Interacting pdfTEX, PERL and ConTEXt

Gilbert van den Dobbelsteen
Papaverstraat 130
7514 XH Enschede
Netherlands
gilbert@login-bv.com

Abstract

PERL, pdfTEX, and ConTEXt are extremely usefull in the production af large documents which need a lot of interaction too. This article resulted from a job I did for a good friend, yielding in over 2000 pages of PDF output.

The power here is to use the right tool for the right job. Almost everything created for this job could be done in TEX, but since I am just a 'Ben Lee User', I use different tools to get the job done. So it is not a matter of which tool is the best for the job, but more like *Which tool is best for the person using the tool.*

Introduction

A few months ago, a good friend (let's call him Bart, because that's his name) had a problem. He had taken on a job where he needed to create an interactive document consisting of over a thousand paragraphs. All texts needed to be clickable, and as a result a poster should pop-up with the same text, but artistically enhanced. The texts originated from the LOESJE association and so did the posters.

I advised him to take a look at ADOBE ACROBAT. He did and he already made a framework with some buttons and clickable links. He started calculating, and decided this was too much work. Every link had to be manually created, And since each poster-text had about three to four categories this meant drawing over 5000 clickable areas by hand in ACROBAT. Though LOESJE has many volunteers, you can't give them such a boring job. It would simply kill the relaxed atmosphere, normally hanging around LOESJE.

So I told him that he could probably use some programming tool to automate things. Since Bart is dyslectic (it is very difficult for him to read words from paper or screen), he is unable to do classical software engineering jobs, so in the end I volunteered to do the job for him.

I usually write small documents, which aren't larger then 100 pages, but I was very sure TEX is capable of doing so. Interactive programs usually have big problems dealing with large files and many pages, but since TEX is batch oriented I knew this wasn't going to be a problem.

About loesje

LOESJE is an association of people who design strong texts for different applications. Some text-categories are: Elections, politics, Year 2000 problems, Astrology, Economy, stock exchange, Christmas, nature, animals, poetry, religion, school, health-care, etcetera.

These texts are put onto posters and flyers and you can see them anywhere around the Netherlands. You can also buy post-cards and other stuff.

The main idea is to trigger people to think about what is going on. A typical text from LOESJE:

Year 2000: Suppose the end of the world is near and God forgot to make a backup

The LOESJE association has been around since 1983, and throughout the years the created 1350 different texts.

To celebrate their 15 year existance they decided to creat a cd-rom with all posters on it, and with a nice catalogue, where you browse the texts category-wise or chronologically.

How things got started

So I had to define some structure before I could begin. In the beginning of LOESJE they used markers and pencils to create posters by hand, and reproduced them with a large xerox machine. So those posters weren't available in a digital format. They started using computers many years later, so much of the material was only available on paper.

To assure quality and consistent presentation they decided to scan all posters. LOESJE has a Scanjet 2 and lot's of volunteers. The scanner was old and the compressed TIFF output generated

TIFF files with errors, so they had the fall back to uncompressed TIFF.

After a few weeks Bart came to me with 10 CD's full of uncompressed TIFF bitmaps. Each file was 4Mb in size consisting of a 600DPI A4 scan of each poster. This started to terrify me. My computer had about 3Gb of free disk space, which was definitely not enough for ten CDs of data. How to proceed? I knew that I needed the files in PNG format for inclusion in pdfTEX. So I decided to convert all files to PNG with the ImageMagick tools. This took 8 hours of computer time and in the end I discovered the dimensions where lost in the resulting PNG-file. After investigating the originals I concluded the dimensions weren't present there too.

Since the PNG format is compressed, and the monochrome scans are very large, the total size reduced from 10 CDs to $\frac{1}{6}$th CD. This was a managable amount of data.

Texts and categories Besides the scanning of the actual posters, I needed the actual texts that was on the posters.

The text-file contains lines of text for each poster. To keep things simple, LOESJE keyed in all the data. A typical intry looks like:

```
N199312C Actual text, perhaps
sevaral lines

long [3\7\13]
```

The above means: The file N199312C.PNG is the actual poster containing this text, the year is 1993, the month is December (12) and this is the third poster (C) in that month. The poster falls in three categories: 3, 7 and 13.

The resulting type-set layout should observe the new- and empty lines in the files.

To convert the category numbers to actual category names there was another text file: `cat.txt`. This file looked something like this:

```
Alien
         9 Common
        10 Astrology
        11 Space
Future
        72 Common
        73 Dreams/ideals
        74 Plans
        75 2002
```

The above means: The main category 'Alien' contains the subcategories: Common (9), Astrology (10), and Space (11). The main category 'Future' contains the subcategories: Common(72), Dreams / ideals (73), Plans (74), and 2002 (75).

These files are fairly easy to scan with PERL. The scanning code is just a screen or two. After each text definition is scanned a PERL object is constructed with the following attributes:

Year The year of the poster.

Month The month of the poster.

Categories An array containing the category numbers for this particular poster.

Text The actual text.

All the poster objects are put into a hash (a key-value pair array) where the key is the unique poster number (like N199312C).

After the scanning and building of the has is complete the output-files were constructed.

Using different tools

I am a tool-guy. I use whatever tool that I know could do the job easy. The advantage here is obvious: the right tool for the right job gets the work done more quickly. There's also a disadvantage: I usually do not know the exact ins– and outs of a tool. I know little of TEX, in fact the way TEX 'thinks' is definitely not my way. I see TEX as an enhanced M4 macro-processor, with weird syntax, nice output and unlimited possibilities.

Do not blame me for my limited vision of this powerfull typesetting engine, it is just the way I work with it. My macros are not nice and I usually overlook powerfull features, but they get the job done I hope. As I write macros (in any language be it TEX or PERL) I experiment until the result is what I want. If the used tool can't do the job for me (usually because I am too stupid to find the right keywords) then I'll try another tool until the results are satisfactory.

The same story holds for PERL. If a take a look at the packages that come with PERL I am amazed about the possiblities. You can even write web-servers in PERL with just a few lines of code. I used PERL before to convert structured text documents to PDF en HTML with everything cross-linked and it is definitely a *very* powerfull tool for doing system stuff like messing with files, directories and contents of files.

PERL and TEX have something in common: both are a bit weird, though PERL looks more like a conventional programming language to me. To achieve things in both tools, you can use several mechanisms and language constructs. This is better accepted in the PERL world as in the TEX world. I sometimes overhear conversations about TEX where people are trying to convince each-other that their way is the best way to do it. I do not believe in such a concept. The best way to achieve things is the way that generates the most fun and gets the job done.

perling it away

This section should definitely not be read by any advanced TeX user and specifically ConTeXt user. That's because they would claim that all this structurizing I did could easily be done from within ConTeXt. Ok, I admit, that is very true and ConTeXt does support a lot of usable stuff for me. The only problem is that I don't know them well enough.

Using PERL to scan the files was easy. Generating the output however was more difficult.

I first needed to know what kind of browsing these LOESJE guys would need. They wanted two things:

1. Chronological. You can browse through the poster-texts sorted on date. Below should be a button-bar with the available years, and above that a button-bar with the months in that year. Each poster-text will be included once.
2. Categorial. It is similar to the chronological but organized by main- and subcategories.

So I decided to use a section / subsection mechanism as found in LaTeX.

A sample of the output:

```
\\YEAR{1993}
\\MONTH{January}
\startposter{N199312C}
Hi there, this is some poster text
\stopposter
```

The macro's `\startposter` and `\stopposter` should do all the work (I'll come back on those later).

Using ConTeXt to do the layout

Many of you probably know ConTeXt as a very powerfull program for creating interactive documents. If you don't believe me, try it for yourself. The trouble with ConTeXt is finding the right way to do it. There are usual several.

Almost all of the features found in the PDF specifications can be used. In some aspects ConTeXt defines more functionality than PDF has to offer. The whole concept behind ConTeXt is well thought-out and Hans Hagen is a true wizard when it comes to functionality and completeness. If you have a nice generic package or add-on, Hans is willing to integrate it in ConTeXt given the time. Modularity in ConTeXt is something weird, because the package is large and monolithic. In fact the basic services in ConTeXt are rather limited when it comes to 'I want to write an article'. But once you get the hang of it you discover that customizing things is a breeze, compared to whatever I've ever encountered in TeX miracle land. You do not need to know a lot about TeX (which is definitly a big plus) and it usually works the way you expect. And if you're not certain about the correctness of the output, you simply turn on the visual debugger, and you can actually *see* where you forgot that extra percent sign, yielding that unwanted space.

Defining the layout Take a look at figure 1. It is the basic layout. All the screens in the product are similar to this one, so I designed a basic layout to create this.

Figure 1: Basic layout

Defining the layout is simple. You first setup the papersize:

```
\setuppapersize
  [S6]
```

The S6 means: Screen based papersize. It is similar to A4, except that the width is 600pt and the height is 450pt. ConTeXt setups margins and automatically calculates the text-area. For screen based layout, there's one thing for sure: Whatever ConTeXt calculates it is never what you want. It works fine for paper based output.

So now let's setup the areas to be used:

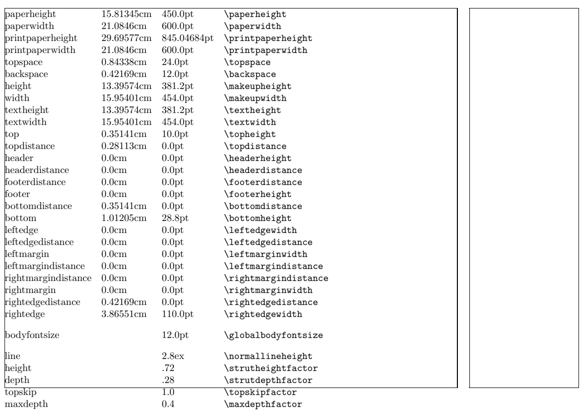
```
\setuplayout
  [topspace=24pt,
   header=0pt,
   footer=0pt,
   bottomdistance=10pt,
   bottom=28.8pt,
   topdistance=8pt,
   top=10pt,
   backspace=12pt,
```

```
  margin=0pt,
  edgedistance=12pt,
  rightedge=110pt,
  height=fit,
  width=fit]
```

What does this all mean?

- There is 24pt of space on the top, before the text-area begins.
- There is no header text (above the text) or footer text (below the text). These are usually used to put in page numbers of chapter headings. I don't need them for screen-layout.
- There is bottom-text below the text area, its height is 28.8pt. There is also top-text above the text, height 10pt.
- There are no margins.
- The distance from the text-area to the edge is 12pt. The width of the rightedge is 110pt.
- And finally we say: 'Dear ConTEXt I do not know what the width and height of the text-area should be, so calculate them based on the given settings'.

That's about it. After that you can verify what you have done by saying: `\showframe`. ConTEXt then draws some frames where you defined them, so you can actually see what is going on. See figure 2.



paperheight	15.81345cm	450.0pt	\paperheight
paperwidth	21.0846cm	600.0pt	\paperwidth
printpaperheight	29.69577cm	845.04684pt	\printpaperheight
printpaperwidth	21.0846cm	600.0pt	\printpaperwidth
topspace	0.84338cm	24.0pt	\topspace
backspace	0.42169cm	12.0pt	\backspace
height	13.39574cm	381.2pt	\makeupheight
width	15.95401cm	454.0pt	\makeupwidth
textheight	13.39574cm	381.2pt	\textheight
textwidth	15.95401cm	454.0pt	\textwidth
top	0.35141cm	10.0pt	\topheight
topdistance	0.28113cm	0.0pt	\topdistance
header	0.0cm	0.0pt	\headerheight
headerdistance	0.0cm	0.0pt	\headerdistance
footerdistance	0.0cm	0.0pt	\footerdistance
footer	0.0cm	0.0pt	\footerheight
bottomdistance	0.35141cm	0.0pt	\bottomdistance
bottom	1.01205cm	28.8pt	\bottomheight
leftedge	0.0cm	0.0pt	\leftedgewidth
leftedgedistance	0.0cm	0.0pt	\leftedgedistance
leftmargin	0.0cm	0.0pt	\leftmarginwidth
leftmargindistance	0.0cm	0.0pt	\leftmargindistance
rightmargindistance	0.0cm	0.0pt	\rightmargindistance
rightmargin	0.0cm	0.0pt	\rightmarginwidth
rightedgedistance	0.42169cm	0.0pt	\rightedgedistance
rightedge	3.86551cm	110.0pt	\rightedgewidth
bodyfontsize		12.0pt	\globalbodyfontsize
line		2.8ex	\normallineheight
height		.72	\strutheightfactor
depth		.28	\strutdepthfactor
topskip		1.0	\topskipfactor
maxdepth		0.4	\maxdepthfactor

Figure 2: Empty layout

The resulting PDF file still has a problem though: the layout is OK, but the page size is still A4. To overcome this you say:

```
\setupinteractionscreen
  [option=max,
   width=fit,
   height=fit]
```

This more or less means: Open the document in full-screen, and make the width and height fit to the calculated values.

Defining backgrounds I now defined a simple white page, so let's enhance it with background-colors and graphics. If you do this the traditional way, by drawing something in the output routine, before you actually shipout the page you will have a problem: The resulting file will contain the graphics images on each and every page. That is not very efficient, since the background images are a total of 73Kb. If you multiply that by the number of pages (about 800) you will get a pdf file of 60 Megabytes. All of this while you know for sure that PDF supports something that's called *objects* to do the job properly.

Luckily you don't have to worry about that. You simple use and include as many figures as you like. If ConTEXt sees you've actually used the figure before, it won't include it again. It will simply creates a reference to the figure. So each figure is included only once.

First we define a background-color for the enitre page:

```
\setupbackground
  [page]
  [background=color,
   backgroundcolor=pagebackgroundcolor]
```

So what is `pagebackgroundcolor` you might ask. Well:

```
\definecolor
  [pagebackgroundcolor]
  [r=.9, g=.9, b=.9]
```

Easy ha? You can probably guess how to define cmyk colors. The right edge has a similar background color, but with a little different setup:

```
\setupbackgrounds
  [text][rightedge]
  [background=color,
   backgroundcolor=edgebackgroundcolor]
```

So now the picture in the text-area. This is a bit more difficult, because this works with overlay-techniques:

```
\defineoverlay
  [world]
  [{\externalfigure
    [world]
    [width=\overlaywidth,
      height=\overlayheight]}]

\setupbackgrounds
  [text][text]
```

```
  [background={color,world},
   backgroundcolor=textbackgroundcolor]
```

What's this? We are defining two backgrounds here, a color, and a picture that comes from world.eps file. The order of specification defines the overlaying. I do not know if there are any limits here but knowing Hans there is probably no real limit to the number of backgrounds you can stack upon each-other.

Also note the sizing of the image. The image is a bitmap converted to en .eps file (using CorelDraw) and has some transparency. You only need to get the aspect ratio of the .eps file correct, and ConTEXt automatically scales the image to the width and height of the text area.

Defining the right-edge navigation bar The navigation bar on the right contains many buttons stacked above each other. The normal command to do this is `\button`. But since the buttons have backgrounds I needed to define the backgrounds first. The trick is this: You first define what you want to have, put it all in a box and then you say to ConTEXt:

```
\setuptexttexts
  [edge]
  [][\ButtonList]
```

This means: The content of the left edge is empty, and the right edge contains `\ButtonList`.

`\ButtonList` is defined as follows (some buttons are left out here to save some space):

```
\def\ButtonList{%
  \hbox to \rightedgewidth{\vbox to \vsize
    {\midaligned{\previousbutton}\par
     \midaligned{\nextbutton}\par
     ...
     \vfill
     \midaligned{\stopbutton}\par
     \vfill
     \midaligned{\PrevNextBar}\par
     \vss}}}
```

To define a button:

```
\def\previousbutton{%
  \button
    [frame=off,
      width=104bp,
      height=25bp,
      background=previous]
      {}
      [previouspage]}
```

There is no frame around the button, because I supplied a picture. The button contains no text (the empty braces{}), and when someone presses the button, the previous-page should be displayed. All buttons are created this way.

Now there is one little speciality here: The `\PrevNextBar` command. When a series of text is displayed, and they don't fit on a single screen, a button automatically appears indicating that there is more to see in this series. Now I wanted those buttons to be smart: On the first page you only see a button to go to the next page, and in the last page of a series you only see a button to go to the previous page. The pages inbetween (if any) have both buttons enabled.

Again ConTEXt helps me out here. There is something called sub-page numbering. I setup subpage numbering by series, and when placing the buttons I ask ConTEXt how many pages there are in this series. Based on that, and the current subpage number I include the correct buttons:

```
\def\PrevNextBar%
  {\ifnum\nofsubpages>1
     % all right, we need some buttons
     % here
     \ifnum\subpageno=1
       % This is the first page so only
       % include the right button
       \framed
         [frame=off,
           width=\arrowwidth,
           height=25bp]{}\hss
         \arrowrightbutton%
     \else\ifnum\subpageno=\nofsubpages
       % This is the last page, so only
       % include the left button
       \arrowleftbutton\hss
     \else
       % We are somewhere in between
       % include both buttons
       \arrowleftbutton\hss
       \arrowrightbutton%
     \fi\fi
   \else
     % No sub pages, so no buttons
     \framed
       [frame=off,
         height=25bp,
         width=\rightedgewidth]
       {}
   \fi}
```

Now that's a nice trick isn't it?

Defining the bottom navagation bar The bottom navigation bar is a bit more difficult, since it contains data from the actual data-file itself. Since all years and months are placed in lists, I can easily extract that information and put it anywhere where I want. I only provide the details here, the actual commands to setup texts in the bottom area are similar to the right-edge button bar. It consists of two vboxes, a box for the months and one for the years.

```
\def\YearMonthList{%
  \vbox to \bottomheight{%
    \vbox{%
      \placelist
        [MONTH]
        [variant=none,
         criterium=YEAR,
         command=\SIMPLE,
         before=\strut]
        \vss}%
    \vss
    \vbox{%
      \placelist
        [YEAR]
        [variant=none,
         criterium=all,
         command=\SIMPLE,
         before=\strut]
        \vss}}}

\def\SIMPLE#1#2#3{#2}
```

At first this didn't do the job, because all years were on a seperate line. I guess the `\placelist` results in a lot of vboxes, and these are stacked vertically by TEX. By adding the `\strut` the problem was over. This is probably not the prefered solution, but I can't write an email to Hans every time I am in trouble.

As you can see I didn't provide any commands for specifying the interaction and high-lighting. This is what ConTEXt automatically does for me.

Results and some samples

Take a look at the figures. Figure 3 shows the main screen of the categories.

Figure 4 Is a actual poster.

Conclusion

Well, what can I say? ConTEXt saved me a lot of time, and the remaining time creating the product was fun to spent. I only covered some basic issues here, there are several other commands needed to get things going.

Using the example files provdided by ConTEXt gave me a good idea of the possibilities. Though the examples presentations do not look like this product, they serve as good examples of what is possible.

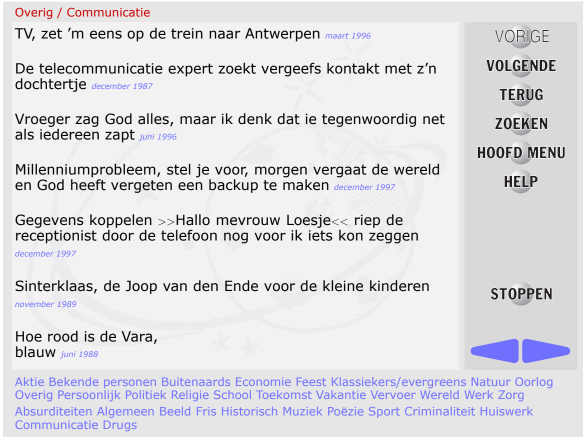


Figure 3: A category screen

Figure 4: An actual poster

I liked the tooling very much. It is easy to generate TEX documents from PERL, and ConTEXt is pretty relaxed in using such documents.

You probably discovered that the included graphics is in Dutch. Most of the words are what you expect of them, so I expect this is not a big problem.