How we try to make working with TEX comfortable

Hans Hagen
TUG Conference
Tokyo, October 2013

Convenience	Macros	Setups
Modes	Integration	Extensions
Definitions	Formatters	Interfacing

Convenience

- It all starts with as much structure as possible so that we get configurability and reuse for free. It also leads to less errors.
- The source code has to look nice too. The worse the source looks, the more chance that the result looks bad too.
- An edit–preview cycle has to be pleasant which means that processing has to happen fast and the (pre)viewer has to be good.
- Some form of project management support helps reuse of content and resources. Image management is a must. It's more than running TEX.
- Coding should be easy and methods should suit the needs. Mixing methods should still look nice and consistent.
- Here I will show a few variants of coding.

Macros

- The ConTEXt interface was originally driven by line–by–line syntax highlighting: if we can't make it look good and highlight it well, it should be done differently.
- Wherever possible we use square brackets for optional arguments and configurations. In cases where that makes no sense we use braces.
- Users can use their own macros but of course have to make sure they don't clash. Most mechanisms have hooks.

`example-macros.tex`

Setups

- There are several ways to reuse data, for instance using buffers and blocks and of course components in the project structure.
- We added so called setups to isolate large blocks of runtime code.
- Instead of passing arguments to macros or setups you can pass variables.
- Setups are used all over the place from processing nodes in an XML tree to rendering alternatives for lists, section heads, etc.

`example-setups.tex`

Modes

- Already early in the development of ConTeXt modes were introduced to control alternative rendering of documents (products).
- (Combinations of) modes can be set and unset in the document (style).
- You can also use the command line: `context --mode=answers somefile`.
- The system itself also uses modes to communicate states.
- We often use them in job control files (like `jobname.ctx`).

`example-modes.tex`

Integration

- One of the first subsystems was runtime MetaPost graphics.
- Other subsystems showed up after that, but instead of core support they now rely on the `filter` module.

`example-integration.tex`

Extensions

- We've chosen Lua as the language for extending the TeX engine.
- You can use this language from the TeX end but you can also access much of TeX from the Lua end.
- Embedding Lua code is supported in various ways and for sure more will show up.
- The most extreme examples are `cld` documents.

`example-extensions.tex / example-cld.cld`

Definitions

- As an experiment I started playing with the macro language.
- We keep what is there but have a cosmetic layer on top.
- Part of MkIV uses this approach, and when used this code is tagged MkVI.

`example-definitions.tex`

Formatters

- There are a lot of Lua helpers available and an API to the internals is evolving.
- Some helpers are integrated into the `context` namespace.
- Mechanisms that are used elsewhere in our toolchain also get included and interfaced.

`example-formatters.tex / example-templates.tex`

Interfacing

- How far do we want to go with interfaces?
- ConTeXt always had a multi–lingual user interface. How useful is this and how should it evolve?

`example-interfaces.tex`

example-macros.tex

```
\setupbodyfont
  [dejavu]

\starttext

\startchapter[title={My Title}]

    Just some text before we itemize.

    \startitemize[packed]
        \startitem first one \stopitem
        \startitem second one \stopitem
    \stopitemize

\stopchapter

\stoptext
```

example-setups.tex

```
\setupbodyfont
  [dejavu]

% document setups

% \setupdocument
%   [after={
%     \startsetups document:after
%         \startstandardmakeup
%             \startalign[middle]
%                 The End.
%             \stopalign
%         \stopstandardmakeup
%     \stopsetups
%   }]

\setupdocument
  [after=\setup{document:after}]

\startsetups document:after
    \startstandardmakeup
        \startalign[middle]
            The End.
        \stopalign
    \stopstandardmakeup
\stopsetups

% other setups

\setvariables
  [example]
  [set=\setup{example:action}]

\startsetups example:action
    \blank
    \midaligned {Here is \quotation {\getvariable {example} {whatever}}}
    \blank
\stopsetups

% here we start the document

\startdocument

    \input{ward}

    \setvariables[example][whatever=Some Text]

    \setvariables[example][whatever=Some Other Text]

\stopdocument
```

example-modes.tex

```
% \enablemode[dyslexic]
\enablemode[dyslexic,smaller]

\doifmodeelse {dyslexic} {
    \setupbodyfont[opendyslexic]
} {
    \setupbodyfont[pagella]
}

\startmode[smaller]
    \setupbodyfont[10pt]
\stopmode

\starttext

    \input {davis}

\stoptext
```

example-integration.tex

```
\starttext

\startMPcode
    fill fullcircle xysized 10cm withcolor .5[red,green] ;
    draw textext("\bf TUG 2013") xsized 5cm withcolor white ;
\stopMPcode

\startuseMPgraphic{fuzzy}{color}
    fill OverlayBox squeezed -.5ExHeight withcolor \MPvar{color} ;
\stopuseMPgraphic

\defineoverlay[fuzzy][\useMPgraphic{fuzzy}{color=darkgreen}]

\framed
  [background=fuzzy,
   align=middle,
   offset=5mm,
   frame=off]
  {\input{ward}}

\startuseMPgraphic{fuzzy}
    fill OverlayBox squeezed .5ExHeight withcolor OverlayColor ;
\stopuseMPgraphic

\defineoverlay[fuzzy][\useMPgraphic{fuzzy}]

\framed
  [background=fuzzy,
   backgroundcolor=darkblue,
   foregroundcolor=white,
   align=middle,
   offset=5mm,
   frame=off]
  {\input{ward}}

\stoptext
```

example-extensions.tex

```
\starttext

    \startluacode

     -- context.strut()
        context("Hi there!")

    \stopluacode

    \blank

    \startluacode
        context.bTABLE()
            for i=1,15 do
                context.bTR()
                    for j=1,5 do
                        context.bTD()
                        context("cell (%s,%s) is %s",i,j,document.variables.text or "unset")
                        context.eTD()
                    end
                context.eTR()
            end
        context.eTABLE()
    \stopluacode

\stoptext
```

example-cld.cld

```
context.setupbodyfont { "dejavu" }

context.starttext()

    context.startchapter { title = "MyTitle" }

        context("The number π is about %1.16f.",math.pi)

    context.stopchapter()

context.stoptext()
```

example-definitions.tex

```
% macros=mkvi

\starttext

\def\testmacro#one#two%
  {\par
   [#one]%
   [#two]%
   \par}

\testmacro{1}{2}

\testmacro{one}{two}

\testmacro{second}{first}

\starttexdefinition testmacro #one #two
    \par
    [#one]
    [#two]
    \par
\stoptexdefinition

\testmacro{alpha}{beta}

\stoptext
```

example-formatters.tex

```
\setupbodyfont
  [dejavu]

\starttext

    \setbox \scratchbox = \hbox {A test}

    \startluacode
        context("the width of this box is %p",tex.box.scratchbox.width)
    \stopluacode

    \startluacode
        document.mytemplate = [[
            \starttext

                \startchapter[title={%title%}]
                        \input {%filename%}
                \stopchapter

            \stoptext
        ]]

        context.templates[document.mytemplate] { title="Ward", filename="ward.tex" }
    \stopluacode

\stoptext
```

example-templates.tex

```
% macros=mkxi

% Context recognizes the file suffix as well as the preamble. The mkix filetype just
% compiles, while the mkxi filetype also applies mkvi translation. This last step is
% somewat tricky as it is applied on the template.

\setupbodyfont
  [dejavu,8pt]

\starttext

    \bTABLE
        <?lua for i=1,15 do ?>
            \bTR
                <?lua for j=1,5 do ?>
                    \bTD cell (<?lua inject(i) ?>,<?lua inject(j)?>) is <?lua inject(variables.text
or "unset") ?>\eTD
                <?lua end ?>
            \eTR
        <?lua end ?>
    \eTABLE

    \page

    \startluacode
        context.bTABLE()
            for i=1,15 do
                context.bTR()
                    for j=1,5 do
                        context.bTD()
                        context("cell (%s,%s) is %s",i,j,document.variables.text or "unset")
                        context.eTD()
                    end
                context.eTR()
            end
        context.eTABLE()
    \stopluacode

\stoptext
```

example-interfaces.tex

```
% engine=luatex macros=mkvi

\definefont [testfont] [heiseiminstd-w3] [script=kana,language=jan]

\starttext

    \testfont

    \startluacode

        local function 例題(str)                -- example
            context("例題 1.%s: 数 %s",str,str) -- example ...: number ...
        end

        for i=1,10 do
            context(例題(i))
            context.par()
        end

    \stopluacode

    \def\例題#1{例題 2: 数 #1\par}

    \例題{2.1}

    \startluacode
        context.例題(2.2)
    \stopluacode

    \starttexdefinition test #1
        例題 3: 数 #1 \par
    \stoptexdefinition

    \test{3}

    \starttexdefinition 例題 #1
        例題 4: 数 #1 \par
    \stoptexdefinition

    \例題{4}

    \def\例題#数{例題 5: 数 #数\par}

    \例題{5}

    \starttexdefinition 例題 #数
        例題 6: 数 #数 \par
    \stoptexdefinition

    \例題{6}

    \starttexdefinition unexpanded 例題 #数
        例題 7: 数 #数 \par
    \stoptexdefinition

    \例題{7}

    \startluacode
        function commands.Σ(...)
```

```
            local t = { ... }
            local s = 0
            for i=1,#t do
                s = s + t[i]
            end
            context("% + t = %s",t,s)
        end
    \stopluacode

    \ctxcommand{Σ(1,3,5,7,9)}

    \def\Σ#1{\ctxcommand{Σ(#1)}}

    \Σ{1,3,5,7,9}

\stoptext
```