Appendix 7 Alecsis library manager -- alm

While speaking about Alecsis we mentioned forming of libraries. Alecsis supports use of libraries, and we recommend it. All function calls, as well as calls of modules, model cards, and other object can be performed from previously formed libraries. Library file can store the following objects: global data (**`data`**), functions (**`funct`**), model cards (**`model`**), and modules (**`module`**). The user can define data, functions, model cards, and modules, than compile them and create a library. This also means that you can create a commercial library containing, for example models of some widely used logic circuits in a particular technology, than compile it, and distribute it without the source code.

If a user works with multiple libraries concerning a particular field of simulation, it is not very efficient for him or her to always list the libraries to be searched during the linking process. It is also not practical to expect the user to remember which module is in which library. It would be useful if a user could create the most suitable library by freely manipulating the elements of existing libraries (these are: listing the content, copying of elements, erasing, moving, listing, and similar operations with the elements of a library).

A7.1. Handling of libraries in Alecsis

Let us say that the file named `file.ac` contains something we wish to compile and store in a library file. We can obtain that library file by calling Alecis using option '`-c`', which points out the file to be compiled but not linked.

```
alec -c file.ac
```

The result of this command is the creation of a library file named `file.ao`.

Library content can be listed using the option '`-a`' while giving the full name of the library file (including the extension '`.ao`')

```
alec -a file.ao
```

In case `root module` is in the file `file.ac`, we will receive a warning and this module will not be compiled into the library file. The `root module` can be compiled and linked only during a simulation, since it cannot be invoked from another `module`.

We can use command `library` (can be anywhere in the simulation file) to instruct compiler which libraries to search during the linking procedure.

```
library file, other_library_1, other_library_2;
```

The other possibility to transfer the information to linker is to list the libraries at the command line using option '`-l`':

```
alec -lfile -lother_library_1 -lother_library_2 test_file.ac
```

With the command `library` you can list the whole absolute or relative path in quotations to the library file. To avoid this you define the path to directory where Alecsis should search for libraries by setting the environment variable `ALEC_LIB_PATH`, or you can list these directories from the command line using option '`-L`', as described in *Appendix 1*.

You can see that program Alecsis does not have extensive capabilities in handling libraries. These capabilities are limited to listing of libraries, which is not enough when dealing with complex library files. For manipulating libraries, you can use program for library processing named **`alm`** (Alecsis Library Manager). We will discuss this program in the following sections.

A7.2. Capabilities of alm

`alm` should provide easy manipulation with the elements of libraries. The program works interactively, or it can execute a series of commands from a batch file. When started interactively program gives a prompt:

```
1:alm ->
```

At this moment the user types the commands, and after every command the command counter increases by one. When the user opens a library, the name of the library its name is written in the prompt instead the name of the program `alm`. Command **`open`** opens a desired number of libraries, as many as the operating system can handle. The current library is called the **foreground library**. All other libraries are **background libraries**. A newly opened library becomes the foreground library, except if the changes to the previous foreground library are not saved. The change of status of a library command is performed using:

```
fg lib_name
```

This command makes the previously opened library `lib_name` the foreground library, and the library is ready to be processed. However, if there are changes in the content of the previous foreground library, the user, before the change of the foreground library has to save the changes, or withdraw from the changes. This points out the program organization: **changes can be applied to one library at a time**, only. The other libraries can be listed, read, but the changes are possible only with the foreground library. All changes are stored in the memory, and the library does not change really until saving of the changed library is requested.

Command **`ll`** gives the list of libraries that are open.

Command **`ls`** lists the content of a foreground or explicitly named library. Command **`ls`** with the option **`-l`** gives detailed information about the content of a library.

Command **`h`** gives a short description of all `alm` commands.

Command **`close`** closes the library. You cannot close the library whose content is copied into the foreground library, until saving the new content of the foreground library, or giving up the changes.

Command **`alias`** defines alias (abbreviations) for more complex library names, or strings of commands. For example, command:

```
alias ttl /users/hybrid/alec/exams/lib/ttl.ao
```

enables use of shorter name of the absolute path to the library `ttl.ao` in the directory `/users/hybrid/alec/exams/lib`. This command can define aliases for other `alm` commands, too. For example:

```
alias dir ls -l
```

If your command has more than one option, you can list these options separately in any order, or you can combine them. All option arguments begin with a hyphen '-'. The following commands are equivalent:

```
ls -l -n12 lib_name.ao
ls -ln12 lib_name.ao
ls -n12 -l lib_name.ao
```

Interactive approach is satisfactory for the largest number of applications. There are, however, applications when a repeated or a packet processing of commands is needed. One application is with the copyingof libraries in parts, when some parts of libraries, depending upon the user, should be installed in a resulting library, and some should not. In this case you can call a program to execute commands from a file. However, this way of work is not flexible, since the user cannot take the appropriate action in case of an illegal command, that is the program stops working giving out the error message. To overcome this obstacle you can use option '`-v`' that prints the current command on the screen, so you can follow the execution.

You can invoke UNIX commands from `alm`. Everything following the exclamation sign '`!`' will be passed to the operating system. This can be used for an overview of the directory content, copying a file, change of the status of a library, etc.

Notice that `alm` honours the rights of access imposed by the operating system. This means that a user cannot change a library file if he or she has the read-only authorization. If the user wants to change such library, he or she can save it under a different name, or change its status using appropriate commands of the operating system.

Program `alm` occupies less than 100Kb of RAM memory, without dynamically allocated memory for data storage when working with libraries. The program is in C according to the old definition by B.W. Kernighan and D.M. Ritchie, and is made for UNIX environment.

A7.3. Command line options

Program is invoked using command `alm`.

Syntax:

```
alm [-i] [-v] [filename]
```

Remarks:

When `alm` uses commands from batch file, it stops working whenever it reaches an instruction requiring interaction with the user, and gives an error report.

Options:

`-i` starts work from the named batch file. `alm` reads, and executes line after line. Every line beginning with '`#`' is a non executable command (comment).

`-v` when working from a batch file, it writes out the current command.

A7.4. alm commands

A7.4.1. List of alm commands

alias	creating aliases
close	closes a library file
cp	copying the elements into the foreground library
fg	sets the foreground library
h	prints the list of commands
ll	printing list of open libraries
ls	lists content of the named library
new	creates a new library file of the given name
open	opens a library file
q	ends the program
rep	moving elements inside the foreground library
rm	deleting elements from the foreground library
touch	updating the last modification date
w	saving a library
!	call of a shell command

A7.4.2. Overview of alm commands

alias

Effect:

Creates an alias.

Syntax:

```
alias [new_name] [old_name]
```

Remarks:

Command without arguments shows the already defined abbreviations. All abbreviations can be redefined. It is not possible to create an abbreviation for the command `alias`.

close

Effect:

This command closes a previously opened library.

Syntax:

```
close lib_name[.ao]
```

Remarks:

The extension '`.ao`' is not necessary in the name of the library. It is not possible to close a library whose elements are copied to the foreground library before recording the foreground library. When you close the foreground library, the first opened library becomes the foreground library.

`cp` (copy)

Effect:

Copies elements from a named library to the foreground library.

Syntax:

cp [-n*num*] [-t] lib_name[.ao]/element_name

Remarks:

It is legal to use meta-characters '`*`' and '`?`' with the elements to be copied (see command `touch`). The extension '`.ao`' is not necessary in the library name. It is not legal for two elements to exist in the same library under the same name, so the elements that already exist in the foreground library are not copied.

Options:

`-n`*num*	*num* is a positive integer. The first copied element comes to position *num*; the second on the position *num*+1 and so on.
`-t`	default option, copy to the end of the library.

`fg` (foreground)

Effect:

Previously opened library, whose name is the argument of the command, becomes the foreground library. The modifications can be performed in the foreground library only.

Syntax:

```
fg lib_name[.ao]
```

Remarks:

The extension '`.ao`' is not necessary in the name of the library. If the changes to the foreground library are not saved, you cannot declare another library the foreground library. The program will offer you to save the changes to the foreground library, give up the changes to the foreground library, or give up the whole action.

`h` (help)

Effect:

This command gives a list of `alm` commands along with the syntax.

Syntax:

```
h
```

`ll`

Effect:

Gives the list of all open libraries.

Syntax:

```
ll
```

`ls`

Effect:

Shows the content of a library.

Syntax:

`ls [-l] [-n`*`num`*`] [-s] [lib_name[.ao]] [element_name]`

Remarks:

The extension '`.ao`' is not necessary in the name of the library. If the name of the library `lib_name` is not given, default is the foreground library. You can use meta-characters (see command `touch`) in `lib_name` i `element_name`.

Options:

`-l`	gives additional information on elements. Class of element is given beside the name of the element (`module`, `model`, `data`, `funct`), as well as the position of the element, the length, and the date of the last change.

`-n`*num*	*num* is a positive integer. This option defines the number of lines printed on the screen as *num*, when the system waits for the user to press *return* for the printout to continue.
`-s`	goes along with setting of the parameter `element_name`. This option lists only the element with the name `element_name`, if it exists in the library. The option is useful if the library has a large number of elements, and you want to find the one with a particular name.

Examples:

`ls`	lists all elements of the foreground library
`ls lib_name`	lists all elements of the library `lib_name` (it has to be open)
`ls -s inv_model`	checks if the foreground library has the element `inv_model`
`ls -s inv*`	lists all elements of the foreground library whose name begins with `inv`

`new`

Effect:

Creates a new library file that initially does not contain any objects. You can write desired content into it by copying from other libraries.

Syntax:

```
new lib_name[.ao]
```

Remarks:

The extension '`.ao`' is not necessary in the name of the library. A newly opened library becomes the foreground library, except if the changes to the previous foreground library are not saved.

`open`

Effect:

This command opens the library and reads the heading into `alm`. This procedure allows the access to the elements of the library (`alm` cannot access a library that is not open).

Syntax:

```
open lib_name[.ao]
```

Remarks:

The extension '`.ao`' is not necessary in the name of the library. A newly opened library becomes the foreground library, except if the changes to the previous foreground library are not saved.

`q` (quit)

Effect:

Exit from `alm`.

Syntax:

```
q[!]
```

Remarks:

If there are changes to the foreground library, the user can save the changes, leave without changes or cancel the action. The special form of the command '`q!`' lets the user leave the program without any checks.

`rep` (replace)

Effect:

Shifts an element within the foreground library.

Syntax:

```
rep sourcenum destnum
```

Remarks:

You give the order number of the element you wish to shift - `sourcenum`. The order number the element will obtain in the library is `destnum`. The elements with the number `destnum`, and all other numbers with the order number larger than `destnum` increase their number by one. If `destnum` is larger than the total number of elements in the library, the destination is the end of the library.

`rm` (remove)

Effect:

Deletes an element from the foreground library.

Syntax:

```
rm element_name
```

Remarks:

It is legal to use meta-characters '`*`' and '`?`' with the elements to be deleted (see command `touch`).

`touch`

Effect:

The command records a new modification date for the elements it is applied to.

Syntax:

```
touch element_name
```

Remarks:

It is legal to use meta-characters '`*`' and '`?`' with parameters `element_name`. Character '`?`' replaces any one character, and '`*`' replaces any group of characters (including an empty string).

Example:

```
touch inv?a*
```

This is: apply the date to all elements whose name starts with `inv`, than there is any character, then `a`, and than anything till the end of the name. For example, names satisfying this criteria are `invqa`, `invaanam`, `inv1a_model`, `inv3a`, `inv_a_____32`.

`w` (write)

Effect:

Records the foreground library under the new or existing name.

Syntax:

```
w [lib_name[.ao]]
```

Remarks:

The extension '`.ao`' is not necessary in the name of the library. If no name is given the library is saved under the present name, with the loss of the previous content. Otherwise, the library is saved under the new name, while the old library stays unchanged. In that case, the new library becomes the foreground library, and the old library is erased from the list of open libraries.

!

Effect:

Calls a command of the operative system, i.e. passes the string to the UNIX shell for execution.

Syntax:

```
!string
```

Examples:

```
!ls
!mkdir new_lib
```