# Modules_`mkiv`

- I started to compile pdf modules for documentation around the end of 2006
- Original motivation was to learn something more about programming in ConTEXt and discover hidden features due lack of documentation
- Also a way to check systematically the new releases of ConTEXt_`mkii`, which at that time was already stable
- It was never intended as a complete test-bed: most the time the patches were not critical

## With ConTEXt-`mkiv` things changed a lot:

- new kind of code to document, the `Lua` modules
- `mkiv` was and still is 'a moving fast target': generating documentation harder than before because now patches can be useless or dangerous (the code already changed enough to reject the patch)
- moving so fast also makes a little significance for informational purposes.

From couple of years `mkiv` has a `current` release which is updated less frequently (actually about 3 or 4 times in a year) and hence Modules is more appropriate for `current` than for `beta`.

I believe that Modules is not a vital piece for documentation purpose: to understand a piece of code now it's necessary to see together both the `lua` and the `mkiv` module and two disjoint pdfs are not practical.

Anyway, thank to Fabrice Popineau and Taco actually Modules_`mkiv` has a home at Supelec, at `http://foundry.supelec.fr/gf/project/modules/` that I update at least once at year.

Both Modules_`mkii` and Modules_`mkiv` were generated by shell (`Bash`) script:
for this meeting I've decided to replace this script with LuaTEX code to drop the necessity of a Bash interpreter.

## The idea is very simple: save this file as `modules-mkiv.tex`

```
\starttext
\startmode[*first]
\ctxlua{document.lscarso.modules.setup()}
\ctxlua{document.lscarso.modules.mkiv()}
\ctxlua{document.lscarso.modules.mkiv_savedata()}
\stopmode
\ctxlua{document.lscarso.modules.report =
   document.lscarso.modules.mkiv_report;
   dofile("report.txt")}
\stoptext
```

## and compile it

```
#>context modules-mkiv
```

The idea is as follow: we parse the file `context.mkiv` to extract the name of module to compile from a line that match `\loadcorefile` or `\loadmarkfile` and then we process it in batch-mode; we can decide if it was correctly generated by analyzing the result of compilation. Also note that we respect the order of `context.mkiv` not the alphabetical one.

The function

```
function document.lscarso.modules.mkiv()
```

does exactly this: it generates all pdfs and put each of them under the folder

```
pdf/<filename>/<filename>-mkiv.pdf
```

## The function

`function document.lscarso.modules.mkiv_savedata()`
saves all the results into the `report.txt` file in a way that they are callable by the lua function
`document.lscarso.modules.report`
which by default does nothing.

## The meaning of

```
\ctxlua{document.lscarso.modules.report =
   document.lscarso.modules.mkiv_report;
   dofile("report.txt")}
```
seen before is to replace the default function with our function and than evaluate it on `report.txt`

## report.txt:

```
document.lscarso.modules.report{["done"]={
"bibl-bib.mkiv","strc-mat.mkiv",...
"symb-ini.mkiv",},
["errors"]={"math-int.mkiv","tabl-ntb.mkiv",
"tabl-tab.mkiv","math-ali.mkiv",
"font-ini.mkiv","buff-ver.mkiv","typo-ini.mkiv",}}
```

Using LuaTEX as a data description language.

## The function

`document.lscarso.modules.mkiv_report(data)`
is pretty simple: it typesets a document with the subject **mkiv done** that contains a tabular with the names all pdfs that are OK, and the subject **errors** with those ones that are wrongs.

With the ConTEXt-`mkiv` version `2010.07.22 19:04` there are `199` files correctly generated and `7` with errors; the total runtime was 2681.937 seconds on my laptop.

## Conclusion

- It was easy to translate a `Bash` script in ConTEXt-`mkiv` but probably the most important thing is the data description side of `Lua`, which is a powerful mechanism to customize the elaboration of data.
- The next step is the management of errors. Quite often the errors are typo and hence the relative patch is trivial; the Unix program `diff` can quickly produce a patch file and it should be interesting to implement in LuaTEX (and perhaps then the `patch` program too).

- Creating a pdf to index each pdf of Modules should be an easy task: originally Hans did something like this for `mkii` but the resulting pdf is too much heavy because contains all pdfs. The idea is hence a light pdf with only references to external pdfs.
- Updating the svn repository is a manual job but I see no valid reason to investigate on a LuaTEX implementation, apart an os system command that relies on a local svn client.
- Finally an eventually integration with the test suite from Hans is doable but not mandatory and anyway the test files from Hans should take the precedence over the Modules files.

That's all

Thank you !