Maarten Wisse

xml

Hacking TEX4ht for XML Output The Road towards a TeX to Word Convertor

abstract
This article explains how the author employs the TEX4ht convertor to manage multiple format (XML and PDF) output from a single LATEX source by writing a TEX4ht configuration file and a LATEX class file. Furthermore, it is explained how TEX4ht and the new OpenOffice package can be used to create a new LATEX to MS Word convertor.

Introduction

XML is a hot item in today's IT world. In the area of TeX typesetting, most attention is paid to processing XML input by the TeX typesetter. This is no surprise, because on the negative side, XML input capable typesetters are rare, and on the positive side, TEX does an excellent job in typesetting on demand. Among the mainstream command sets for TEX, ConTEXt seems to have most extensive support for XML output by its recent addition of the XML tag mapping mechanism to the publicrelease.[1]
However, in this contribution, I would like to show that TEX, and more specifically LATEX, can do an excellent job in generating XML output as well. Although originally built as a TEX to HTML converter, the TEX extension TEX4ht can be tweaked in such a way that it outputs arbitrary document type definitions. At this moment, extensive support is contained in the standard distribution for XHTML and MathML, and limited support for the Docbook and TEI DTDs. It is well known that installing and configuring TEX4ht is by no means an easy task. The programming interface is not always intuitive, documentation fragmentary and debugging information rather minimal. Hence, hacking on it is not for the fainthearted, but it must be added that the author of the system is very generous in providing information, solving problems and contributing code in case one does not find a way out of the labyrint.

The Story: What I Wanted and Why

In this article, I want to present two cases in which hacking TEX4ht was valuable for me. As an editor of *Ars Disputandi*, an academic ejournal for philosophy of religion,[2] I developed a configuration file for generating XML output according to the DTD used by the Roquade Publishing project, a joined project of the academic libraries of Utrecht University and Delft University of technology.[3] This was the main reason why I took the effort to learn hacking TEX4ht. Once I managed that problem, I saw a way out of a problem which had bothered me for a long time already: TEX to Word conversion.[4]
Notwithstanding the exciting experience which my use of TEX has been for almost three years, the lack of a good converter – supporting the very special BibTEX configuration I use – from TEX to Word gave rise to numerous problems each time when I needed to

1. See Berend de Boer, 'From Database to Presentation via XML, XSLT and ConTEXt', in: Simon Pepping (editor), *TEX and META: The Good, the Bad and the Ugly*, EuroTEX proceedings 2001, 27–39.
2. `http://www.arsdisputandi.org`.
3. See `http://www.roquade.nl`.
4. All three configuration files (roqart.cls, roqart.4ht, and ooffice.4ht), can be obtained from my website: `http://www.pmwisse.myweb.nl`.

more or less officially publish research articles in today's WWW: MSWord Wide World. Someone who read an article written by me and published on the basis of a very provisional TEX to Word conversion recently told me that the negation sign in one of the crucial definitions in that article is missing, apparently caused by the conversion.[5] That's a pity—obviously. Furthermore, my provisional way of converting involved manual regeneration of all footnotes – many, in our area of research – in the resulting Word version, something which is time consuming and prone to errors. The phenomenon of 'footnotes' will return in this article.

The story about the Roquade project is somewhat different. The aim of the project is to bypass the commercial publication of academic knowledge by giving the publishing process back in the hands of the university. The project does this by providing a technical XML based epublishing infrastructure which enables scientists to initiate fresh epublishing projects, mainly journals. The long term aim of the project is that the XML generation process is simplified in such a manner that the academic staff could handle it with a minimal amount of training. The current situation is that most XML generation is done by people of the project and this is currently done by linking template based styles in MS Word to XML tags, a conversion which is carried out by a freely available tool Majix. A web based publishing management tool provides editors with XML uploading facilities, followed by a XSLT based HTML output.

As a known techie among the staff of the project, it was suggested that I should try to generate the XML source myself instead of the library staff, much in line with the eventual aim of the project. This is in fact a major advantage for me as an editor, because it provides me with full control over the publication process. I initially started using the Word-Majix procedure, but this quickly turned out to be problematic. First, the procedure lacked necessary robustness, primarily in connection with – yes, again – footnotes. The connection between the styles in the template on the one hand and the XML tags on the other was easily broken by, for instance, a foreign origin (notably WordPerfect) of the Word file. This took me some afternoons to get the XML out of Word. Moreover, the desire to have full and rapid control over the publishing process was hindered by the fact that for every Word to XML conversion, I needed access to a MS Windows machine, because all of my other activities are Linux based.[6] Finally, the desire to have PDF versions of all articles brought TEX into focus as a tool which might generate XML as well as PDF from the same source.[7]

The Roquade Case

I'm not going to repeat all details of the configuration process. The basic steps of tweaking TEX4ht output, along with a special appendix which explains the setup of a new DTD from scratch, can be found in the *The LaTeX Web Companion*.[8] I will only mention those steps which differ from the *Web Companion*. In this section, I will deal with the high level issues, whereas in the final section, I will explain some low level clues which might help people out when developing a new configuration.

The basic thing as explained in the appendix of the Companion is that one creates a .cfg file which fills those hooks needed for a particular setup.[9] I decided to take a twofold

5. Maarten Wisse, 'The Authority of the Bible', *Religious Studies* 36 (2000), 479.
6. When it comes to hardware requirements, an additional advantage of my current TEX based setup is that I'm able to edit everything on my recently purchased Intel 386SX Toshiba T2200SX notebook :-)
7. Yes, I know that the best solution to this problem is to generate both HTML and PDF from the XML source, but due to a lack of human resources among the technical staff of the Roquade Project, this is not to be expected in the near future.
8. Michel Goossens, Sebastian Rahtz et al., *The* LATEX *Web Companion* (Reading Mass.: Addison Wesley, 1999, 164–184, 404–415.
9. Goossens, Rahtz, et al., *Web Companion*, 404–408.

approach to my problem. On the one hand, I developed a LATEX class file roqart.cls which should take care of the markup of the PDF version of the file. This class file loads article.cls, adding some commands such as `\journalvolume` and `\journalyear`. Furthermore, it automatically generates the articleheading by the `\makehead` command, similar to `\maketitle`.[10] On the other hand, I made a file roqart.cfg which contained all the TEX4ht configuration hooks. Thus, I kept XML and PDF configuration completely separate, as well as completely hidden from the user.[11]
Let me explain my approach by example. The preamble of a Roquade article might look like this:

```
\documentclass{roqart}
\journalvolume{2}
\journalyear{2002}
\title{The Title}
\subject{The Subtitle}
\author{Firstname}{Lastname}
\authorsemail{my@email.com}
\affil{Famous University, UK}
\begin{document}\makehead
```

The remainder of the file is standard LATEX. The definition of the `\makehead` command shows how the class file takes care of the different output formats. By an if statement which checks whether pdfoutput is true or false, the class file decides what kind of `\makehead` command to expand. In case of PDF output, the `\makehead` command looks as follows:

```
\newcommand{\makehead}{%
  \vspace*{-1.8cm}\noindent\hspace*{-3.5cm}
  \parbox[b][36pt][t]{5cm}{
    \small\raggedleft\journalname\\Volume \@jrnvol~\@jrnyear\\\issn
    }
  \hspace*{6pt}
  \parbox[b][36pt][b]{6cm}{
    \@logo
    }\par
  \vspace*{36pt}
  \noindent\hspace*{-2.5cm}
  \parbox[t][\height][t]{4cm}{
    \normalsize\raggedleft\itshape\firstname\ \lastname\\
    \footnotesize\raggedleft\scshape\MakeLowercase{\@affil}
    }
  \hspace*{6pt}
  \parbox[t][\height][t]{13cm}{
    \LARGE\raggedright\@title\\
    \vspace*{12pt}
    \ifx\@subject\@undefined\else
    \large\@subject\\
    \fi
    \ifx\@intro\@undefined\else
    \vspace*{24pt}
    \normalsize\@intro
    \vspace{24pt}
    \fi
    }

  \thispagestyle{firstpage}
  \setcounter{footnote}{0}
  }
```

10. The reason that I did not use `\maketitle` is that TEX4ht uses many special configurations for that command, which make it very difficult to modify.
11. All package loading, header definition etc. is carried out by the class file as well.

For PDF output, the command generates specific markup. It even automatically adds the logo of the journal on top of the page. When normal LATEX – the basis of the TEX4ht run – is generated, the `\makehead` command looks like this:

```
\newcommand{\makehead}{%
\PreHead\par\PreTitle \@title \PostTitle\par
\ifx\@subject\@undefined\else
\PreSubject \@subject \PostSubject\par
\fi
\ifx\@intro\@undefined\else
\PreIntro \@intro \PostIntro\par
\fi
\PreFirstname \PreEmail\firstname\ \PostFirstname \PreLastname \lastname\PostEmail
\PostLastname\par
\PreAffil \@affil \PostAffil\par
\PreKeywords \@keywords \PostKeywords\par
\PostHead\par
}
```

The 'Pre' and 'Post' commands shown in this definition are actually defined as empty elsewhere in the class file. Hence, a normal LATEX run on them will result in a heading with simple lines of text without special markup. However, when run through TEX4ht, these commands will be redefined to XML tags in the file roqart.cfg, resulting in the following XML output of the `\makehead` command:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE article PUBLIC "" "article.dtd" [
<!ENTITY tab ' '>
]>
<article>
   <h1><ht>The Title</ht>
   <subject>The Subtitle</subject>
   <authors><link url="mailto:myname@email.com">Firstname Lastname</link></authors>
   <biography>Fancy University</biography>
```

Given a valid LATEX file, generating PDF and XML files of a Roquade article is carried out by typing the following on the command line:

```
maarten@FT190 > pdflatex filename
maarten@FT190 > roqlatex filename 'roqart,html'
```

The latter command executes a shell script which invokes TEX4ht with the appropriate Unicode fonts. This is basically the way it works. It turned out that the procedure described in the Companion, using a .cfg file with the configuration, didn't work for my setup because it leaves certain internal TEX4ht constructs undefined which are needed by my configuration. Therefore, Eitan Gurari offered me a different way of adding new configurations by making native .4ht files. These are loaded *after* the initial configuration of TEX4ht, and therefore, all basic constructs are available for redefinition.[12]

The TEX to Word Case

Because my TEX to Word convertor was intended to handle the source code of my PhD dissertation files, it had to obey the rules of the standard article class. No new class file was needed. The problems of this case appeared in another area. Let me first describe how I managed the TEX to Word case, because in fact, my convertor is not a real Word convertor, but rather a TEX to OpenOffice convertor. OpenOffice is the name of the open

12. More information about making .4ht files and the requirements which need to be met in order to function properly can be found in the file mktex4ht.4ht in the TEX4ht distribution.

source project behind Sun's recently acquired StarOffice suite.[13] As soon as I decided to learn XML hacking on TEX4ht, I realized that this was basically important for my conversion problem because of two reasons:

- StarOffice 6.0/ OpenOffice 1.0 would get XML based file formats.
- SO/OO is very good at MS Word conversion.

Hence, if I could succeed in letting TEX4ht output OpenOffice XML, I would basically have Word output as well. The StarOffice 6.0 beta was of excellent quality when it came to reliability and conversion to Word, so that the experiment was worth the job.

The idea was simple. I created a Open Office document to which I added all the markup features which I normally use in my TEX documents, i.e. footnotes (sic!), italics, bulleted and enumerated lists, section, and subsection headings. For the sake of completeness, I also added small caps, boldface, and superscript. I saved that file and began to investigate it in order to sort out the format of the XML.

Now, the most difficult problem emerged, because StarOffice/ OpenOffice does not output a plain XML file – which my TEX4ht convertor does – but, as I figured out later on, a zipped file of a particular structure. At this stage, the open source character of the OpenOffice program became important. On the Open Office site, I found, first of all, a description of the file format, and secondly, an in depth reference manual of the XML DTD at the basis of OpenOffice/StarOffice. I found out that the XML generated by StarOffice is split up into several files, among which are content.xml, styles.xml, and meta.xml, which respectively store the content, styles, and metadata of the file.

The OpenOffice.org site told me as well that StarOffice/OpenOffice is not currently able to read plain XML file conforming to their own DTD. It only reads the zipped files. This seemed the end of my experiment, but it was not. The solution I found was basically simple and obvious. I unzipped my OpenOffice file containing all markup I needed to a new directory, wrote a configuration file ooffice.4ht[14] for TEX4ht producing OpenOffice XML output as found in the content.xml file, replaced the existing content.xml file by the result of the TEX4ht compilation, zipped the file, and loaded it into OpenOffice again. That worked. Of course, some initial errors appeared, but OpenOffice enormously helped me by telling exactly on which line, which character it found an error in the content.xml input, making the debugging process as easy as possible. Furthermore, the seemingly laborious process of replacing the content.xml file and rezipping the directory into the complete OpenOffice file was easily simplified by a script which automates these tasks.

Of course, my convertor is by no means what people might expect from a full featured TEX to Word convertor. At this moment, it does not support images, tables, formulas, columns etc., although the fact that OpenOffice builds upon MathML means that Eitan Gurari could easily integrate TEX4ht's existing MathML support into ooffice.4ht. The same goes for SVG support. The main thing which ooffice.4ht brought me is native footnote conversion. My `\footnote` commands in LATEX become 'real' footnotes in Open Office and, subsequently, footnotes in Word as well. Let me show the configuration hook in ooffice.4ht which does that job:

```
\Configure{footnote}
 {\HCode{<text:footnote text:id="ftn}\FNmark\HCode{"><text:footnote-citation>}\FNmark}
 {\HCode{</text:footnote-citation><text:footnote-body>
   <text:p text:style-name="Footnote">}}
 {\HCode{</text:p></text:footnote-body></text:footnote>}}
```

13. The OpenOffice Project can be found at `http://www.openoffice.org`. Initially I worked with the StarOffice 6.0 beta, but because the StarOffice suite will be released under a commercial license, I switched to the OpenOffice 641C (by now 1.0) version. As far as I can see, the solution will contiue to work with both as long as they keep the same DTD.
14. Ooffice.4ht is actually an elaborated variant of roqart.4ht.

The first and the second argument are put before the footnote text, and the third argument will be put after it. The `\FNmark` command you see in the first argument is an internal TEX4ht command which contains the current footnote mark. It is used twice because OpenOfice adds a `text:id` attribute to the opening tag of the footnote which receives the number of the footnote minus 1. Simply the number of the footnote worked as well. Apart from the footnote support, TEX4ht's reliance upon a real TEX run ensures that all of my special Jurabib formatting features are supported.[15] Finally, TEX4ht and Open Office's extensive Unicode support – although initially somewhat buggy in TEX4ht – means that even Babel based Greek text is properly converted into StarOffice Greek text. The same goes for Hebrew.

Some Clues

So far, I hardly showed any actual TEX4ht configuration code when explaining my XML setups. The reason for this is that it would take huge amounts of space to explain the details of TEX4ht configuration. Much of basic TEX4ht configuration is explained in the LATEX *Web Companion*, and many of the configuration hooks in roqart.4ht and ooffice.4ht are rather simple applications of the information in that book. However, due to the fact that some aspects of configuring TEX4ht are pretty counter-intuitive, I think that it could help users out of a difficult setup process when I explain some tricky issues which are not in the *Web Companion* and nevertheless useful.

The basic idea behind TEX4ht is that it redefines standard LATEX commands in such a way that they receive pre- and post hooks which the user can modify by the `\Configure` command. For example, the standard LATEX command `\textit` is redefined in such a way by TEX4ht that it can be configured as follows (excerpt from roqart.4ht):

```
\Configure{textit}{\Tg<i>}{\Tg</i>}
```

This means that when processed by TEX4ht, the output is not presented in italics (whatever that may be in ASCII!), but preceded by a `<i>` tag and followed by a `</i>` tag, which makes up italics in the Roquade DTD.

This is of course a very simple example. A much more difficult example is the 'paragraph' tag, i.e. the tag preceded and followed by each normal paragraph seperated by the typical TEX-ic blank line. The paragraph hook is a command requiring four arguments, respectively representing the tag before a normal paragraph and an indented paragraph, and the tag after a normal and an indented paragraph. Hence, taking an example from the ooffice.4ht configuration file, one finds the following definition:

```
\Configure{HtmlPar}
   {\EndP\HCode{<text:p text:style-name="Text body">}}
   {\EndP\HCode{<text:p text:style-name="Text body">}}
   {\HCode{</text:p>\Hnewline}}     {\HCode{</text:p>\Hnewline}}
```

The first two arguments contain the strange code `\EndP` appears, a code which is omnipresent in TEX4ht files and typically counter intuitive. Given the name 'EndP' one would expect that it appears in the last two arguments of the paragraph hook, but it does appear at the start of the first two. The *Web Companion* explains that 'The task of `\EndP` is typically to deliver code from the start of a paragraph to its end.',[16] but this does not help us much further.

In some way or another – Eitan had his reasons for it, of course[17] – the `\EndP` command works from the start of a new paragraph in order to terminate the *previous* paragraph and place its closing tag (e.g. `</text:p>`). Hence, the `\EndP` is placed in the arguments of the

15. For more information about Jurabib, see `http://www.jurabib.org`.
16. *Web Companion*, 183.
17. Eitan: 'The reason is that TEX offers access to the start of paragraphs (through `\everypar`) and not to their ends :-('

pre tags in order to place the post tag of the previous paragraph when a new paragraph is started. A similar trick is used for the `\IgnorePar` command, which is frequently found before or after `\EndP` commands in TEX4ht configuration files. This command, used in conjunction with `\EndP`, lets the previous paragraph end and starts a new one, but dumps the pre and post code of that paragraph, which would normally appear. Let me finally give two code examples which illustrate this trick:

```
\renewcommand{\PreSubject}{\EndP\IgnorePar\HCode{<subject>}}
\renewcommand{\PreIntro}{\EndP\IgnorePar\HCode{<intro>}}
\renewcommand{\PreKeywords}{\EndP\IgnorePar\HCode{<keywords>}}
\renewcommand{\PostSubject}{\HCode{</subject>}}
\renewcommand{\PostIntro}{\HCode{</intro>}}
\renewcommand{\PostKeywords}{\HCode{</keywords>}}
```

These are the redefnitions which fill up the empty commands defined in roqart.cls with the appropriate XML code for the Roquade DTD. When we look at the code in roqart.cls, we see that every line is terminated by a `\par` command. In the code from roqart.4ht, we see that every pre command reckognises this `\par` command by executing a `\EndP` command. However, given the fact that the default paragraph hook has been configured to start each paragraph with a <p> and corresponding </p> tag, this would mean that the tags on top of the article would get it as well, which is not correct in this DTD. Therefore, each pre command gets a `\IgnorePar` command as well which drops the <p> and </p> code.

Finally, let's go into a really difficult problem from a TEX4ht configuration perspective, i.e. configuring a simple itemized or enumerated list. We take the configuration of the itemize environment from roqart.4ht as an example. The XML code of an itemized list according to the Roquade DTD is kept very simple. It looks like this:

```
<list type="itemized">
<item>This is an item</item>
<item>This is another one</item>
</list>
```

The TEX4ht code which produces this XML code looks as follows:

```
\ConfigureList{itemize}%
   {\EndP\HCode{<list style="bullet">\Hnewline}\def\end@Item{}}
   {\EndP\HCode{</item></list>}\ShowPar}
   {\EndP\end@Item\DeleteMark}
   {\HCode{<item>}\par\ShowPar \def\end@Item{\Tg</item>}}
```

The *Companion* describes the `\ConfigureList` command as:

```
\ConfigureList{name}{pre-list}{post-list}{pre-label}{post-label}
```

The different lists of LATEX carry the hooks in the following manner:

```
<pre-list hook>
<pre-label hook> MARK <post-label hook> CONTENT OF ITEM 1
<pre-label hook> MARK <post-label hook> CONTENT OF ITEM 2
............
<pre-label hook> MARK <post-label hook> CONTENT OF LAST ITEM
<post-list hook>
```

To allow marking at the end points of the content of the items (e.g., `</item>`), we must attach those marks to the `<pre-label hook>` and `<post-list hook>`. However, the first `<pre-label hook>` should not carry an end mark. Moreover, in the boundary case of empty lists with no specified items, the `<post-list hook>` should also avoid producing such an end mark. The `\end@Item` helps handling these cases correctly, and the `\EndP` takes care of in-time providing the closing paragraphs for those opened within the items.

In some cases, like in itemized and enumerated lists, but unlike in the cases of description lists, we don't want the marks provided by LATEX. Instead, we expect them to be created by the interprets of the XML code. The `\DeleteMark` comes handy here.

The LATEX model of paragraphs is not always in line with the model of paragraphs in the XML standard in use. The `\IgnorePar` and `\ShowPar` have been introduced to help bridge the differences. The `\par\ShowPar` forces a start of paragraphs at the start of each item, allowing to introduce the `<p>` tag there. (The LATEX model allows paragraph breaks withing the content of items, but not at their start points.)[18]

Conclusion

It might have become clear to the reader that the configuration of TEX4ht is no easy task. However, I hope that it is clear as well how extremely useful the system can be once properly configured. Equipped with this convertor, I manage both an e-journal and TEX to OpenOffice/MS Word conversion with little effort.[19]

18. Thanks to Eitan for offering this explanation of the `\ConfigureList` problem.
19. I would like to thank Eitan Gurari for his contribution to and comments on this article.