Proofing and production with ConTEXt

Ton Otten

ConTEXt has several built-in tools that can help you during the proofing stage of your project. Furthermore there are a few ConTEXt features you can invoke in your style file that gives you visual feedback of inconsistencies or errors. And then there are modes.

Below I will describe a number of the techniques I am using in the proofing stage of the Math4All project.

Project background

The Math4All project is an XML project. The resources are used to generate a website for an online curriculum (`www.math4all.nl`) and a paper version using a customizing service that produces textbooks and tests in PDF format.

> You can try the customizing service yourself:
>
> link: `https://pod-m4all.pragma-pod.nl`
> user: `context user group`
> pswd: `guest`
>
> Available until 31 December 2021.

To give you an impression of the scope of the Math4All project:

- the content covers the math education of the complete 'Voortgezet Onderwijs' ("advanced education") in the Netherlands for pupils aged between 12 and 18 years old
- it covers all levels of high school math: MAVO (4 years), HAVO (5 years) and VWO (6 years)
- the resources contain 20,000 XML snippets, 8000 svg and 9000 JPEG images
- the resources are aggregated into 150 chapters with an average of 50 A4 reader pages of theory, examples and questions and 30 A4 reader pages of answers
- the math is coded in MathML and AsciiMath

The authors work directly in the XML and do their proofing when they publish the website. Since publishing for the web (HTML) is much more versatile than publishing in ConTEXt, there is still a lot to do.

Clean Up

The XML resources stem from another project so massive clean-ups were inevitable. These were done in the Oxygen editor. For example:

- remove leftovers such as old XML directives
- remove distortions from other editors
- fix wrong coding
- remove spurious spaces, e. g. like in > . and > :
- insert new XML directives that are used in this project

I use Oxygen because it enables me to perform massive editing actions on XML content that is spread over several (sub) directories.

After that I rely on my ConTEXt workflow.

Workflow

The characteristics of the Math4All proofing and production XML2PDF workflow are:

- operating system: Windows
- text editor: SciTE
- PDF viewer: SumatraPDF
- PDF viewer sometimes: Acrobat, because of tooltips, etc.
- typesetting engine: ConTEXt LuaMetaTEX Version 2.06.16
- runfile: `runfile.xml`
- ...

I am not using the latest LuaMetaTEX version, because I have to be certain that the text flow on all laptops and servers is identical. Once a year, LuaMetaTEX is updated on all machines.

SyncTEX

By default the `synctex` option is set in the style file by:

```
\setupsynctex[state=start]
```

This feature 'synchronizes' the PDF and the source file. Double-clicking the text in the PDF file opens the designated XML file in SciTE at the correct line location. With the content stored in hundreds of small XML files, this is an essential feature.

It is said that using SyncTEX comes at a performance loss. The average chapter of 50 A4 pages runs in 7.5 seconds versus 7.3 seconds without it (on my laptop) so I do not think this is a serious problem.

Log file

When a file is processed, a lot of information is directed to the log file. The log file is essential when the process is halted and an error is generated. Fatal errors in an XML project are mostly due to wrong XML, MathML or AsciiMath coding.

Minor syntax errors are also shown in the log file. In case of the forgotten right parenthesis in "<am>sqrt (x </am>", the log file will show the following message:

```
mathematics  > asciimath > not enough right fences: 1: sqrt ( 23 :
\asciimathsqrt{(} x
```

In the PDF itself there is also a remark. Instead of the math expression you will see:

```
< not enough right fences : 1 >
```

In the proofing trajectory, the log file is most useful while scanning for overfull boxes and/or missing images.

If necessary, you can also write your own information to the log file with:

```
\writestatus {>>>>>>>>>>> WORK TO BE DONE}
```

at the right location in the style file.

Modes

To support proofing, I introduced a number of modes into the styles which give me visual feedback when, for example, specific XML content is missing. The missing content is flagged in the PDF.

Since we produce for the web and paper (with multiple products and variations), signalling missing content takes place at the start of the proofing process.

- images
- answers and explanations (to the questions)
- content for specific target groups
- appendices for specific products
- worksheets
- ...

For example:

```
<item>
  <intro>
    Intro to question.
  </intro>
  <question>
    Is this a question?
  </question>
  <studentanswer>
    Yes.
  </studentanswer>
  <studentexplanation>
    It is only a question when the line ends with a question mark.
  </studentexplanation>
```

```
  <webfeedback>
  </webfeedback>
  <teacheranswer>
    Please refer to questions in general.
  </teacheranswer>
</item>
```

In the mode ‘complete’, I need to see all content. In the style you can write something like this:

```
\doifmode {complete} {
  \xmldoifelseempty {#1} {webfeedback} {
    \inmargin[frame=on] {No feedback!}
  } {
    \xmlflush{#1}
  }
}
```

Spell check

For obvious reasons, we want to check the spelling of the complete content of a chapter. This must be done in the early stages of the proofing trajectory, since corrections may influence hyphenation, line breaks, placement of floats and text flow.

After I received the first ten chapters I started the project-specific word list. I used the tex file below to generate the first version.

```
\setupspellchecking[state=start,method=2]

\ctxlua{languages.words.threshold=3}

\starttext

\xmlprocessfile{main}{the-project-content.xml}{}

\ctxlua{
  context.par()
  local data = table.load(file.addsuffix(tex.jobname,"words"))
  context(data and data.total or "nothing")
}

\stoptext
```

If you process this `jobfile.tex`, it will generate a file `jobfile.words` containing all the words of more than three characters (`languages.wordsthreshold=3`) that are used in the file `the-project-content.xml`.

After making the project-specific word list, you need to check and edit it manually. After that you can use it during the spell checking phase in the proofing trajectory. New words are added to `*.words` files continually until the project is finished.

Preferably the word list is stored in the style file map in your ConTEXt distribution.

You can invoke the spell check in your style file with:

```
\doifmode{spelling}
  {
    \loadspellchecklist[nl][jobfile.words]
    \setupspellchecking[state=start,method=1]
  }
```

In this case, the mode 'spelling' is used.

After making the project-specific word list, you need to check and edit it manualy. After that you can use it during the spell cheching phase in the proofing trajectory. New words are added to `*.words` files continually until the project is finished.

The words in red are incorrect.

Hyphenation

When you work in a specific domain, like high school math for example, you encounter specific words that require their own hyphenation patterns. This calls for a specific hyphenation list.

You can add your hyphenation list in the main style file or use a separate hyphenation file. To do this, use the `\hyphenation` command:

```
\hyphenation {
   rich-tings-coëffi-ciënt
   rich-tings-coëffi-ciënten
   vari{-}{}e{}{}ë}ren
   vari{-}{}e{}{}ë}rend
}
```

You can also use the `\start...\stopexceptions` command:

```
\startexceptions[nl]
  Pytha-go-ras
  bac-te-riën
  rich-tingscoëf-fi-ciënt
  pa-tiën-ten
\stopexceptions
```

When the hyphenation point comes directly after an ë, ï, ü or ö, you have to lose the diaeresis in Dutch texts. This takes some more coding in your hyphenation list.

With the `\setuplanguage` command, you setup the minimum number of characters you want before and after the hyphen:

```
\setuplanguage[nl]
  [lefthyphenmin=3,
   righthyphenmin=3]
```

Partial run

If your XML project consists of thousands of XML snippets (and a chapter is sometimes aggregated out of hundreds of XML snippets), it is convenient to be able run the XML snippets seperately. It saves a lot of time.

For this, the XML snippet is taken out of its surrounding XML element so you have to make some adjustments in the style file; I used a mode called `standalone` that is invoked by a directive.

See the example below.

```
<exercise>
  <?context-directive job ctxfile m4all-folio-standalone.ctx ?>
  <single-item>
    <item>
      <question>Is this a question?</question>
      <answer>Yes.</answer>
    </item>
  </single-item>
</exercise>
```

The ConTEXt directive calls up a `ctx` file that gives job-specific information to the XML2PDF process:

```
<ctx:job>
  <ctx:message>math4all project</ctx:message>
  <ctx:process>
    <ctx:flags>
        <ctx:flag>run standalone</ctx:flag>
    </ctx:flags>
    <ctx:resources>
        <ctx:mode>standalone</ctx:mode>
        <ctx:environment>mystyle.tex</ctx:environment>
    </ctx:resources>
  </ctx:process>
</ctx:job>
```

Preferably, the `ctx` file is located in the style map in your ConTEXt distribution.

Templates

During proofing you have to edit the XML snippets. It is a good custom to have templates at your disposal that you can insert whenever necessary.

The template file to be used is defined in the XML snippet by means of a directive:

```
<?context-directive job ctxtemplate m4all-template.lua ?>
```

In SciTE, the template is invoked by hitting the Ctrl-I key; then you can select the template you want to insert.

The project specific-template file is a lua file that could look like this:

```
return {
  xml = {
    {
      name     = "itemize",
      nature   = "display",
      template = '<itemize number="n" type="packed">\n
<item>?</item>\n <item></item>\n</itemize>',
    },
    {
      name     = "paragraph",
      nature   = "display",
      template = '<p>?</p>',
    },
    {
      name     = "italic",
      nature   = "inline",
      template = '<i>?</i>',
    },
  },
}
```

A template can be of `nature = "display"` or `inline`. The question mark indicates the location of the cursor after inserting the template in the text editor.

Preferably, the template file is stored in the project map in the ConTEXt distribution.

Aggregation

Early on in the project, we decided to store the content in small identifiable educational units like: `<exercise>`, `<theory>`, `<example>`, `<background>`, etc. and that we would aggregate these into subjects, sections and chapters.

An average aggregated chapter XML file contains some 10,000 lines so I think this decision was justified.

When you type:

```
\ctxlua{xml.save(lxml.id("main"),"the-content.xml")}
```

in the right location in the style file, the file `the-content.xml` is exported.

This file is used for inspection reasons mostly. It enables you to answer the question: “Is the XML content really there?” or “What am I really processing?”

Visual alarms

For a more refined proofing, you can use the ConTEXt feature:

```
\showsuspects
```

Let’s give an example:

> Twee hardlopers op de 400 meter trainen samen. Op basis van vele wedstrijden gaat de trainer uit van een normaal verdeeld snelheidsverschil tussen deze twee van gemiddeld 0 m/s met een standaardafwijking van 0,205 m/s.De laatste zes wedstrijden die deze twee hardlopers samen hebben gelopen , geven de volgende snelheden.

In this case you are warned of e. g. the space before the last comma.

Interaction

In order to be able to proof hyperlinks (websites, applets), internal references to pages or numbered text blocks, you can make the PDF interactive.

This is done in your style file with:

```
\setupinteraction[state=start]
```

Processing will take somewhat longer, but it is indispensable in the proofing trajectory.

The other stuff I do: production

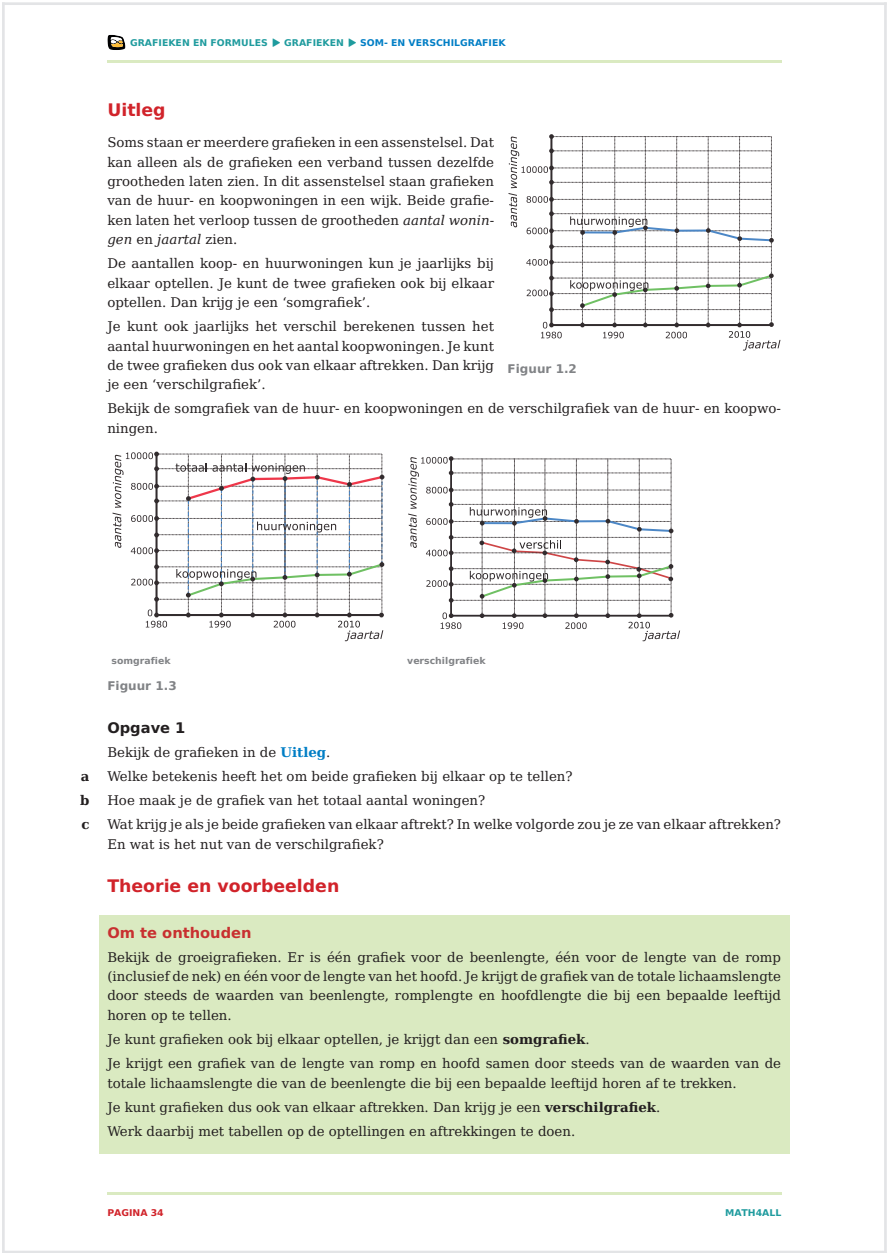
When you use ConTEXt, it is said that typesetting is done automatically. Well, in types of projects where math, images and text have to merge into a consistently made PDF document, automated typesetting is more or less irrelevant.

This is illustrated by the fact that, on the average, a 50-page A4 chapter is processed 55 times before it looks okay.

During production, I check every PDF product visually. Below I describe the main issues.

Images

First of all I check if the images are not too small or too big. For the web their natural width is mostly sufficient. For the folio products readability is a good indicator. I have a number of float locations at my disposal: like `here` and `right` (a sidefloat), and sometimes I combine images into one float.



GRAFIEKEN EN FORMULES ▶ GRAFIEKEN ▶ SOM- EN VERSCHILGRAFIEK

Uitleg

Soms staan er meerdere grafieken in een assenstelsel. Dat kan alleen als de grafieken een verband tussen dezelfde grootheden laten zien. In dit assenstelsel staan grafieken van de huur- en koopwoningen in een wijk. Beide grafieken laten het verloop tussen de grootheden *aantal woningen* en *jaartal* zien.

De aantallen koop- en huurwoningen kun je jaarlijks bij elkaar optellen. Je kunt de twee grafieken ook bij elkaar optellen. Dan krijg je een 'somgrafiek'.

Je kunt ook jaarlijks het verschil berekenen tussen het aantal huurwoningen en het aantal koopwoningen. Je kunt de twee grafieken dus ook van elkaar aftrekken. Dan krijg je een 'verschilgrafiek'.

Figuur 1.2

Bekijk de somgrafiek van de huur- en koopwoningen en de verschilgrafiek van de huur- en koopwoningen.

Figuur 1.3

Opgave 1

Bekijk de grafieken in de **Uitleg**.

a Welke betekenis heeft het om beide grafieken bij elkaar op te tellen?

b Hoe maak je de grafiek van het totaal aantal woningen?

c Wat krijg je als je beide grafieken van elkaar aftrekt? In welke volgorde zou je ze van elkaar aftrekken? En wat is het nut van de verschilgrafiek?

Theorie en voorbeelden

Om te onthouden

Bekijk de groeigrafieken. Er is één grafiek voor de beenlengte, één voor de lengte van de romp (inclusief de nek) en één voor de lengte van het hoofd. Je krijgt de grafiek van de totale lichaamslengte door steeds de waarden van beenlengte, romplengte en hoofdlengte die bij een bepaalde leeftijd horen op te tellen.

Je kunt grafieken ook bij elkaar optellen, je krijgt dan een **somgrafiek**.

Je krijgt een grafiek van de lengte van romp en hoofd samen door steeds van de waarden van de totale lichaamslengte die van de beenlengte die bij een bepaalde leeftijd horen af te trekken.

Je kunt grafieken dus ook van elkaar aftrekken. Dan krijg je een **verschilgrafiek**.

Werk daarbij met tabellen op de optellingen en aftrekkingen te doen.

PAGINA 34 MATH4ALL

Figure 1. Example folio page

Tables

I check tables and adjust the width of table cells wherever necessary for the folio product. I use the HTML table format:

```
<table border="0">
  <tbody>
    <tr>
      <td colspan="1" rowspan="1" paperwidth="1cm">A</td>
      <td colspan="1" rowspan="1">9,2</td>
      <td colspan="1" rowspan="1">9,2</td>
    </tr>
    <tr>
      <td colspan="1" rowspan="1" paperwidth="1cm">B</td>
      <td colspan="1" rowspan="1">9,2</td>
      <td colspan="1" rowspan="1">9,0</td>
    </tr>
  </tbody>
</table>
```

With the attribute `paperwidth` the cell width in the folio product is set.

Line Breaks

I correct overfull boxes created by math expressions that don't break. This is done with a line break directive:

```
<?context-directive injector crlf book ?>
```

Page Breaks

Page breaks are forced with yet another directive:

```
<?context-directive injector page book ?>
```

Units

One particular aspect of proofing is keeping values and units together at line breaks. For that purpose, a non-breakable space is inserted.

```
<am>A = 13</am>␣cm<sup>2</sup>
```

In SciTE the non-breakable space (here: ␣) is visualized by an orange rectangle ( ).

Paper Efficiency

One of the main concerns during production is limiting the number of pages. Or in other words, to limit white space in general and the vertical white space around images and tables (sidefloats).

For example the `\testpage[5]` for exercises is set at 5. This works perfectly fine but... Many exercises have an introduction that contains a sidefloat. So at a page

break, the label **Exercise 13** is often widowed while the exercise itself is at the next page. A manual page break before the exercise is therefore required. Or you can play a bit with:

- the width/height of the image
- the vertical white space on rest of the page

Since I am not allowed to change the order of exercises, the content itself or the images, my options are very limited.

Conclusion

When you start working on an extensive project, please check if there are features and facilities in ConTEXt and its supporting programs that can be of help. And:

- optimize your workflow by selecting the right tools
- use SyncTEX (if possible)
- use a project-specific word list for spell checking
- use a project-specific hyphenation list
- create proofing and production modes
- make your own templates
- check if you can use partial runs
- use visualization features

Photo: Harald König