A bit of HTML and a bit of ConTeXt

HTML constructs translated to ConTEXt

Abstract
Described is a module for the typesetting of a subset of HTML operators. These can be used to build data sets in XML with HTML as formatting elements and have them typeset in ConTEXt. Other features are the inclusion of predefined content and provision for language localized words and expressions.

Keywords
ConTeXt, HTML, XML, include, vocabulary

Introduction

Let us start with a tiny example, just to get the idea. For the sake of exposition the content of this example is enclosed as if it were HTML source. That is not necessary though, any root node can contain these elements. For it to work one has to load the module with `\usemodule[hvdm-xml]` first. It in turn depends on modules `[hvdm-ctx]` (a few general macros) and `[hvdm-lua]` (some Lua coded functions). By the way, note the mandatory XML-notation for the characters < and > and of course the same will apply to the ampersand `&` as these are XML-related (call it) features. Also note that the example code below can be rendered in a browser unchanged.

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
<head>
  <body bgcolor="bbccbb"/>
  <title>Ignored head</title>
</head>
<body>
  Example: &lt;body&gt;
  <hr/> <!-- comment: rule -->
  <rm>roman</rm>
  <i>italic</i>
  <b>bold</b>
</body>
</html>
```

Example: <body>

---

roman *italic* **bold**

Below follows a description of the API of this module, here and there illustrated with an example. Many of the nodes are standard to HTML but there are some extras, primarily because the author needed them for his work. The idea behind the development of the module was to stay close to HTML and use its vocabulary where possible.

Document Structure

These are nodes for structuring your XML document.

- ☐ `<html>`
  The well known enclosure of all HTML.
- ☐ `<head>`
  The header part of the document. Its content is ignored because it is not supposed to play a role in the typesetting process.
- ☐ `<body>`
  The body part of the document. The content of this node can contain all the typeset material.
- ☐ `<h1 align="" color="" style="">...<h1>`
  Typeset headers in diminishing size according to the digit behind the 'h'. The alignment can be `left`, `center`, `middle` or `right`, the color of the heading as well as its style can be specified. The possibilities in size are `<h1>` ...`<h6>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <h2 align="center" color="blue"
      style="bold">
    heading text
  </h2>
</root>
```

heading text

Following are nodes for paragraphs and linebreaks.

□
Break the line.
□ <p height="" color="" align="" style="">
Encloses a paragraph, although <p/> does equally well as a simple paragraph break. Extra white can be placed above the paragraph with the height attribute. The content can be colored , aligned and given a specific style, all with attributes.
□
Just a division as in HTML.

TEX directly

These are nodes for those parts of the document where one has to resort to using TEX directly. One such area is the typesetting of math, allthough it is possible to use MathML here. However, I find that a tedious business and for quick results am using TEX's math directly. MathML translation can come later, when things have to be tidied up.

□ <tex>
Contains pure TEX or ConTEXt.
□ <m>
Encloses the node content in pair of $'s.
□ <M>
Encloses the node content in pair of $$'s.
□ <nohyphens>
Stop hyphenation and put the content inside a \begingroup-\endgroup pair, as might be expected.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  Formula inline: <m>y=a^2+bx+c</m>
<div/>
  Formula displayed: <M>y=a^2+bx+c</M>
</root>
```

Formula inline: $y = a^2 + bx + c$
Formula displayed:
$$y = a^2 + bx + c$$

Appearance

These are nodes for making font, style and color changes.

□ <font family="" size="" style="">
Does font setup \setupbodyfont[family], \switchtobodyfont[size] and/or a simple style change, where appropriate. The presence of a family attribute triggers a \setupbodyfont command, a size is effected with a \switchtobodyfont. The attribute for the size can be big, small, etc. A style can be something like normal which is translated into \tf; similarly for other variants, always under the assumption that the chosen variant is available.

The list of style changes is:
– normal = \tf
– mono = \tt
– bold = \bf
– italic = \it
– bolditalic, italicbold = \bi
– slant, slanted = \sl
– boldslanted, slantedbold = \bs
– smallcaps = \sc
– mediaeval = \os

□ <small step="" min=""> or <smaller>
Typeset content smaller by the step size (usually in points) but enforce a minimum size given in the min attribute.
□ <big step=""> or <bigger>
Typeset content bigger by the step size (usually in points).
□ <color value="">
Sets content in the color given in the attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
Before
<font family="euler" size="big">
  Fontswitch
</font>
 After
</root>
```

Before – Fontswitch – After

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<smaller step="2pt">
  smaller
</smaller>
normal
<big step="2pt">
  bigger
</big>
</root>
```

smaller normal bigger

The common HTML style nodes are present.

☐ <r>
Typeset content in \tf.
☐ <u space="yes/no">
Typeset content underlined, the space attributed governs the breaking of the underlining at word boundaries.
☐ <i>
Typeset content in \it.
☐ <b>
Typeset content in \bf.
☐
Typeset content emphasized as with \em.
☐ <sl>
Typeset content in \sl.
☐ <tt>
Typeset content in \tt.
☐ <code>
Typeset as in the corresponding HTML.
☐ <pre>
Typeset as in the corresponding HTML.

Special operations on text.

☐ <sub>
Typeset content in subscript.
☐ <sup>
Typeset content in superscript.
☐ <lohi>
Typeset content both in subscript and in superscript. The node should contain a <sub> and a <sup> node, in any order.
☐ <quote> and <q>
These enclose their content in single and double quotes, respectively.

Space and Alignment
These are nodes for structuring the typesetting with horizontal and vertical space.

☐ <spacer size="" type="">
By default the kind of spacing depends on whether the typesetting is currently in horizontal or in vertical mode. But its kind can be explicitely specified by setting the attribute type to either horizontal or vertical. A third type fill is evaluated to \hfill or to \vfill respectively. By default the size values are 1em and \normalbaselineskip for the horizontal and vertical displacements. The size may also be given as a percentage, which is applied to \textwidth or \textheight.
☐
Typeset centered.
☐ <left>
Typeset aligned to the left.
☐ <right>
Typeset aligned to the right.
☐ <narrow left="" right="" middle="" option="">
Narrowed paragraph, the option attribute determines the narrowing as in the command \startnarrower, middle is default.
The other attributes determine the value of the narrowing and evaluate to calling command \setupnarrower.
☐ <blockquote>
The HTML blockquote behaviour.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<narrow middle="1cm">
  abc
  <spacer type="fill"/>
  abc
  <spacer type="fill"/>
  abc
  <p/>
  <spacer type="vertical" size="1cm"/>
  abc
</narrow>
</root>
```

abc abc abc

abc

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<center>
  Centered
</center>
</root>
```

Centered

Lists
These are nodes for enumerations, mimicking those in HTML.

☐
List element.
☐ <ol sym="" start="" columns="" option="">
Ordered list with attribute sym specifying the kind of numbering: n, a, A, r, R, g, G for numbers,

letters, roman numerals or Greek letters.
The `start` attribute determines the number of the first item, by default 1 as expected. The `columns` attribute specifies the number of columns (1 by default).
Attribute `option` functions as sort of catch all, because its content is transferred to the first `[]`-argument of the typesetting `\startitemize` macro.

☐ `<ul sym="" columns="" option="">`
Unordered list with the symbol having a great number of variants, switched by a number from 1 to 14 and ranging from bullets to triangles and lozenges (provided the font has them).

☐ `<dl compact="yes/no">`
As in HTML, encloses <dt> and <dd> elements.

☐ `<dt>`
As in HTML.

☐ `<dd>`
As in HTML.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<ul sym="3" columns="2">
  <li>first item</li>
  <li>second item</li>
  <li>third item</li>
  <li>fourth item</li>
</ul>
</root>
```

first item	third item
second item	fourth item

Tables

These are nodes for tables, mimicking those in HTML and translated to XML-table macros of ConTEXt.

☐ `<table foregroundstyle="" offset="" columndistance="" spaceinbetween="" location="" option="">`
The setup attributes implemented here are `foregroundstyle` corresponding to HTML's style, `offset` corresponding to cellpadding, `columndistance` to cellspacing, `spaceinbetween` to rowspacing and `location` for placement `top`, `left` or `middle`. The catch all `option` attribute can be used to set `frameoffset`, `backgroundoffset`, `textwidth`, `textheight`, `leftmargindistance`, `rightmargindistance`.

☐ `<thead>`
As in HTML.

☐ `<tfoot>`
As in HTML.

☐ `<tbody>`
As in HTML.

☐ `<tr>`
As in HTML.

☐ `<td>`
As in HTML.

☐ `<th>`
As in HTML.

Images and graphics

These are nodes for typesetting images and pictures.

☐ `<framed many-attributes>`
Frame the contents of the node. There are a great many of attributes here, corresponding with the most important ones from `\setupframed`. To name them without further explanation: `height`, `width`, `offset`, `color`, `bgcolor`, `frame`, `framecolor`, `framecorner`, `frameradius`, `rulethickness`, `strut`, `align`, `valign`, `corner`, `radius`, `bgcorner`, `bgradius`, `option`. The last one `option` can be anything and its value is passed on to the underlying setup call.
Frame parts may be specified with HTML-like attributes for the `frame` attribute and has values

- `on,t,above,vsides` is `topframe=on`
- `on,b,below,vsides` is `bottomframe=on`
- `on,l,lhs,hsides` is `leftframe=on`
- `on,r,rhs,hsides` is `rightframe=on`

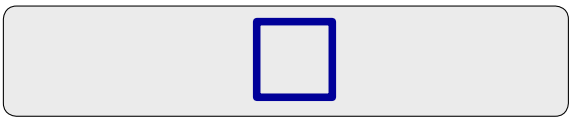
☐ `<img>`
Place an image file through `\externalfigure`. Separate attributes are `height`, `width`, `scale`, `frame`, `corner`, `radius`, `rotation`, `option`. They are passed on in the second `[]`-argument to `\externalfigure`.

☐ `<mpgraphic name="" [file="" buffer=""] parameters="">`
Execute METAPOST on an MPgraphic definition and place this in the document. The node must contain a `\startuseMPgraphic` `\stopuseMPgraphic` pair.
Other sources for the definition of the graphic may be specified with the `file` and `buffer` attribute.
The attribute `parameters` is used to transfer variable values via the `\MPvar` macro to the METAPOST code. Note that the name on the `\startuseMPgraphic` must be the same as the one on the <mpgraphic>.

Example where the graphic is defined in a separate ConTEXt buffer.

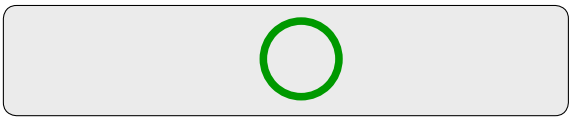
```
\startbuffer[graphic]
\startuseMPgraphic{graphic1}{color}
  pickup pencircle scaled 1mm;
  draw unitsquare scaled 1cm
      withcolor \MPvar{color};
\stopuseMPgraphic
\stopbuffer

<?xml version="1.0" encoding="UTF-8"?>
<root>
<center>
  <mpgraphic buffer="graphic" name="graphic1"
        parameters="color=darkblue"/>
</center>
</root>
```



Example where the graphic is defined in the <mpgraphic> node itself.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<center>
  <mpgraphic name="graphic2"
        parameters="color=darkgreen">
    \startuseMPgraphic{graphic2}{color}
      pickup pencircle scaled 1mm;
      draw unitcircle scaled 1cm
          withcolor \MPvar{color};
    \stopuseMPgraphic
  </mpgraphic>
</center>
</root>
```



Include and definition nodes

Instead of repeatedly copying the same code over and over, one can define these common parts separately (`includes.xml` in the example below) in a file. This file must contain a root node in which the definitions of the common code parts are put. The particular name of this node is not significant, it is merely a container for the <define> nodes inside. In the example below <includes> was chosen.

Each of the definitions contained in the file should be given a name whereby it can be called up for insertion. Loading the file with definitions is done with the <include> node carrying the path of the file to read.

```
<?xml version="1.0" encoding="UTF-8"?>
<includes>
  <define name="mypart">
    .. code to include ..
  </define>
  .. other definitions ..
</includes>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <include file="includes.xml"/>
  .. other nodes ..
</root>
```

A <define> node carrying content is a *definition* node. If more than one <include> file is loaded, then later inclusions have priority above the earlier ones. The first match of a definition breaks off the search.

A <define name="name"/> node not carrying content implies *retrieval* of the definition. Thus the presence or absence of the node's content determines its role. In the following example code the definition `mypart` is substituted for <define name="mypart"/>.

```
<includes>
  <define name="mypart">
    .. code of mypart ..
  </define>
</includes>
<text><define name="mypart"/></text>
```

The <define> can have a `type` attribute if it is a definition node. In that case some special action depending on its value is taken when called up. The preprogrammed actions are:

1. `type="image"`
   When the <define>'s content resolves to the location of a file, this is put into the document as an `\externalfigure`.
2. `type="mpgraphic"`
   The <define> must resolve to code for a graphic.

For example a definition can be

```
<define type="mpgraphic" name="example"
    parameters="color=green,variant=0">
\startuseMPgraphic{example}{color,variant}
  if \MPvar{variant} = 0:
    draw (0,0) -- (10,10)
```

```
        withcolor \MPvar{color};
  else:
    draw (0,0) .. (10,10);
  fi
\stopuseMPgraphic
</define>
```

We then call this definition with

```
<define name="example"
    parameters="color=orange"/>
```

On the call the defined default color green is replaced by orange, while the variant keeps its default of 0. Note that the names on the <define> and the graphic definition must be the same.

Other attributes provided for are height and width or scale for its dimensions, the former having priority above the latter. The dimensions might be given of a percentage, which is taken as a fraction of \textheight and \textwidth respectively. The rotation attribute specifies a rotation of the figure given in degrees.

Beware. The inclusion of a <define> is not the same as macro substitution in a programming language as for instance C. Complicated formulas in MathML are good candidates for a definition.

Vocabulary

It is not uncommon to program in English for all terms that can appear in the typeset document. However, it would be nice if instead these could be customized and typeset in the target language selected with the ConTEXt macro \language[]. The <vocabulary> node can help to enlighten this task. With it one defines for a chosen series of words the equivalent in each of several languages. The next example defines the English word *file*, the Dutch *bestand* and the German *Datei* as equivalents. Note the use of the two letter language designators.

```
<vocabulary>
  <word>
    <language name="en">file</language>
    <language name="nl">bestand</language>
    <language name="de">Datei</language>
  </word>
</vocabulary>
```

One of the languages will function as the *reference language* in which the program specifies the terms (English by default). The macro calls \translate{file} and \Translate{file} are typeset as *bestand* and *Bestand* for either the 'nl' or the 'de' current language. The same is accomplished with the nodes <word> and <Word>.

Further customization is the possibility to specify another language as the reference language than the default English. Give the reference language as its two letter value on the vocabulary attribute:
<vocabulary referencelanguage="">

The vocabulary can be placed in a file or in a buffer whose name must be given as attribute:
<vocabulary file="" buffer="">

Convenience Nodes

Not HTML but probably useful now and then. A save-restore, current time and date macros are provided for.

- <currendate/>
  Typesets the current date in European style.
- <currenttime/>
  Typesets the current time in a 24-hour clock.
- <store name="">
  Store content of the node under name.
- <restore name="">
  Call up content stored under name.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
store then
<store name="test">
  stored text
</store>
restore:
<restore name="test"/>
<div/>
current date: <currentdate/>
<div/>
current time: <currenttime/>
</root>
```

```
store then restore: stored text
current date: 12-07-2013
current time: 11:27
```

Hans van der Meer
H.vanderMeer@uva.nl