¶ **Narzędzia**

Configuring TEX4ht to produce XML

Michel Goossens & Sebastian Rahtz

The basic idea of LATEX is interfaces for identifying structural entities in documents and for associating presentations with such entities. SGML and its descendants aim at a generalization of such an idea in which the interfaces are provided in formats suitable for further processing. The ultimate objective of TEX4ht is to offer a tool that recognizes TEX-based interfaces in general, and of LATEX in particular, and that provides the means to translate these interfaces into arbitrary SGML-based representations.

With this in mind, as we saw in Chapter 4, TEX4ht delegates to the native TEX compiler the task of processing the source files, relieving itself from dealing with many painful details that have little to do with structural issues in documents. To identify the structure of a document, TEX4ht seeds the style files with hooks at strategic locations within the definitions of macros and then offers the means to configure the hooks to produce any sort of output you want. This means that you can create translations to different SGML-based representations.

Section 4.5 showed how the hooks can be configured. We now go on to describe a friendly interface for identifying and configuring hooks and then wander more deeply into low-level features associated with the TEX compiler.

The reader is encouraged to get a live demonstration of the interface discussed in the next two sections, by actually running the different stages of the example.

1. Starting from scratch

The fast evolution of HTML between different versions in recent years and the emergence of the XML language highlight the importance of allowing for large-scale modifications to the output of TEX4ht, to meeting the challenges that arise from new requirements. Let us look at how to strip TEX4ht back to basics and then build up a new converter.

1.1. Loading empty configurations

The package option `0.0` requests empty configurations for the hooks, implying an output consisting of text with no hypertext tags. For example, the following source document outputs just the text `1 Demo With A List` without any tags:

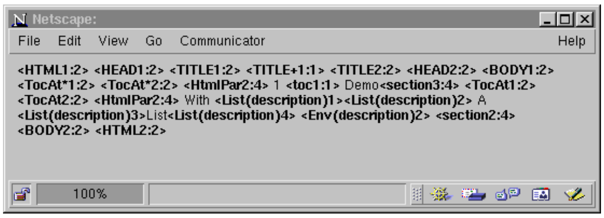
```
\documentclass{article}
  \usepackage[html,0.0]{tex4ht}
\begin{document}
\section{Demo}
With
  \begin{description}
    \item[A] List
  \end{description}
\end{document}
```

We will keep this simple example in mind for the next few sections in order to demonstrate the outcome of the features discussed.

1.2. Looking for hooks

A major problem in reconfiguring the TEX4ht hooks is to find out where they are and what they can do. The option `hooks` gives us a good start by requesting *pseudohypertext tags* for hooks that have no other configuration. A *pseudohypertext tag* shows the name of a hook, the index of an argument within the hook, and, when it is not obvious, the number of arguments for the hook.

Compiling the example source document with the `html,0.0,hooks` package options will produce output that looks like this in a browser:



In this display, the tag `<HTML1:2>` comes from the first argument of the “HTML” hook. That hook is configured with the command

> `\Configure{HTML}` {*first-argument*} {*second-argument*}.

The `<Env(description)2>` tag comes from the second argument of the `description` environmental hook, configured by the command

\ConfigureEnv{description}
{*first-argument*} {*second-argument*}
{*third-argument*} {*fourth-argument*}

The `<List(description)4>` tag comes from the fourth argument of the description list hook, configured by the command

\ConfigureList{description}
{*first-argument*} {*second-argument*}
{*third-argument*} {*fourth-argument*}

1.3. Viewing the hooks

While current browsers are likely to display the output resulting from the hooks option, the file itself is missing a proper HTML code at the start and the end of the file. That code can be quite easily introduced by configuring the "HTML" hook. We can also give the tags special characteristics within the display to attract the attention of the reader to their presence. The appearance may be modified within the CSS file, as well as by changing the following default configuration:

```
\Configure{hooks}
  {\HCode{<STRONG CLASS="hooks">&lt;}}
    {\HCode{&gt;</STRONG>}} {}{}
```

By introducing a configuration file try.cfg as follows and loading it with the option list try,html,0.0,hooks, the HTML file will become legal, and the tags will shown in green.

```
\Configure{HTML}{\StartHtml}{\EndHtml}
\newcommand\StartHtml{\IgnorePar\HCode{%
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
  Transitional//EN">\Hnewline
  <HTML><HEAD><LINK REL="stylesheet"
  TYPE="text/css" HREF="\jobname.css">\Hnewline
  <TITLE>\jobname</TITLE></HEAD>\Hnewline
  <BODY>} \Css{.hooks{color:green;}}}
\newcommand\EndHtml{\HCode{</BODY></HTML>}}
\Preamble{}
\begin{document}
\EndPreamble
```

1.4. Reconfiguring the hooks

Starting from output consisting of *pseudohypertext* tags, you can gradually configure the different hooks to get the output you want. Along the way, distracting tags of no interest can be removed by assigning invisible nonempty code to their corresponding parameters.

By configuring the hooks "BODY", "HEAD", "TITLE+", "TITLE", "TocAt*", "TocAt", and "toc" to add the \empty to their parameters, the example reduces to the following:

```
1 Demo<section3:4> With
<List(description)1> <List(description)2> A
<List(description)3> List <List(description)4> <Env(description)2>
<section2:4>
```

With straightforward configurations supplied to the remaining hooks, you obtain a view like this:

```
[DOC] [SECTION][NUM]1[/NUM][TITLE]Demo[/TITLE]
With [DLIST] [MARK] A [/MARK]List
[/DLIST] [/SECTION][/DOC]
```

A configuration file which produces output like this is as follows:

```
\Configure{HEAD}{\empty}{\empty}
\Configure{HTML}
  {\IgnorePar\HCode{<HTML><HEAD>
     <TITLE></TITLE></HEAD><BODY>}[DOC]}
  {[/DOC]\HCode{</BODY></HTML>}}
\Preamble{}
   \Configure{BODY}{\empty}{\empty}
   \Configure{TITLE}{\empty}{\empty}
   \Configure{TITLE+}{\empty}
   \Configure{TocAt}{\empty}{\empty}
   \Configure{TocAt*}{\empty}{\empty}
   \Configure{toc}{\empty}
   \ConfigureEnv{description}
     {[DLIST]}{[/DLIST]}{}{}
   \ConfigureList{description}
     {}{}{[MARK]}{[/MARK]}
   \Configure{section}
      {[SECTION]}{[/SECTION]}
      {[NUM]\arabic{section}[/NUM]
       [TITLE]}{[/TITLE]}
\begin{document}
\EndPreamble
\Configure{HtmlPar}{}{\empty}{}{}
```

1.5. Cleaning up

Having completed configuring the hooks of interest, we can remove the code we introduced just for getting an improved view of the output, and we can take out the hooks option in the \usepackage command.

2. Adding XML tags

The XML language is good for exhibiting how TEX4ht can be reconfigured, so we will now look at adding some XML tags to our output. As long as the newly assigned configurations to the hooks do not contain HTML tags, the browsers should have no problem displaying the outcome of these assignments.

TEX4ht provides the three commands:

```
\Tg<name>
\Tg</name>
\Tg<name/>
```

for producing start, end, and empty XML tags, respectively. The edit package option makes these new commands produce visible forms of the tags by introducing code for *displaying* the tags instead of actually *creating* them.

For example, replacing the [*name*] with commands of the form \Tg<*name*> in the configuration file of our example, we get a view like this in HTML browsers:

```
<DOC> <SECTION> <NUM> 1</NUM><TITLE>Demo</TITLE>
With <DLIST> <MARK> A </MARK>
List </DLIST> </SECTION> </DOC>
```

using the following version of the configuration file:

```
\Configure{HTML} {\Tg<DOC>}{\Tg</DOC>}
\Preamble{}
  \ConfigureEnv{description}
    {\Tg<DLIST>}{\Tg</DLIST>}{}{}
  \ConfigureList{description}
    {}{}{\Tg<MARK>}{\Tg</MARK>}
  \Configure{section}
    {\Tg<SECTION>}{\Tg</SECTION>}
      {\Tg<NUM>\arabic{section}
       \Tg</NUM>\Tg<TITLE>}{\Tg</TITLE>}
\begin{document}
\EndPreamble
```

The default look of the visible "tags" is set by the command

```
\Configure{edit}{\HCode{<STRONG>&lt;}}
  {\HCode{&gt;</STRONG>}}
```

Removing the edit option would make a proper XML document.

2.1. Typesetting the abstract tags

The job of the XML tags is to identify logical units within the document. During the editing phase in which the tags are introduced, it can be very useful to highlight the nature of the different tags and the structural relationships they maintain.

To meet this objective, the \Tg tags can be enhanced with the commands

```
\Configure<name>{before}{after}
\Configure</name>{before}{after}
\Configure<name/>{before}{after}
```

for producing whatever sort of display you find useful.

The addition of commands like

```
\Configure<NUM>{\HCode{<DIV ALIGN="CENTER">}}{}
\Configure</TITLE>{}{\HCode{</DIV>}}
\Configure<DLIST>{\HCode{<BR>}}{}
\Configure</DLIST>{}{\HCode{<BR>}}
```

to the configuration file results in a layout like the following within the browser:

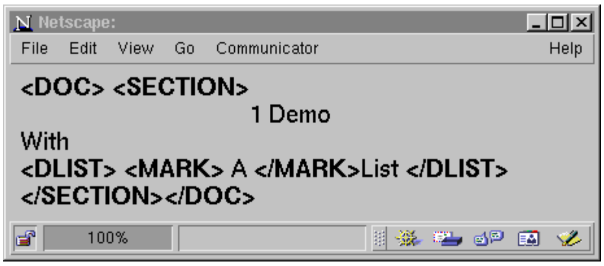
```
<DOC> <SECTION>
     <NUM>1</NUM><TITLE>Demo</TITLE>
With
<DLIST> <MARK> A </MARK>List </DLIST>
</SECTION> </DOC>
```

When the first argument is a hyphen character -, the configuration commands produce just the second argument without showing the tag itself. In this case, the configuration commands define a virtual browser for the new tags.

Consider the following variants of the configuration commands:

```
\Configure<NUM>-{\HCode{<DIV ALIGN="CENTER">}}
\Configure</NUM>-{ }
\Configure<TITLE>-{}
\Configure</TITLE>-{\HCode{</DIV>}}
```

These commands will produce a browser display like this:



2.2. Checking containment relationships

The verify option requests warning messages for unknown parent-child containment relationships among the tags defined by the \Tg command. Containment relationships can be specified by listing pairs of parent and child names with the string --> between the commands

```
\Verify
\EndVerify
```

Different pairs within the listing should be separated by a comma. Thus if

```
\Verify --> DOC, DOC --> SECTION \EndVerify
```

is placed before the \begin{document} of the configuration file try.cfg, and the package option verify is used, you will get warning messages like these:

```
4. --- warning --- SECTION --> NUM ?
4. --- warning --- SECTION --> TITLE ?
4. --- warning --- SECTION --> DLIST ?
4. --- warning --- DLIST --> MARK ?
```

All we have specified is that the `SECTION` tag is expected within a `DOC` tag. The system reports that it has found a `MARK` within a `DLIST`.

The package option `verify+` is an extension of the `verify` option in which the containment relations found by the system are detailed in the log file.

Similarly the `hooks+` package option is an extension of the `hooks` option. Besides requesting a display of how parameters of hooks are used, this option also asks for a listing in the log file of how hooks are configured. This is useful when looking at example configurations which may provide useful guidelines on how to define new configurations. Typically such examples are spread around different files and might be buried within other unrelated code.

A `\Tg`<*argument*> command is a special case of a `\HCode{`<*argument*>`}` command that allows us, during an editing phase, to show the tags and to check that they satisfy proper containment relationships. A weaker `\TG` version of the command is offered for instances where you do not care about verifying containment relationships. On the other hand, a `\tg` variant of the `\TG` command which corresponds to an <*argument*>, instead of to `\HCode{`<*argument*>`}`, is available.

3. Getting deeper for extra configurations

Our attention so far has been on structural features that apply mainly to large items such as chapters, lists, and tables. These entities are typically defined within style files; they do not really rely on the low-level features that are provided in the TEX typesetting engine. Now we need to look at some of the means available in TEX4ht for dealing with these low-level commands.

3.1. Math mode

Mathematical formulae are rich in low-level TEX features. The default settings of TEX4ht provide graphic versions of mathematical formulae that are likely to have complex structures. The package option `math` requests an alternative setup in which the formulae are seeded by hooks that can be configured to specify new output representations.

The mathematical environments `\(`*formula*`\)`, `\[`*formula*`\]`, `$`*formula*`$`, and `$$`*formula*`$$` can be configured with the following commands:

```
\Configure{()}{before$at-start}{at-end$after}
\Configure{[]}{before$$at-start}{at-end$$after}
\Configure{$}{before}{after}{at-start}
\Configure{$$}{before}{after}{at-start}
```

The configuration of the latter pair of environments applies also to the `math` and `displaymath` environments, respectively.

```
\Configure{()}{[equation]$}{$[/equation]}
\(a+b\)
```

[equation]a + b[/equation]

Some of the features we will look at require TEX4ht to be in a special mode of operation. The commands

```
\DviMath
\EndDviMath
```

may be used to set this up.

3.2. Math classes for symbols

At the lowest level of math reside the primitive tokens representing numbers, variables, operators, parentheses, and so forth. Inspired by the classification of mathematical symbols within TEX, a similar (but independent) classification is applied to symbols in TEX4ht.

Specifically the symbols might be assigned a class number between 0 and 9, where an initial classification puts large operators in class 1, binary operators in class 2, relational operators in class 3, opening delimiters in class 4, closing delimiters in class 5, punctuation symbols in class 6, and the rest of the symbols in class 0.

The following command may be used to associate output tags with the symbols of a particular class and to add symbols into the class:

```
\Configure{MathClass}{class-number}{string}
{before} {after} {symbols}
```

If the *string* argument is not empty, it must be a single character not appearing in the arguments *before* and *after*. In this case, those arguments specify a content to be inserted before and after the symbols of the *class*.

The argument *symbols* should be a sequence of symbols. When it is not empty, the symbols specified in the argument are assigned to the specified class.

```
\Configure{()}{\DviMath$}{$\EndDviMath}
\Configure{MathClass}{0}{*}
   {[ordinary]}{[/ordinary]}{}
\Configure{MathClass}{2}{*}
   {[operator]}{[/operator]}{}
\Configure{MathClass}{7}{*}
   {[digit]}{[/digit]}{0123456789}
\(a+1\)
```

[ordinary]a[/ordinary] [operator]+[/operator] [digit]1[/digit]

As when the arguments of `\HCode` are processed, the *before* and *after* are processed just for macro expansions and not for fonts, definitions, and computations.

The commands

```
\PauseMathClass
\EndPauseMathClass
```

can be used temporarily to suppress the contributions of the different classes.

3.3. Math classes for delimiters and words

Open and close delimiter symbols, which group tokens for the `\Send{BACK}` instructions (Section 3.4), should be assigned their math classes with the command:

```
\Configure{MathDelimiters}{open}{close}
```

When the package option `math` is used, the default setting invokes the command `\Configure{MathDelimiters}{(}{)}` to assign the parentheses symbols "(" and ")" to the math classes 4 and 5, respectively, and to inform TEX4ht about the pairing of these delimiters.

TEX4ht provides the commands `\mathord`, `\mathop`, `\mathbin`, `\mathrel`, `\mathopen`, `\mathclose`, and `\mathpunc` for associating, respectively, classes 0 through 6 to subformulae. The following instruction configures these commands, while indirectly referring to the commands through their class numbers:

```
\Configure{FormulaClass}{class-number}{string}
{before} {after}
```

If the argument *string* is not empty, it should be a character not appearing in the arguments *before* and *after*. If *string is* empty, we specify the configuration to be used for the characters of the specified *class-number*.

```
\Configure{()}{\DviMath$}{$\EndDviMath}
\Configure{FormulaClass}{4}{*}{}{[}
\Configure{FormulaClass}{5}{*}{]}{}
\(\mathopen{open}a+1\mathclose{close}\)
```

open[a + 1]close

3.4. dvi-based groups

The output of TEX is the low-level page-description language dvi, which relies heavily on a hierarchical grouping mechanism. That hierarchy reflects the structure specified explicitly and implicitly in the source document. You sometimes need access to this information when writing translators.

```
\Trace{GROUP}
\EndTrace{GROUP}
\Configure{GROUP}{string}{open-1} {open-2}
{close-1} {close-2}
```

The first two commands delimit the region in which the groups are to be shown. The third command controls the manner in which the groups are shown.

The arguments *open-1* and *open-2* specify what is to be inserted at the start of groups, and, if the first of these arguments is not empty, the level of nesting and the group number are also included. The arguments *close-1* and *close-2* specify in a similar manner the content to be inserted at the end of groups.

```
\Configure{GROUP}{*}{[}{]}{[/}{]}

\noindent \Trace{GROUP}
\(A \stackrel{\string~}{=} B\)
\EndTrace{GROUP}
```

A[3 480][4 481] =[/4 481] [4 482] [/4 482][/3 480] B

The next two commands enable us to submit content to the start and end points of the current group. The groups are identified by their level of nesting, relative to where the command appears. Level 0 refers to the group that immediately includes the `\Send` command. The higher levels refer to groups that follow the command.

```
\Send{GROUP}{level}{content}
\Send{EndGROUP}{level}{content}
```

```
\Configure{()}{\DviMath
    \Send{GROUP}{1}{[1]}
    \Send{EndGROUP}{1}{[/1]}
    \Send{GROUP}{2}{[2]}
    \Send{EndGROUP}{2}{[/2]}
  $}{$\EndDviMath}
\(A \stackrel{\string~}{=} B\)
```

A[1][2] =[/2] [/1] B

The command

```
\Send{BACK}{content}
```

may be used to send content backwards over the most recent symbol, group, or region enclosed between delimiters that have been declared a pair with the `\Configure{MathDelimiters}{open}{close}` command.

3.5. Subscripts and superscripts

There are several alternative ways to configure subscripts and superscripts. They can all make use of the `\Send{BACK}{content}` instructions, when the bases of the subscripts and superscripts also need to be configured.

```
\Configure{SUB}{before}{after}
\Configure{SUP}{before}{after}
\Configure{SUBSUP}{before}{between}{after}
\Configure{SUPSUB}{before}{between}{after}
\Configure{SUB/SUP}{before-1}{between-1}{after-1}
                   {before-2}{between-2}{after-2}
```

The first pair of commands are for cases where the subscripts and superscripts appear in isolation.

The third command applies to adjacent subscripts and superscripts, but only if subscripts are always to appear before the superscript in the output, regardless of their order in the source code. The fourth command is a variant of the third command, where subscripts always are to appear after the superscripts. The fifth command is another variant, requesting the output to preserve the order of the subscripts and superscripts in the source.

The arguments *before-1*, *between-1*, and *after-1* apply when the subscript precedes the superscript and the arguments *before-2*, *between-2*, and *after-2*, when the superscript comes first.

```
\Configure{()}{\DviMath$}{$\EndDviMath}
\Configure{SUBSUP}
   {\Send{BACK}{[base]}[/base][sb]}
   {[/sb][sp]}
   {[/sp]}
\(a^b_c\)
```

[base]a[/base][sb]c[/sb][sp]b[/sp]

3.6. Accents

TEX4ht deals with accents by looking up a table constructed by use of the following command:

```
\Configure{accent}\textcmd\mathcmd
{{in-1}{out-1}{in-2}{out-2}...{}{out-last}}
{do-found} {do-not-found}
```

The pairs `{in-1}{out-1}`, `{in-2}{out-2}`,... set up series data records for the accent. The first field of the last pair of data fields must be empty.

The table is assigned to a text-based accent command `\textcmd` and a corresponding math-based accent command `\mathcmd`.

When an accent command is encountered in the source document, its table is searched for a record where the first field matches the argument of the command. If a match is found, the parameter *do-found* is activated. In this case, the second field of the pair is available as the macro parameter `#1`.

If no match is found, the *do-not-found* is activated. In this case, the parameter `#1` represents the accent command under its original meaning in LATEX, and the parameter `#2` represents the argument of the command.

```
\Configure{accent}\^\hat{a{a}{}{}}
    {[hat]#1[/hat]}
    {[HAT]#2[/HAT]}
\Configure{accent}\vec\vec{{}{}}
    {} {[vec]#2[/vec]}
$\vec a=(\hat a,\hat b)$
```

[vec]a[/vec] = ([hat]a[/hat] , [HAT]b[/HAT])

If the second argument is empty, the previous table for the accent commands is used. Similarly, if both *do-found* and *do-not-found* are empty, the previous fragments of code are used.

◇ Michel Goossens

◇ Sebastian Rahtz

Powyższy tekst jest fragmentem nowej książki *The LATEX Web Companion: integrating TEX, HTML, and XML*. W książce szczegółowo omawiane są problemy związane z publikowaniem plików LATEX-owych w sieci WWW, a w szczególności omawiane są: wykorzystanie technologii Acrobat, zamiana dokumentów LATEX-owych na HTML/XML. Przedstawiane są m.in. następujące narzędzia: `latex2html`, `tex4ht`, narzędzia do tworzenia i interpretowania formatu XML oraz do zamiany dokumentów w formacie XML z wykorzystaniem standardów DSSSL/XSL (`jade`, `xt`). W książce można znaleźć wiele informacji nt. standardu XML.

Dziekujemy wydawcy i autorom za zgodę na publikację i udostępnienie tekstu [Red.].

Michel Goossens Sebastian Rahtz *The LATEX Web Companion: integrating TEX, HTML, and XML* (Addison-Wesley series on tools and techniques for

computer typesetting), Addison-Wesley Longman 1999,
ISBN 0-201-43311-7.