Playing with Flash in ConTEXt-mkiv
Animace ve Flashi pomocí ConTEXt-`mkiv`

Luigi Scarso

Abstract: Starting from release 9 AdobeReader, the reference PDF viewer from Adobe, has a Flash player embedded. The recent addition to CTAN of flashmovie package by Timo Hartmann prompted me to investigate the feasibility of an integration between ConTEXt-`mkiv` and swf figures. All tests were performed under Linux Ubuntu 8.04 with AdobeReader 9.3.3 installed, but I suppose they also work under Windows or Mac operating systems.

Keywords: Flash, Flash animation, ConTEXt, Mark IV, Lua

Abstrakt: Článek představuje jednu z možností, jak vložit animace vytvořené ve Flashi do PDF. Používá na to ConTEXt-`mkiv`, který využívá programovacího jazyka Lua.

Klíčová slova: Flash, animace ve Flashi, ConTEXt, Mark IV, jazyk Lua

Introduction

In the Issue 2010, Number 1 of th PracTEX journal I have published a first tentative to support SWF files in ConTEXt-`mkiv`. It was a literal translation of the `flashmovie.sty` stylesheet [2] and the result was an unusual mix of pdfLATEX and ConTEXt-`mkiv` code, but the main reason was to gain a good knowledge of the specifications and to test some applications. Just before the article was published Hans translated the stylesheet into the ConTEXt-`mkiv` lingo, so ConTEXt users can already use the swf files as figures: what I suggest here is a *all-or-nothing* way to implement the requirements of specification and also show some applications.

Implementation

The first step to support SWF files as external figures in ConTEXt-`mkiv` is to register the swf interface into the grph-inc.mkiv file with `\registerctxluafile`:

```
grph-inc.mkiv:
\registerctxluafile{grph-swf}{1.001} % this will change
```

Inside grph-swf.lua the function figures.checkers.swf(data) inserts the annotation object that identifies the swf figure using the good old `\pdfannotation` macro instead of a Lua function `node.write(pdfannotation(width,-height, 0,annot()))` (the code is commented, as one can sees), but it's one of the fews still present:

```
grph-swf.lua:
local format = string.format
local texsprint = tex.sprint
local ctxcatcodes = tex.ctxcatcodes
local pdfannotation = nodes.pdfannotation
function figures.checkers.swf(data)
    local dr, du, ds = data.request, data.used, data.status
    local width = (dr.width or figures.defaultwidth):todimen()
    local height = (dr.height or figures.defaultheight):todimen()
    local foundname = du.fullname
    local controls = dr.controls or nil
    local display = dr.display or nil
    dr.width, dr.height = width, height
    du.width, du.height, du.foundname = width, height, foundname
    texsprint(ctxcatcodes,format(
         "\\ startfoundexternalfigure{%ssp}{%ssp}",width,height))
    local annot, preview, ref = backends.pdf.helpers.insertswf {
         foundname = foundname,
         width     = width,
         height    = height,
    -- factor      = number.dimenfactors.bp,
         display   = display,
         controls = controls,
    -- label       = dr.label,
    }
 -- node.write(pdfannotation(width,-height,0,annot()))
    texsprint(ctxcatcodes,format("\\ pdfannot width %ssp height %ssp {%s}",
           width,height,annot()))
-- brrrr
    texsprint(ctxcatcodes,"\\ stopfoundexternalfigure")
    return data
end
figures.includers.swf = figures.includers.nongeneric
figures.registersuffix("swf","swf")
```

Actually the code before is modification of mine, where I've simply uncommented the `display` and `controls` variables because I need them later. The complete specifications consist of the PDF Reference sixth edition book and Adobe® Supplement to the ISO 32000 BaseVersion: 1.7 ExtensionLevel: 3 both available from [1]. The chapter 9.6 Rich Media of the Supplement describes the additional entries of the RichMedia annotation dictionary, and it's the guide to what follow; carefully reading of the RichMedia chapter and the code below reveals that there is almost an one-to-one map between the specifications and the implementation that is done by the Lua tables:

```
local format = string.format
local pdfconstant = lpdf.constant
local pdfboolean = lpdf.boolean
local pdfstring = lpdf.string
local pdfunicode = lpdf.unicode
local pdfdictionary = lpdf.dictionary
local pdfarray = lpdf.array
local pdfnull = lpdf.null
local pdfreference = lpdf.reference
function backends.pdf.helpers.insertswf(spec)
    local width, height, filename = spec.width, spec.height, spec.foundname
    local controls = spec.controls or nil
    local display = spec.display or nil
    if controls  = 'no' then
        if (parametersets[controls].replace_helper == true) and
        (type(parametersets[controls].private_helper) == "function") then
          local annotation
          annotation = parametersets[controls].private_helper(spec)
          return annotation,nil,nil
        end
    end
    local eref = backends.codeinjections.embedfile(filename)
    local configuration = pdfdictionary {
        Type    = pdfconstant("RichMediaConfiguration"),
        Subtype   = pdfconstant("Flash"),
        Instances  = pdfarray {
             pdfdictionary {
                Type     = pdfconstant("RichMediaInstance"),
                Subtype   = pdfconstant("Flash"),
                  Params = pdfdictionary {
                    Type     = pdfconstant("RichMediaParams"),
                    -- FlashVars =
                      Binding = pdfconstant("Foreground")
                  },
                  Asset = eref
             },
         },
    }
    local configuration_ref = pdfreference(pdf.immediateobj(tostring(configuration)))
    local content = pdfdictionary {
         Type            = pdfconstant("RichMediaContent"),
         Assets          = pdfdictionary {
             Names = pdfarray {
                  pdfstring(filename),
                 eref,

            }
        },
       Contents = pdfarray { configuration_ref },
}
local content_ref = pdfreference(pdf.immediateobj(tostring(content)))
local settings = pdfdictionary {
    Type               = pdfconstant("RichMediaSettings") ,
    Activation = pdfdictionary {
        Type            = pdfconstant("RichMediaActivation"),
        Condition       = pdfconstant("PO"),
        Animation       = pdfdictionary {
             Subtype    = pdfconstant("Linear"),
             Playcount = 1,
             Speed      = 1,
        },
        Configuration = configuration_ref,
        Presentation = pdfdictionary {
             PassContextClick = true,
             Style             = pdfconstant("Embedded"),
             Toolbar           = false,
             NavigationPane    = false,
             Transparent       = true,
             Window            = pdfdictionary {
                 Type      = pdfconstant("RichMediaWindow"),
                 Width     = pdfdictionary {
                      Default = 100,
                      Min     = 100,
                      Max     = 100,
                 },
                 Height    = pdfdictionary {
                      Default = 100,
                      Min     = 100,
                      Max     = 100,
                 },
                 Position = pdfdictionary {
                      Type    = pdfconstant("RichMediaPosition"),
                      HAlign = pdfconstant("Near"),
                      VAlign = pdfconstant("Near"),
                      HOffset = 0,
                      VOffset = 0,
                 }
                }
            }
        },
        Deactivation = pdfdictionary {
                Type      = pdfconstant("RichMediaDeactivation"),
                Condition = pdfconstant("XD"),
        },
    }
    local settings_ref = pdfreference(pdf.immediateobj(tostring(settings)))
    local annotation = pdfdictionary {
       Subtype           = pdfconstant("RichMedia"),
       RichMediaSettings = settings_ref,
       RichMediaContent = content_ref,
    }
    return annotation, nil, nil
end
```

RichMedia annotations have a huge set of options and the more convenient way to manage them is by a Lua table: ConTEXt-`mkiv` has an experimental mechanism that uses the global table parametersets to store and retrieve the values. What follow is not the canonical syntax `\startluaparameterset [<namespace>]..\stopluaparameterset` but a Lua version that is essentially the same:

```
\startluacode parametersets["swf:Main:controls:1"] = {
   replace_helper = true,
   private_helper = document.lscarso.insertswf
\stopluacode
\externalfigure[Main.swf][width=320px,height=180px,controls=swf:Main:controls:1]
```

The idea is clear: there is only one option `controls` instead of many keys/values and this option “points” to a dictionary of keys/values. ConTEXt-`mkiv` has also another option `display`, because the idea is a clear separation between presentation and control, but I’ve not used it in my implementation. The standard support for `swf` figures doesn’t manage the option controls, so I have added my own code:

```
local controls = spec.controls or nil
local display = spec.display or nil
if controls  = 'no' then
   if (parametersets[controls].replace_helper == true) and
    (type(parametersets[controls].private_helper) == "function")  then
       local annotation
       annotation = parametersets[controls].private_helper(spec)
       return annotation,nil,nil
    end
end
```

Now it’s time to explain what I mean with “all-or-nothing”. If you want just use a swf figure just do *nothing*, i.e. `\externalfigure[Main.swf]` is suffice. But if you need to specify some options then you must pass them to `externalfigure` together with *all* the defaults ones, and a way to pass different options is just replace the Lua function `backends.pdf.helpers.insertswf(spec)` with a private implementation `document.lscarso.insertswf` by storing its reference into the `swf:Main:controls:1` table. This is the meaning of

```
\startluacode
parametersets["swf:Main:controls:1"] = {
    replace_helper = true,
    private_helper = document.lscarso.insertswf
}
\stopluacode
```

replace_helper = true is hence a signal to backends.pdf.helpers.insertswf (spec) to replace itself with the private implementation document.lscarso .insertswf(spec). A trivial implementation of document.lscarso.insertswf (spec) is almost a copy of the standard backends.pdf.helpers.insertswf (spec):

```
\startluacode
document.lscarso = document.lscarso or {}
function document.lscarso.insertswf(spec)
   local format = string.format
   local pdfconstant   = lpdf.constant
   local pdfboolean    = lpdf.boolean
   local pdfstring     = lpdf.string
   local pdfunicode    = lpdf.unicode
   local pdfdictionary = lpdf.dictionary
   local pdfarray      = lpdf.array
   local pdfnull       = lpdf.null
   local pdfreference = lpdf.reference
   local width, height, filename = spec.width, spec.height, spec.foundname
   local controls = spec.controls or nil
   local display = spec.display or nil
local eref = backends.codeinjections.embedfile(filename)
local configuration = pdfdictionary {
   Type       = pdfconstant("RichMediaConfiguration"),
   Subtype    = pdfconstant("Flash"),
   Instances = pdfarray {
      pdfdictionary {
         Type    = pdfconstant("RichMediaInstance"),
         Subtype    = pdfconstant("Flash"),
         Params = pdfdictionary {
            Type     = pdfconstant("RichMediaParams"),
            Binding = pdfconstant("Foreground")
         },
         Asset = eref
      },
   },
}
local configuration_ref = pdfreference(pdf.immediateobj(tostring(configuration)))
local content = pdfdictionary {
   Type            = pdfconstant("RichMediaContent"),
   Assets          = pdfdictionary {
      Names = pdfarray {
         pdfstring(filename),
        eref,
      }
   },
   Contents = pdfarray { configuration_ref },
}
local content_ref = pdfreference(pdf.immediateobj(tostring(content)))
local settings = pdfdictionary {
   Type               = pdfconstant("RichMediaSettings") ,
   Activation = pdfdictionary {
      Type          = pdfconstant("RichMediaActivation"),
      Condition      = pdfconstant("PO"),
      Animation      = pdfdictionary {
         Subtype    = pdfconstant("Linear"),
         Playcount = 1,
         Speed     = 1,
      },
      Configuration = configuration_ref,
      Presentation = pdfdictionary {
         PassContextClick = true,
             Style             = pdfconstant("Embedded"),
             Toolbar           = false,
             NavigationPane    = false,
             Transparent       = true,
             Window            = pdfdictionary {
                Type      = pdfconstant("RichMediaWindow"),
                Width     = pdfdictionary {
                   Default = 100,
                   Min     = 100,
                   Max     = 100,
                },
                Height    = pdfdictionary {
                   Default = 100,
                   Min     = 100,
                   Max     = 100,
                },
                Position = pdfdictionary {
                   Type    = pdfconstant("RichMediaPosition"),
                   HAlign = pdfconstant("Near"),
                   VAlign = pdfconstant("Near"),
                   HOffset = 0,
                   VOffset = 0,
                }
             }
          }
       },
       Deactivation = pdfdictionary {
          Type       = pdfconstant("RichMediaDeactivation"),
          Condition = pdfconstant("XD"),
       },
    }
    local settings_ref = pdfreference(pdf.immediateobj(tostring(settings)))
    local annotation = pdfdictionary {
       Subtype           = pdfconstant("RichMedia"),
       RichMediaSettings = settings_ref,
       RichMediaContent = content_ref,
    }
    return annotation, nil, nil
end
\stopluacode
```

The rationale behind this implementation is that most of the time the user wants to specify only some keys/values, but sometimes a bit of programming is required, as for example to calculate the indirect reference of an object. Needless to say that Lua is almost perfect for this, so it seemed to me a natural solution to delegate the user to write the appropriate function (in this way he *must know* the options and their meaning) and let ConTEXt-mkiv replace the standard implementation with the user's one.

Application

As simple application, I've considered the programs as3compile and swfc from the swftools suite [4]. The goal is to achieve something similar to METAPOST: typeset the code and straight insert the result into the pdf, where in this case the code is ActionScript3 code that is compiled into a swf figure with the as3compile compiler, an external program. The implementation is also simple: the Action-Script code is enclosed between a couple of start/stopSWFtoolsAScode macros (with some options as the name of the script and the path of the compiler) that are in turn almost a verbatim copies of start/stopluacode macros:

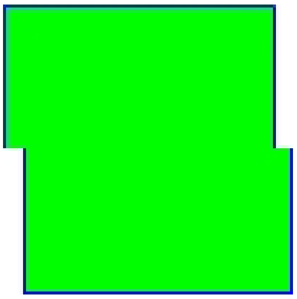
```
\long\def\dostartSWFtoolsAScode[#1]
  {\getparameters[as.][ name={out-as},preamble=preamble,compiler=as3compile,#1]%
   \begingroup
   \obeylualines %% yes, lua
   \obeyluatokens %% yes, lua
   \dodostartSWFtoolsAScode}
\long\def\dodostartSWFtoolsAScode#1\stopSWFtoolsAScode
  {\normalexpanded{\endgroup\noexpand\dododostartSWFtoolsAScode[#1]}}%
\long\def\dododostartSWFtoolsAScode[#1]{
\startluacode
   local preamble = ''
   local outfile = tostring("\csname as.name\endcsname") .. ".as"
   local swffile = tostring("\csname as.name\endcsname") .. ".swf"
   local ascompiler = tostring("\csname as.compiler\endcsname")
   local asscript_body = [=[#1]=]
   print('')
   local asscript = preamble .. asscript_body
   io.savedata(outfile,asscript)
   as_execute = string.format("\%s \%s -o \%s ",ascompiler,outfile,swffile)
   os.execute(as_execute)
\stopluacode%
}
\unexpanded\ef\tartSWFtoolsAScode{\ostartSWFtoolsAScode} % lua catcodes
\startSWFtoolsAScode[name=smile,
compiler={/opt/luatex/minimals-2010-brejlov/tex/texmf-project/bin/as3compile}]
package
  {  import flash.display.MovieClip
     public class Main extends MovieClip
     { function Main()
         {    this.graphics.beginFill(0xcccc00)
             this.graphics.drawCircle(200,200,200)
             this.graphics.endFill()
             this.graphics.beginFill(0x000000)
             this.graphics.drawCircle(140,150,50)
             this.graphics.drawCircle(260,150,50)
             this.graphics.drawRoundRect(140,270,120,10,20);
             this.graphics.endFill()
         }
     }
  }
\stopSWFtoolsAScode
\externalfigure[smile.swf][width=100px,height=100px]
```

We should also supply a default representation for the viewers that are unable to display `swf` figures, but this time it's not necessary to specify complicated options: just use the mode feature of ConTEXt as in the following example

```
\startmode[Flash]
\externalfigure[smile.swf][width=100px,height=100px]
\stopmode
\startnotmode[Flash]
\externalfigure[smile.png]
\stopnotmode
```

The same approach can be used to implement the start/stopSWFtoolsSCcode macros where the code is a swf script and the compiler is swfc (both are proprietary of swftools, see [5]):

```
\startSWFtoolsSCcode[name=action,
      compiler={/opt/luatex/minimals-2010-brejlov/tex/texmf-project/bin/swfc}]
.flash filename="action.swf" bbox=300x300 fps=50
.box mybox color=blue fill=green width=100 height=100
.put mybox
.frame 1
    .action:
         _root.angle += 0.05;
         mybox._x = 100*Math.cos(_root.angle)+100;
         mybox._y = 100*Math.sin(_root.angle)+100;
    .end
.frame 2
    .action:
         gotoFrame(0);
         Play();
    .end
.frame 3
.end
\stopSWFtoolsSCcode
\externalfigure[action.swf][width=150px,height=150px]
```



Conclusion

From the point of view of a *traditional* (i.e. not TEX) programmer ConTEXt-`mkiv` has a neat approach for implementing the PDF specifications. The Lua language is small and complete, and the PDF specifications itself are clear enough: the problem arises with the rendering of the document. On average, a free PDF viewer other than AdobeReader has not the capability to show a RichMedia content and the printing of the pdf can be also problematic, so we **must** supply the correct alternative content at least with the modes mechanism. Following the same way of `start/stopSWFtoolsAScode` it is possible to implement a `start/stopFlexAS code` (see [6]) which is the preferable to the swftools compiler due some incompatibilities in the implementation of the ActionScript3 language.

References

[1] Adobe: PDF Technology Center. Available at URL: `http://www.adobe.com/devnet/pdf/`

[2] Timo Hartmann: The flashmovie package. Available at URL:
`http://www.ctan.org/tex-archive/macros/latex/contrib/flashmovie`

[3] Luigi Scarso: Playing with Flash in ConTEXt-mkiv. In The PracTEX Journal, Vol. 5, No. 1, 2010. ISSN 1556-6994. Available at URL:
`http://www.tug.org/pracjourn/2010-1/scarso`

[4] SWFTools – SWF manipulation and generation utilities.
Available at URL: `http://www.swftools.org`

[5] SWFC Manual. Available at URL: `http://www.swftools.org/swfc/swfc.html`

[6] Adobe: Adobe Flex. Available at URL: `http://www.adobe.com/products/flex`

luigi (dot) scarso (at) gmail (dot) com
Padova, Italy