to construct a stereo distorsion. This correspond to the following scheme



```
4
1. IN
2. ../effects/disto.asm
3. ../effects/disto.asm
4. OUT
4
1. 1(1) -> 2(1)
2. 1(2) -> 3(1)
3. 2(1) -> 4(1)
4. 3(1) -> 4(2)
```

5.2 Example 2: A little bit harder ...

This example is a little bit more complicated. Don't know if it can produce a good sound, ... but let's produce a good script!



```
7
1. IN
2. OUT
3. ../effects/delay.asm
4. ../effects/disto.asm
5. ../effects/tremolo.asm
6. ../effects/monoizer.asm
7. ../effects/monoizer.asm
10
1. 1(1) -> 4(1)
2. 4(1) -> 5(1)
3. 4(1) -> 6(1)
4. 4(1) -> 6(1)
5. 5(1) -> 6(2)
6. 5(1) -> 3(1)
7. 6(1) -> 2(1)
8. 3(1) -> 7(1)
9. 1(2) -> 7(2)
10. 7(1) -> 2(2)
```

6 Limits

The main limit of our application generator is the fact that we don't allow loops i.e. links corresponding to

```
Module1 -> Module2 -> Module3 -> Module1
```

will return an error message.

For future things to do, have a look at the TODO file.

7 Contact

You can send bugs or reports to

Gregor.Soyez@swing.be

Contents