Modules_mkiv

Abstract

Modules_mkiv is a way to compile the documentation of all ConTEXt-mkiv modules using only lua and the minimals. Can also be view as a partial test-bed for ConTEXt-mkiv, eventually to be integrated with Hans's test files.

Introduction

I started to compile pdf modules for documentation around the end of 2006. My original motivation was to learn something more about programming in ConTEXt and discover hidden features due lack of documentation, but during the time I cultivated the idea of a monolithic pdf as a container of all pdfs (but it was too much heavy to generate and also to read) and subsequently a pdf as index of all external modules, which I still consider a good solution.

In parallel with this, Modules was also a way to systematically check the new releases of ConTEXt_mkii, which at that time was already stable, but it was never intended as a complete test-bed simply because most the time the TEX code for documentation is simple: the check had some sense because Hans usually add to documentation of the new feature a simple example and so it was also possible to verify the new feature versus the rest of the documentation.

But most the time the example is not a border case or part of a unit-test so the patches sent back for corrections were not critical, hence it was normally that the same error was corrected after some releases. In the end the main meaning of modules_mkii was, and still is, documentation.

With ConTEXt-mkiv things changed a lot: first because we have now a new kind of code to document, the Lua modules and second because mkiv was and still is 'a moving fast target', i.e new beta releases happen very often: this makes generating documentation harder than before because now patches can be useless or dangerous (the code already changed enough to reject the patch) and also make a little sense for informative 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

But in spite of this I believe that is not an important 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 here.

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 a shell (Bash) script, while updating of the svn repository is done manually; for this meeting I've decided to replace the script with LuaTEX code to drop the necessity of a Bash interpreter.

Generating Modules_mkiv

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 first run generates all pdfs and put each of them under the folder

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

of the current directory. At the end `modules-mkiv.pdf` contains a simple list of pdf correctly generated or not; the file `report.txt` is a lua file that should be not deleted, because it used by `modules-mkiv.tex`.

The code is really simple: after a short section for data structures

```
document.lscarso = document.lscarso or {}
document.lscarso.modules =  {}
document.lscarso.modules.texsprint = tex.sprint
document.lscarso.modules.ctxcatcodes = tex.ctxcatcodes
function document.lscarso.modules.report(data) end -- stub
```

there is a function to initialize the data with the correct directories and file types (actually `mkii` and `lua` are not used)

```
function document.lscarso.modules.setup()
  local texmf = kpse.expand_path("$TEXMFCONTEXT") --$ for emacs...
  local base = texmf .. "/tex/context/base"
  local mkiv = base .. "/*.mkiv"
  local mkii = base .. "/*.mkii"
  local lua  = base .. "/*.lua"
  local ctxmkiv =  base .. "/context.mkiv"
  document.lscarso.modules['texmf']= texmf
  document.lscarso.modules['base'] = base
  document.lscarso.modules['mkivs'] = mkiv
  document.lscarso.modules['mkiis']= mkii
  document.lscarso.modules['lua']  = lua
  document.lscarso.modules['ctxmkiv']  =  ctxmkiv
  local g = {}
  dir.glob(mkiv,g)
  document.lscarso.modules['table-mkiv']  =  g
  document.lscarso.modules['outdir']  =  "pdf"
  dir.mkdirs(document.lscarso.modules['outdir'])
end
```

The idea is as follow: we parse the file `context.mkiv` to extract the file 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.

```
function document.lscarso.modules.mkiv()
  local setup = document.lscarso.modules or  {}
  local ctxmkiv = setup["ctxmkiv"]
  local base = setup["base"]
  local target = {}
  local m
  for v in io.lines(ctxmkiv) do
   m = string.match(v,"\\loadcorefile{([^}]+)}") or  string.match(v,"\\loadmarkfile{([^}]+)}")
   if m ~= nil then
     target[base .. "/" .. m ] = true
   end
  end
  local found = {}
  local g = document.lscarso.modules['table-mkiv']
  for i,v in ipairs(g) do
    local vv = string.gsub(v,'.mkiv','')
```

```
  if target[vv] == true  then found[#found+1]= vv end
end
local cwd = lfs.currentdir()
local errors = {}
local done = {}
for i = 1,#found do
  local res
  local target = file.basename(found[i])
  local outdir = document.lscarso.modules['outdir'] .. "/" .. target
  dir.mkdirs(outdir)
  lfs.chdir(outdir)
  --
  -- In local dir now
  --
  file.copy(found[i] .. ".mkiv", target .. '.mkiv')
  res = os.execute(string.format("mtxrun --script modules --convert \"\%s.mkiv\"",target))
  if res == 0 then
    res = os.execute(
        string.format("context --batchmode  --usemodule=mod-01 \"\%s-mkiv.ted\"",target))
    if res == 0 then
      os.remove( target .. ".mkiv")
      os.execute("context --purgeall")
      done[#done+1] = target .. '.mkiv'
    else
      errors[#errors+1] = target .. '.mkiv'
    end
  else
   errors[#errors+1] = target .. '.mkiv'
  end
  lfs.chdir(cwd)
  local attr = lfs.attributes (found[i] .. '.tex')
  if not(attr == nil) then
   file.copy(found[i] .. ".tex", target .. '.tex')
   res = os.execute(string.format("mtxrun --script modules --convert \"\%s.tex\"",target))
   if res == 0 then
     res = os.execute(
          string.format("context --batchmode  --usemodule=mod-01 \"\%s-tex.ted\"",target))
     if res == 0 then
       os.remove( target .. ".tex")
       os.execute("context --purgeall")
       done[#done+1] = target .. '.tex'
```

```
      else
        errors[#errors+1] = target .. '.tex'
      end
    else
     errors[#errors+1] = target .. '.tex'
    end
    lfs.chdir(cwd)
   end
  end
  document.lscarso.modules.done = done
  document.lscarso.modules.errors = errors
  lfs.chdir(cwd)
end
```

This function saves all the results into the `report.txt` file in a way that they are callable the lua function `document.lscarso.modules.report`:

```
function document.lscarso.modules.mkiv_savedata()
 local done = document.lscarso.modules.done
 local errors = document.lscarso.modules.errors
 local report = "document.lscarso.modules.report{"
 if #done > 0 then
  report = report .. '["done"]={'
  for i,v in ipairs(done) do
    report = report .. '"' .. v .. '",'
  end
  report = report .. "}"
 end
 if #errors > 0 then
  report = report .. ',["errors"]={'
  for i,v in ipairs(errors) do
    report = report .. '"' .. v .. '",'
  end
  report = report .. "}"
 end
 report = report .. "}"
 local fd = io.open("report.txt","w")
 fd:write(tostring(report))
 fd:close()
end
```

In the end the function `function document.lscarso.modules.mkiv_report(data)` replace `document.lscarso.modules.report` to analyze the report: this is the meaning of

```
\ctxlua{document.lscarso.modules.report =
   document.lscarso.modules.mkiv_report; dofile("report.txt")}
```

seen before.

The function is pretty simple: the subject **mkiv done** contains a tabular of all pdf that are OK, the subject **errors** that one with errors.

```
function document.lscarso.modules.mkiv_report(data)
 local done = data["done"]
 local errors = data["errors"]
 if not(done==nil) and #done > 0 then
   texprint = document.lscarso.modules.texsprint
   ctxcatcodes = document.lscarso.modules.ctxcatcodes
   texprint(ctxcatcodes,"\\subject{mkiv done}")
   texprint(ctxcatcodes,"\\starttabulate[|l|l|l|l|l|]")
   local c= 1
   for i,v in ipairs(done) do
     texprint(ctxcatcodes,string.format("\\NC \%s \\NC",v))
     if (c > 0 and math.mod(c,5)) == 0 then
       texprint(ctxcatcodes,"\\NR")
     end
     c= c+1
   end
   texprint(ctxcatcodes,"\\stoptabulate")
  end
  if not(errors==nil) and #errors > 0 then
   texprint = document.lscarso.modules.texsprint
   ctxcatcodes = document.lscarso.modules.ctxcatcodes
   texprint(ctxcatcodes,"\\subject{errors}")
   texprint(ctxcatcodes,"\\starttabulate[|l|l|l|l|l|]")
   local c= 1
   for i,v in ipairs(errors) do
     texprint(ctxcatcodes,string.format("\\NC \%s \\NC",v))
     if (c > 0 and math.mod(c,5)) == 0 then
       texprint(ctxcatcodes,"\\NR")
     end
     c=c+1
   end
```

```
   texprint(ctxcatcodes,"\\stoptabulate")
  end
end
```

Here is the results with the ConTEXt-`mkiv` version `2010.07.22 19:04`: there are 199 files correctly generated and 7 with errors.

1

mkiv done

bibl-bib.mkiv	strc-mat.mkiv	tabl-nte.mkiv	syst-mes.mkiv	prop-lay.mkiv
typo-dig.mkiv	spac-grd.mkiv	type-ini.mkiv	syst-rtp.mkiv	attr-div.mkiv
lang-mis.mkiv	typo-brk.mkiv	lpdf-ini.mkiv	spac-hor.mkiv	core-ini.mkiv
syst-aux.mkiv	page-par.mkiv	strc-tag.mkiv	strc-sbe.mkiv	trac-deb.mkiv
page-flw.mkiv	mlib-pdf.mkiv	node-rul.mkiv	scrn-hlp.mkiv	colo-ini.mkiv
page-ins.mkiv	pack-box.mkiv	core-fil.mkiv	node-mig.mkiv	strc-doc.mkiv
page-lin.mkiv	page-spr.mkiv	grph-inc.mkiv	supp-ran.mkiv	core-mis.mkiv
core-con.mkiv	mlib-ctx.mkiv	lang-url.mkiv	core-job.mkiv	supp-mat.mkiv
trac-vis.mkiv	spac-fnt.mkiv	char-act.mkiv	syst-fnt.mkiv	font-col.mkiv
buff-ini.mkiv	syst-str.mkiv	luat-bas.mkiv	page-sid.mkiv	page-lay.mkiv
page-mak.mkiv	typo-krn.mkiv	luat-lib.mkiv	back-ini.mkiv	mult-cld.mkiv
font-gds.mkiv	strc-ren.mkiv	chem-ini.mkiv	strc-lnt.mkiv	page-bck.mkiv
lang-ara.mkiv	page-not.mkiv	strc-pag.mkiv	prop-mis.mkiv	page-set.mkiv
meta-fun.mkiv	typo-spa.mkiv	strc-prc.mkiv	pack-bar.mkiv	task-ini.mkiv
anch-bar.mkiv	strc-ini.mkiv	hand-ini.mkiv	strc-lst.mkiv	pack-lyr.mkiv
strc-bkm.mkiv	node-ini.mkiv	lang-lab.mkiv	sort-ini.mkiv	java-ini.mkiv
font-tra.mkiv	math-def.mkiv	math-frc.mkiv	lang-wrd.mkiv	supp-fil.mkiv
supp-dir.mkiv	math-lan.mkiv	meta-pdf.mkiv	scrp-ini.mkiv	luat-cod.mkiv
scrn-fld.mkiv	page-ini.mkiv	page-plg.mkiv	lang-ini.mkiv	strc-flt.mkiv
spac-par.mkiv	tabl-tsp.mkiv	font-uni.mkiv	back-pdf.mkiv	anch-snc.mkiv
page-str.mkiv	page-imp.mkiv	attr-ini.mkiv	font-unk.mkiv	grph-fig.mkiv
type-set.mkiv	strc-xml.mkiv	spac-pag.mkiv	page-app.mkiv	lang-cjk.mkiv
x-xtag.mkiv	math-for.mkiv	spac-ver.mkiv	scrn-nav.mkiv	page-flt.mkiv
math-del.mkiv	toks-ini.mkiv	chem-str.mkiv	page-txt.mkiv	tabl-ltb.mkiv
scrn-but.mkiv	prop-ini.mkiv	core-fnt.mkiv	blob-ini.mkiv	typo-cap.mkiv
core-def.mkiv	mlib-pps.mkiv	strc-sec.mkiv	lang-sla.mkiv	lang-sla.tex
spac-ali.mkiv	page-mar.mkiv	meta-tex.mkiv	node-par.mkiv	page-mul.mkiv
math-pln.mkiv	strc-des.mkiv	core-env.mkiv	pack-rul.mkiv	strc-itm.mkiv
strc-def.mkiv	colo-ext.mkiv	pack-obj.mkiv	strc-mar.mkiv	math-scr.mkiv
lxml-ini.mkiv	meta-ini.mkiv	typo-rep.mkiv	mult-ini.mkiv	tabl-pln.mkiv
strc-syn.mkiv	core-sys.mkiv	lpdf-pdx.mkiv	core-ctx.mkiv	catc-ini.mkiv
luat-ini.mkiv	scrn-men.mkiv	meta-pag.mkiv	anch-pos.mkiv	tabl-tbl.mkiv
grph-trf.mkiv	syst-lua.mkiv	regi-ini.mkiv	math-dis.mkiv	strc-not.mkiv
bibl-tra.mkiv	anch-pgr.mkiv	math-ini.mkiv	core-two.mkiv	tabl-com.mkiv
node-spl.mkiv	scrn-bar.mkiv	mult-chk.mkiv	typo-mir.mkiv	node-bck.mkiv
math-inl.mkiv	core-gen.mkiv	strc-reg.mkiv	enco-ini.mkiv	lxml-sor.mkiv
strc-ref.mkiv	char-utf.mkiv	char-ini.mkiv	strc-blk.mkiv	core-var.mkiv
page-one.mkiv	trac-tex.mkiv	spac-def.mkiv	node-fin.mkiv	scrn-int.mkiv
math-arr.mkiv	strc-num.mkiv	core-uti.mkiv	unic-ini.mkiv	char-enc.mkiv

2

meta-fig.mkiv syst-con.mkiv page-mis.mkiv symb-ini.mkiv

errors

math-int.mkiv tabl-ntb.mkiv tabl-tab.mkivmath-ali.mkiv font-ini.mkiv
buff-ver.mkiv typo-ini.mkiv

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` programm 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.