Hans Hagen
PRAGMA ADE
Ridderstraat 27
8061GH Hasselt NL
pragma@wxs.nl

sgml

# The NTG MAPS bibliography
## from SGML to TEX to PDF

**abstract**
A few years ago, the ntg decided to put their Maps volumes on the internet in the pdf file format. At about the same time, it was decided to build the associated bibliography in such a way that it could be used to produce both a html and pdf document.
Recently, the Maps bibliography has been converted to a proper xml document source. In the process, the descriptions were made as consistent as possible. The xml source was used as input for a pdf document, that provides extensive browse and search options. This pdf file, along with the Maps articles, is provided to ntg members as an additional service.
In this article the electronic ntg Maps will be presented and the specific characteristics of the production process will be explained. Also, some of the complicating aspects will be discussed. I assume that the reader is familiar with sgml and TEX. The focus will be on the interface between sgml, TEX and pdf.

**keywords**
cdrom, ConTEXt, ntg, Maps, pdf, sgml, xml

### Introduction

The Dutch speaking TEX Users Group NTG was founded over ten years ago. Right from the start, the minutes and appendices (MAPS) of the meetings have played an important role not only in registering what was taking place, but also in providing useful information to the members on how to use TEX and where to find macros and programs.

During those ten years the MAPS, which is sent to the members two times a year, has grown to about 200 pages per volume. At the tenth anniversary it was decided to redesign the layout and change the name into a meaningless succession of four characters. From that moment on, the articles became the MAPS and the minutes got their own A5 booklet. One reason for splitting up the content was that pure membership issues were not that relevant for non members who could fetch the latest volumes from CDROM's and the NTG internet site.

When the MAPS was made available on the NTG internet site, a bibliography was set up by Erik Frambach. This first version was coded in the BIBTEX format and converted to HTML afterwards. Erik also provided a version coded in TEX, that was used to construct a typeset PDF version.

When maintenance of the bibliography was transferred to the MAPS editors, Taco Hoekwater converted the database into XML. It was this database that provided the input for the PDF document I will describe here in more detail. This document does not differ that much from the prototype version. In the process of fine–tuning the document, I also normalized the index entries and author names.

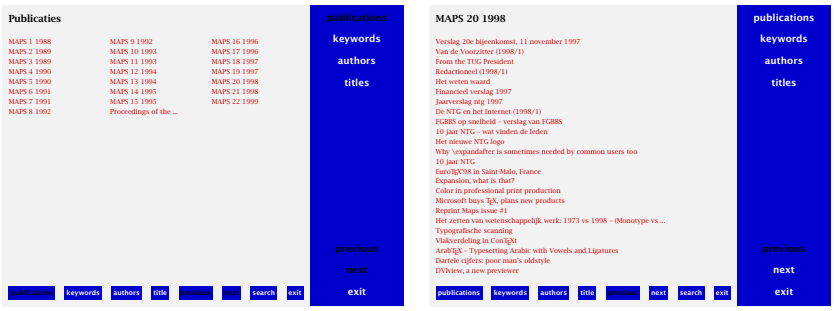
### The structure

When we open the bibliography, the title page (as shown in figure 1) shows up. The abstract is typeset in black, but other components come in red, white, and blue: the colors of the dutch national flag. The belgian flag is in black, yellow, and red, and the yellow shows up when we click on a button.


title page article description

**Figure 1**

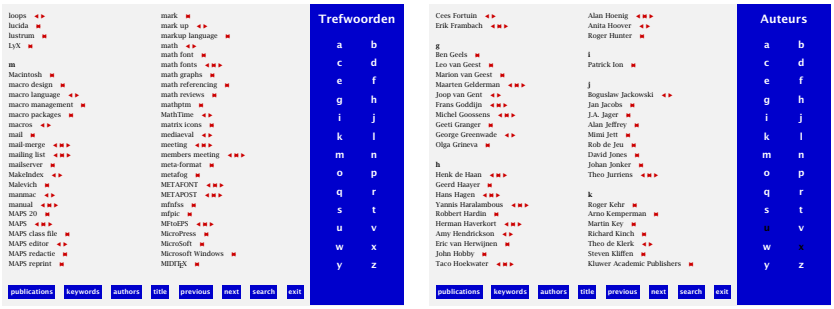
The right side and bottom areas of the screen are used for navigational ornaments. In a document like this one will, in most cases, use indices to locate a topic. Nevertheless, we provide some tables of contents: a main one and one per volume (see figure 2).




main table of contents volume table of contents

**Figure 2**

There are three indices, of which the keyword and author index are linked registers (figure 3). A linked index permits the user to 'walk though the document', i.e., showing each page to which the index entry refers.




keyword index author index

**Figure 3**

The title index (figure 4) collects the articles per author. Finally, there is a colofon page that tells some more about the NTG. Like the MAPS, the bibliography is a mix of dutch and english.



title index colofon

**Figure 4**

## The Database

The database is coded in XML, a restricted form of SGML that was introduced recently as something new and better, and seems to be accepted more easily. The 22 volumes that make up the document described here have 710 abstracts. The resulting document has 786 pages and 18,421 hyperlinks. The PDFTEX version 14a binary packs this into a 4.5 Megabyte PDF document. Although not that complicated and big, this document is a nice example of what TEX can do with such a database.

The abstract shown in figure 1 is coded in XML as follows:

```
<bibentry type="article" id="20-44" language="en">
  <title>
    The Calculator Demo -- Integrating &tex;, MetaPost, JavaScript and PDF
  </title>
  <author>
    <au index="hagenh"><fn>Hans</fn><sn>Hagen</sn></au>
  </author>
  <published volume="20" year="1998" pages="290-296" size='201'>
    <journal>MAPS</journal>
  </published>
  <keywords>
    <key>METAPOST</key> <key>JavaScript</key> <key>PDF</key>
    <key>&context;</key> <key>pdf&tex;</key>
  </keywords>
  <abstract>
    Due to it's open character, &tex; can act as an authoring tool. This
    article demonstrates that by integrating &tex;, MetaPost, JavaScript
    and PDF, one can build pretty advanced documents. More and more
    documents will get the characteristics of programs, and &tex; will be
    our main tool for producing them. The example described here can be
    produced with pdftex as well as traditional &tex;.
  </abstract>
</bibentry>
```

Some logo's are coded as special tokens using the `&token;` syntax. Certain entries come with key–value pairs. Although work is in progress by Taco Hoekwater to let a special version of TEX directly process such input, for the moment, we convert it into something more natural to TEX.

```
\endSGML[bibentry]
\begSGML[bibentry][type=article,id=20-44,language=en]
\begSGML[title]The Calculator Demo -- Integrating \tokSGML{tex}, MetaPost,
JavaScript and PDF\endSGML[title]
\begSGML[author]
\begSGML[au][index=hagenh]
\begSGML[fn]Hans\endSGML[fn]
```

```
\begSGML[sn]Hagen\endSGML[sn]
\endSGML[au]
\endSGML[author]
\begSGML[published][volume=20,year=1998,pages=290-296,size=201]
....
```

The conversion is straightforward and takes only a few lines of PERL. All manipulations will be done in CONTEXT. Before we will uncover some of the details of this manipulation, I explain the way the layout and basic struture is defined.

## Layout and basic structure

A screen has an aspect ratio quite different from the average paper dimensions, like A4 and letter. Here we use one of the predefined paper sizes S6, that has a width of 600pt and an aspect ratio of 4:3. We use the same size for typesetting and printing, which means that the page is automatically cropped to the paper size we print on.

```
\setuppapersize [S6] [S6]
```

We don't use headers, footers, and top areas, but only the text and bottom ones. The bottom as well as right edge are used for navigational tools.

```
\setuplayout
  [topspace=10pt, backspace=10pt,
   width=440pt, height=fit, header=0pt, footer=0pt,
   rightedge=120pt, rightedgedistance=20pt, margin=0pt,
   bottomdistance=10pt, bottom=20pt]
```

The bibliography is typeset in 10 point Lucida Bright, a font that renders very well on screens. We preload symbolset `navigation 1` that is a subset of CONTEXT navigation symbol font. This font is part of the standard CONTEXT distribution.

```
\setupbodyfont  [lbr,10pt]
\usesymbols     [nav]
\setupsymbolset [navigation 1]
```

We turn on colors. To save some processing time we use local color mode, which means that bookkeeping is minimized to page bound coloring.

```
\setupcolors [state=local]
```

The page has a light gray (.95) screen as background. The rightedge text and bottom areas become blue. The offset ensures that the text will not touch the border.

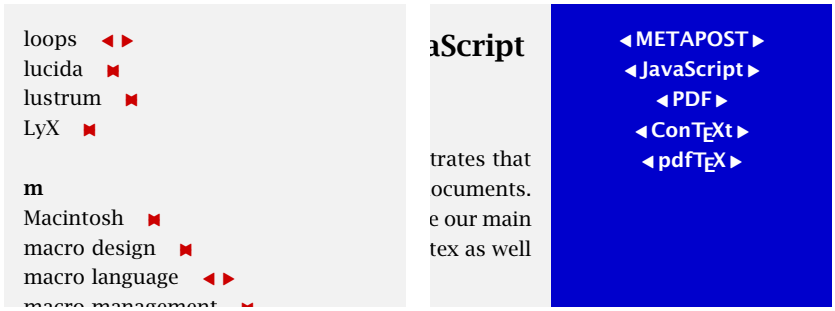
```
\setupbackgrounds
  [page]
  [background=screen]

\setupbackgrounds
  [text,bottom] [rightedge]
  [background=color, backgroundcolor=darkblue, backgroundoffset=10pt]
```

Because we are dealing with an interactive document, we have to set up the interaction. In a document like this we can save quite some bytes when we use page destinations in hyperlinks instead of named ones. By saying `page=yes` we tell CONTEXT to use page destinations when possible. Because we will use a few menus, we enable this feature. Especially when we use color, a (bold) sans serif font makes hyperlinks more visible. Hyperlinks will be colored red, unless they point to the current page, in which case they will be typeset in black.

```
\setupinteraction
  [state=start,
   page=yes,
   menu=on,
   style=\ssbf,
```

```
  symbolset=navigation 1,
  color=darkred,
  contrastcolor=black]
```




linked keyword keyword menu

**Figure 5**

The indices are, as already mentioned, cross linked ones. This is a standard ConTeXt feature that was first applied in 1996 in a huge 4000 page document with many indices and lists (different tables of contents). Such linked indices are a useful alternative for endless lists of page numbers in an index. The registers are defined by:

```
\defineregister [keyword]        [keywords]
\defineregister [author]         [authors]
\defineregister [titlebyauthor] [titlesbyauthor]

\setupregister  [keyword,author] [coupling=yes]
```

We must set up the cross linked registers explictly. The next commands load and preprocess the relevant data.

```
\coupleregister [keyword]
\coupleregister [author]
```

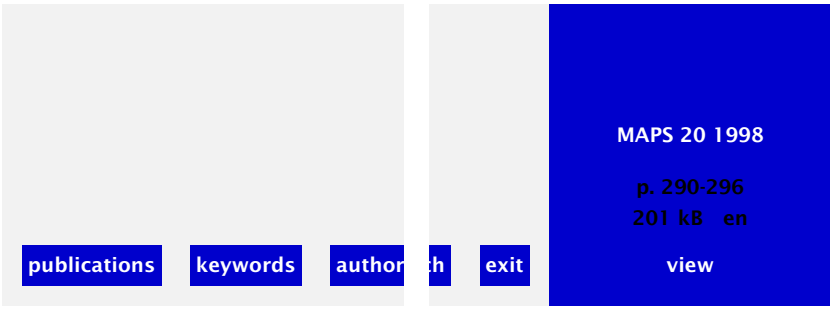
The titles by author index is not cross linked. Here we want to click on the whole entry, and we don't want to see a page number or symbol (figure 8). There is no alphabetic section indicator (a, b, c, etc.). We will expand index entries (more about that later) and limit their typeset size.

```
\setupregister
  [titlebyauthor]
  [n=1, symbol=none, interaction=text, indicator=no]

\setupregister
  [keyword,author,titlebyauthor]
  [maxwidth=.8\hsize, expansion=yes]
```

The keywords themselves are accompanied by left and right triangles that bring us to the previous or next location. Clicking on the keyword itself brings us back to the index. In the index there can be one, two, or three symbols that represent the begin, middle, and/or end of the list.

In ConTeXt one can define menus that can be turned on and off. Normally the menus are located in the top, bottom, left, and/or right edge areas. There can be multiple stacked or overlapping menus. Here we use one bottom menu and three right menus. When available, we will put some additional information on the article in the bottom of the menu (figure 6).




bottom menu status area

**Figure 6**

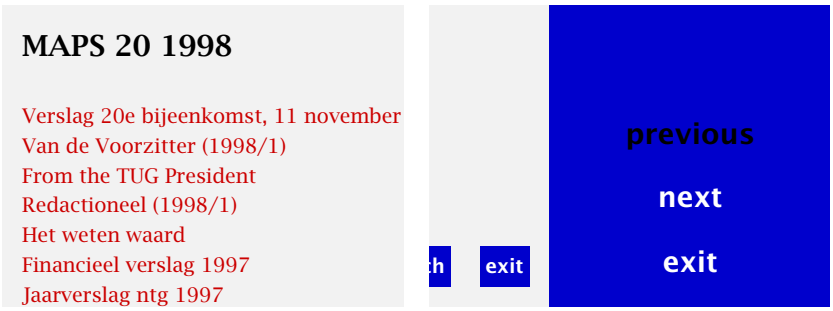
We use no fancy buttons, only colored backgrounds, so we can stick to:

```
\setupinteractionmenu
  [bottom]
  [state=start,
   height=\bottomheight,
   frame=off,
   background=color,
   backgroundcolor=darkblue,
   color=white,
   style=\ssbf]
```

These settings will be applied to:

```
\startinteractionmenu[bottom]
  \but [publications]     publications \\
  \but [keywords]         keywords     \\
  \but [authors]          authors      \\
  \but [titles]           titles       \\
  \but [previoussubpage] previous     \\
  \but [nextsubpage]      next         \\
  \but [SearchDocument]   search       \\
  \but [CloseDocument]    exit         \\
\stopinteractionmenu
```

ConTeXt has a so called integrated cross reference mechanism. Instead of providing dozens of commands that deal with interactive issues, we use only a few that interpret their arguments. In this example, the last two references are special in the sense that they refer to viewer actions (figure 7).




simple lists viewer control

**Figure 7**

At the right of the screen there are a few alternative menus. These are laid over each other and turned on and off when needed. The index menu (the one with the alphabet)

contains a vertical box with a..z buttons, each taking 40% of the width, and typeset as a centered paragraph. (figure 8). The main menu is set up as:

```
\setupinteractionmenu
  [right]
  [state=start,
   frame=off,
   color=white,
   style=\ssbfb,
   distance=overlay]
```

By setting `distance` to `overlay`, the right menu will be typeset on top of other menus we place at the right (otherwise menus would be stacked). The definition below shows that occasionally one cannot do without building boxes and using fills. Even when one uses a high level of abstraction as provided by CONTEXT, some insight in `\xbox`, `\xfill` and `\xskip` (with `x` being `v` or `h`) is useful.

```
\startinteractionmenu[right]
  \boxofsize \vbox \textheight \bottomdistance \bottomheight
    \bgroup
      \setupalign[middle]
      \but [publications]    publications \\
      \but [keywords]        keywords     \\
      \but [authors]         authors      \\
      \but [titles]          titles       \\
      \vfill
      \but [previoussubpage] previous     \\
      \but [nextsubpage]     next         \\
      \but [CloseDocument]   exit         \\
      \unskip
    \egroup
\stopinteractionmenu
```

The `\unskip` is needed because by default some white space is added between buttons and by using an embedded `\vbox`. CONTEXT cannot automatically correct for this at the bottom of the menu.

The second type of structural element (apart from the registers) is the table of contents. Like registers, there can be many of them at each level of sectioning. For readability, we don't use chapters and sections but:

```
\definehead [structuralelement] [chapter]
\definehead [publication]       [chapter]
\definehead [publicationtitle]  [section]
```

They will show up as defined below. We just set up their parents from which they inherit. Numbers don't make sense here.

```
\setuphead
  [chapter,section]
  [style=\bfb,
   before=,
   after={\blank[3*medium]},
   align={right,broad},
   number=no]
```

In CONTEXT there currently are three levels of page numbering: the real page number which runs from 1 to the number of pages in the document, the typeset number that in most cases does not increment on unnumbered pages, and the subpage number, that is used in navigational tools. Some day this scheme will be extended to a more advanced one with more (independent) subpage numbers. Here we use the subpage numbers on a per chapter base:

```
\setupsubpagenumber [state=start, way=bychapter]
```

When we define (or actually clone) a head, we also get a dedicated list similar to a table of contents. In this document, we will typeset this list with the following settings. We use a simple title–only list (figure 7).

```
\setuplist
  [publication,publicationtitle]
  [alternative=f,
   interaction=all,
   maxwidth=.8\textwidth,
   before=,after=]
```

By default heads are not expanded, but when we use variables, which happens to be the case in this kind of SGML processing, we definitely need to expand this variable when we write it to a list.

```
\setuphead [publication,publicationtitle] [expansion=yes]
```

A button is something to click on. Menu buttons are typeset conforming to the menu settings, and those not native to menus show up like:

```
\setupbuttons [color=white, style=\ssbfb, frame=off]
```

The last layout item we will setup before dealing with SGML are the menus that show up along side a bibliography item and the registers.

```
\defineinteractionmenu
  [publication][right]
  [color=white,
   contrastcolor=white,
   frame=off,
   style=\ssbf,
   distance=overlay]
```

The menus alongside the registers will show a navigational alphabet. The next definition makes sure we get big buttons with a width of 40% edge width. For consistency reasons, the struts follow the current font outside the button, so here we have to make sure we use a larger button.

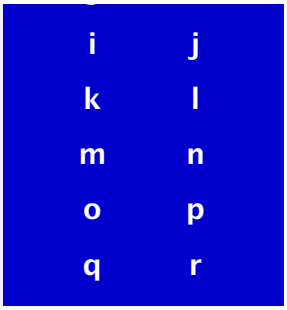
```
\defineinteractionmenu
  [navigation][right]
  [color=white,
   contrastcolor=white,
   distance=overlay,
   width=.4\rightedgewidth,
   frame=off,
   style=\ssbfb\setstrut\strut,
   unknownreference=yes,
   samepage=yes]
```

The last two parameters ensure that buttons pointing to the current page show up too. The menu itself will be defined in the final text flow (discussed later) using the next macro. The last command generates the alphabet buttons (figure 8).

```
\def\indexmenu[#1]#2#3%
  {\setupalign[middle]
   \noindent\button{#3}[#1]
   \blank
   \registermenubuttons[navigation][#2]}
```

## Mapping SGML onto TEX

In the previous layout definitions, we already put some structure into place. This section is dedicated to relating SGML to this structure. We will use some of the interfacing commands defined in the CONTEXT SGML module. This module acts upon a TEX file




Donald Arseneau
The TEX Hierarchy
Typesetting paragraphs of a specifie
Andrey Astrelin
Graphics for TEX: a new implementat
David Barron
Portable Documents: Why Use SGML?
Vladimir Batagelj
Combining TEX and PostScript

alphabetic entries no symbols

**Figure 8**

that results from the conversion by the `smgl2tex` PERL script. The TEX contains at most four different commands:

```
\begSGML[tag][var1=val1,var2=val2,...,varN=valN]
\endSGML[tag]
\sepSGML[tag]
\tokSGML{identifier}
```

These commands are direct translation of:

```
<tag var1=val1 var2=val2 ... varN=valN>
</tag>
<tag/>
&identifier;
```

In SGML setups are separated by spaces (and values can be surrounded by single or double quotes), in CONTEXT we use commas as separators.

The TEX commands are interpreted according to their definitions. Apart from some more obscure alternatives, the main definitions are given here.

```
\defineSGMLgroupedentity  [tag] [init] {actions} {actions}
\defineSGMLisolatedentity [tag] [init] {actions}
\defineSGMLseparateentity [tag] {actions}
\defineSGMLspecial        [identifier] {substitution}
```

Grouped and isolated entities are invoked by `\begSGML` but only the first one has `\endSGML`. The separate entity relates to the `\sepSGML` macro. The entities using a `\begSGML` are subjected to some basic testing on proper nesting. By providing an additional argument, definitions can be defined local to other ones, but we don't need that feature here.

When a document is not processed in the same order as it is coded, or when we want to manipulate some content first, we have to save the entries and process them manually.

```
\defineSGMLgetentity     [tag] [init]
\defineSGMLprocessentity [tag] [init]
```

The first macro defines an entry as to be stored. Once stored, the entry can be flushed (executed, expanded, or whatever term suits best) using

```
\processSGML [tag]
```

The second definition has execution built in. The associated process is defined by:

```
\defineSGMLprocess [tag] {actions}
```

One can add tokens to the stored data by:

```
\appendSGML  [tag] {tokens to append}
\prependSGML [tag] {tokens to append}
```

There are a few tests available:

```
\doifSGMLelse {tag} {when data} {when no data}
\doifSGML     {tag} {when data}
\doifnotSGML  {tag}             {when no data}
```

Parameters (initializations) can be fetched by two macros. The first one returns the variable, while the second one stores it into a macro (in this case \SomeVariable).

```
\varSGML[tag]
\setSGML\SomeVariable[tag]
```

There are some more module specific macros, but the ones described here give us enough machinery to handle the MAPS biographic data.

The whole bibliography is collected (embedded) into the bib entity. The <bib> and </bib> are mapped onto \begSGML[bib] and \endSGML[bib]. The associated TEX commands are:

```
\defineSGMLgroupedentity [bib] {\page\bgroup} {\page\egroup\endinput}
```

This means that when <bib> is encountered, we go to a new page, and when the </bib> is seen, we flush the page and stop reading the file. That way we make sure we skip all kind of comments at the end of the file.

Each <bibentry> has a few associated parameters that we set to nothing by default. Where the previous definition is executed as soon as it is seen, a <bibentry> is defined as an entity that should be saved first and executed by an associated macro to be defined later.

```
\defineSGMLprocessentity [bibentry] [type=,id=,language=]
```

The next definitions are processed directly. The br entity is the only one that is defined in the XML way: <br/>. It comes alone.

```
\defineSGMLseparateentity [br] {\blank}
\defineSGMLgroupedentity  [tt] {\bgroup\tt} {\egroup}
\defineSGMLgroupedentity  [em] {\bgroup\em} {\egroup}

\defineSGMLgroupedentity [itemize] {\startitemize[packed]} {\stopitemize}
\defineSGMLgroupedentity [item]    {\item}                 {}
```

In an average document, we can stick to definitions like this, but here we have to manipulate data. The rest of the definitions take care of storing the data in a structured way. In a few moments we will see the related processing macros.

```
\defineSGMLgetentity[title]
\defineSGMLgetentity[booktitle]

\defineSGMLgetentity[author]
  \defineSGMLprocessentity[au]
    \defineSGMLgetentity[fn]
    \defineSGMLgetentity[prf]
    \defineSGMLgetentity[sn]

\defineSGMLgetentity[editor]

\defineSGMLprocessentity[published][volume=,pages=]
  \defineSGMLgetentity[series]
  \defineSGMLgetentity[journal]
  \defineSGMLgetentity[publisher]

\defineSGMLgetentity[keywords]
  \defineSGMLprocessentity[key][index=]

\defineSGMLgetentity[abstract]
```

Now, what happens when the file is read? The first meaningful entity is the outer <bib> one: it starts a new page. Then comes a sequence of over 700 <bibentry>'s with embedded data. Each entry is temporarily saved, and then executed. In the process, the

individual data components are saved and processed in due course. The next definition defines what should take place after a bibliography entry is read in (stored).

```
\defineSGMLprocess[bibentry]
  {\page
   \bgroup
   \setSGML\CurrentType[type]
   \setSGML\CurrentId[id]
   \setSGML\CurrentLanguage[language]
   \processSGML[bibentry]
   \processaction
     [\CurrentType]
     [inproceedings=>\HandleBook    ,
            article=>\HandleJournal,
            message=>\HandleJournal,
            verslag=>\HandleJournal]
   \egroup}
```

The macros `\setSGML` save some parameters for later use. Next we process everything between `<bibentry>` and `</bibentry>` that was automatically saved. Due to previous definitions, this means that some components are also saved for later use (like the index entries), and some are processed immediately (like `<published>`).

```
\defineSGMLprocess[published]
  {\setSGML\CurrentVolume[volume]
   \setSGML\CurrentYear[year]
   \setSGML\CurrentPages[pages]
   \setSGML\CurrentSize[size]
   \processSGML[published]}
```

We use the type of entry to determine the next action. There are some more entries, but those are simply skipped.

```
\def\HandleJournal%
  {\CheckPublication{\processSGML[journal] \CurrentVolume\space\CurrentYear}
   \HandlePublication}

\def\HandleBook%
  {\CheckPublication{\processSGML[booktitle]}
   \HandlePublication}

\let\CurrentPublication\empty

\def\CheckPublication#1%
  {\doifnot{#1}{\CurrentPublication}
     {\xdef\CurrentPublication{#1}
      \doglobal\stripSGML\CurrentPublication\to\CurrentPublicationAscii
      \publication[\CurrentPublicationAscii]{#1}
      \placelist[publicationtitle]
      \page}}
```

These macros demonstrate that when manipulating data, one cannot do without an occasional more complicated macro. Each article in the bibliography is either a MAPS one, or one published in proceedings. Although each set of articles is preceded by a description of the current volume, using this information to trigger a new volume is not straightforward. It proved to be easier to use a change in volume description as a trigger. The main reason for this is that the bibliography is rather flat, and articles are not embedded in something like `<volume>` and `</volume>`.

The second complication is that we have to make sure that cross references passed to the sectioning commands (between square brackets) are not disturbed by long sequences of low level TEX resulting from expansion. The `\stripSGML` macro makes sure we get a rather pure string.

```
\def\HandlePublication%
  {\setupinteractionmenu[right][state=stop]
```

```
\setupinteractionmenu[publication][state=start]
\setupbottomtexts[edge]
  [][{\hfill\menubutton[bottom]{view}[file(\CurrentId)]\hfill}]
\publicationtitle{\processSGML[title]}
\doifSGMLelse{abstract}
  {\processSGML[abstract]}
  {{\sl geen samenvatting (no abstract)}}
\page}
```

Each publication is typeset on a separate page. At the right we put a dedicated menu that we still have to define. In the bottom right text we put a button that brings us to the typeset article. When no abstract is given, we print a remark. The publication title is, confirming a previous set up, expanded when written to the auxiliary file.

We still have to define the publication menu, and since we now know what we are dealing with, it makes sense to do it right away. It is, by the way, not really a menu. Embedded in a bit of typographic trickery, we place the linked keyword and author lists, a button that brings us back to the relevant table of contents, and a bit of data about the article.

```
\startinteractionmenu[publication]
  \setupinteraction[color=white]
  \startalignment[middle]
  \topskipcorrection
  \ssbf\setupinterlinespace
  \processSGML[keywords]
  \vfill
  \processSGML[author]
  \vfill
  \noindent\gotobox
    {\strut\limitatetext{\CurrentPublication}{.8\hsize}{...}}
    [\CurrentPublicationAscii]
  \blank
  \strut p.~\CurrentPages\quad\CurrentSize~kB\quad\CurrentLanguage
  \stopalignment
\stopinteractionmenu
```

This leaves us with two processes: those that handle the keyword and author entities; both are made up of smaller components.

```
\defineSGMLprocess[au]
  {\bgroup
   \setSGML\CurrentIndex[index]
   \processSGML[au]
   \doifSGML{fn}{\appendSGML[fn]{\space}}
   \doifSGML{prf}{\appendSGML[prf]{\space}}
   \doglobal\stripSGML\processSGML[title]\to\CurrentTitleAscii
   \noindent\hbox to \hsize
     {\strut
      \hss
      \titlebyauthor
        [&\CurrentIndex&\CurrentTitleAscii]
        {\processSGML[fn]\processSGML[prf]\processSGML[sn]&\processSGML[title]}%
      \coupledauthor
        [\CurrentIndex]
        {\processSGML[fn]\processSGML[prf]\processSGML[sn]}%
      \hss}
   \par
   \egroup}

\defineSGMLprocess[key]
  {\bgroup
   \setSGML\CurrentIndex[index]
   \doifelse{\CurrentIndex}{}
     {\doglobal\stripSGML\processSGML[key]\to\CurrentKeyAscii}
```

```
    {\doglobal\stripSGML\CurrentIndex\to\CurrentKeyAscii}
  \noindent\hbox to \hsize
    {\strut\hss\coupledkeyword[&\CurrentKeyAscii]{\processSGML[key]}\hss}
  \par
  \egroup}
```

The main macros here are the `\coupled...` ones. These became available when we defined the related linked registers. In ConTeXt an index entry looks like:

```
\index{alpha} \index{alpha+beta}
```

The second example defined `beta` as a subentry under `alpha`. Of course we want to see a real $\alpha$ and $\beta$, so in practice we separate the typeset entry and sort key, like in:

```
\index[alpha+beta]{\mathematics{\alpha}+\mathematics{beta}}
```

When an entry itself contains a `+`, one can alternatively use an `&`, and by (optionally) passing it as the first character, one can signal the index sort script that this character is to be considered a separator. By doing so we make sure a `+` is not misinterpreted as a separator, as it happens that the `+` is sometimes part of the text.

```
\coupledauthor
  [\CurrentIndex]
  {\processSGML[fn]\processSGML[prf]\processSGML[sn]}
```

This sequence typesets the author's name (composed from three parts) and passes the sort index as provided by the database.

## The main text

Now that everything is set up, we can safely start with the main text flow, the least interesting part of the setup. To save some space, I will skip the title page and colofon. The occasional change in layout ensures that the menu is extended to the bottom of the screen.

```
\starttext

\setuplayout[bottom=0pt]
  \startstandardmakeup
    title page
  \stopstandardmakeup
\setuplayout[bottom=20pt]

\structuralelement[publications]{Publications}
  \startcolumns[n=3]
    \placelist[publication][criterium=all]
  \stopcolumns
  \input mapsbib.tex

\structuralelement[keywords]{Keywords}
  \setupinteractionmenu[navigation][state=start]
  \setupinteractionmenu[right][state=stop]
  \startinteractionmenu[navigation]
    \indexmenu[keywords]{keyword}{Keywords}
  \stopinteractionmenu
  \placeregister[keyword]

\structuralelement[authors]{Authors}
  \startinteractionmenu[navigation]
    \indexmenu[authors]{author}{Authors}
  \stopinteractionmenu
  \placeregister[author]

\structuralelement[titles]{Titles}
  \startinteractionmenu[navigation]
    \indexmenu[titles]{titlebyauthor}{Titles}
```

```
  \stopinteractionmenu
  \placeregister[titlebyauthor]

\structuralelement[ntg]{NTG}
  \setuplayout[bottom=0pt]
  \setupinteractionmenu[navigation][state=stop]
  \setupinteractionmenu[right][state=start]

  The main ... at: \goto {www.ntg.nl} [URL(ntg)].

\stoptext
```

The URL is defined previously by: `\useURL[ntg][http://www.ntg.nl]`.

## Fine points

When typesetting a large quantity of data automatically, one must make sure the outcome looks acceptable. The fact that TEX can typeset so well, does not automatically mean that its output always looks correct. When designing a layout, most of the time goes into the fine points. Setting up a basic page frame and defining some basic structure is a matter of minutes, but finding the optimal and pleasing dimensions sometimes takes hours. In the MAPS bibliography quite some time went into checking the input and making the index consistent within the boundary conditions. This document also provided an ideal environment to play a bit with SGML processing.

To handle unforeseen overfull boxes, two methods are used. First we have made sure TEX's emergency pass takes care of bad line breaks:

```
\setuptolerance [verytolerant]
```

The other method concerns automatically limiting the length of index entries and titles to about 80% of the available width. Most of this is handled by the higher level macros, but on one place we have to do it manually, saying:

```
\limitatetext {\CurrentPublication} {.8\hsize} {...}
```

## Processing

Processing an XML file comes down to converting the file into TEX and then processing that file. In most cases, the SGML file is checked beforehand, but one can never be sure of that. Therefore, much of the low level SGML processing macros is dedicated to checking symmetric use of entities and nesting. These features were added when we were given some 'professional' SGML databases, which unfortunately, and in spite of all parsers, had lots of dangling `<this>` and `</that>`'s. For processing the well coded MAPS database, we could have done without all this overhead.

## Expansion

I have already mentioned expansion a few times. In this section I will spend some more words on this topic, if only because one needs to be well aware of this TEX feature when defining an interface to SGML.

When one writes an article, the typeset text, apart from floats, the table of contents and the index, show up at the place where they are defined. In the more complicated documents, think of educational materials, the order can be different from the order in which the document is coded. Questions and answers can be coded along side each other, but may show up in different places. Formulas may be repeated in an appendix. The MAPS bibliography is simple in the sense that information is only presented once, but moderately complex because the order differs from the order in which the information is coded. Some data, like the indices, are not coded at all but constructed from pieces of code.

Definitions, formulas, chapters, and the like are often numbered. The numbers are not coded, but generated by the TEX macro package that is used. In the bibliography section, numbers will not be typeset. What is not visible in the resulting document is that there are more entries in the XML file than those that show up in the PDF file. It proved to be more practical to distill volume specific information from the entries describing the articles, than to use the entries (one per volume) dedicated to that purpose.

What are the consequences of this for processing the document? The reordering forces us to save the data temporarily, which in itself is not that complicated. A first version used the CONTEXT buffering system. This means that an entry was saved and then reprocessed under different conditions. There was a separate pass for constructing the index entries, making up the abstract, etc. The current implementation uses the save and recall facilities as provided by the CONTEXT SGML module. This is slightly faster and proves to be more convenient.

When we are manipulating data in TEX, we undoubtly come across expansion. In TEX, tables of contents, indices, cross references, certain optimizations and more depend on processing the file in several passes. The current pass uses the information saved during a previous pass. When dealing with tables of contents and indices, there are two extremes: the data is fully expanded versus the data is not expanded at all. In CONTEXT, by default, the content is not expanded. This means that no complicated protection mechanism is needed to prevent unwanted expansion.

But, as said, when processing SGML, we save the data in order to be able to manipulate it. When we are done with these manipulations and pass the data to a section title that also has to show up in a table of contents, expansion definitely *is* needed, but only as far as is comfortable. I will illustrate this with an example. Consider that we have defined:

```
\def\SomeTitle{Some title}
\def\Some     {Some}
\def\Title    {title}
```

When these definitions are permanent, it does not matter how they are saved in an external file that is read in during the next phase, so all next alternatives work out okay.

```
\chapter{Some title}
\chapter{\SomeTitle}
\chapter{\Some\space\Title}
```

When, however, the definition of `\SomeTitle` changes halfway the document, or over 700 times in a bibliography as discussed here, we're in trouble. In that case, we have to expand `\SomeTitle` prior to writing it to the auxiliary file. In a simple case like this, full expansion is no problem, because we are dealing with pure text. In the document at hand however, we have definitions like:

```
\def\Title{The use of \begSGML[tt]verbatim\endSGML[tt] in \tokSGML{tex}}
```

When we want this to end up in the auxiliary files in this way, we can use a one level expansion, which means replacing the macro `\Title` with its content. Unfortunately, macros like `\Title` can be nested or buried inside others, so actually we will need expansion until nothing can be expanded any more. This in itself is not that complicated, but one needs to be aware of this feature and potential side effects in order to write the right interface between TEX and SGML.

Actually, this expansion trickery is the only thing that complicates SGML processing as discussed here, but once one understands the problem, one can use the appropriate CONTEXT features to sort it out. In most cases the option `expansion=yes` will do the job: expand everything as far as permitted.

## Conclusion

The document described here can be found at the ConTeXt part of our internet site www.pragma-ade.nl, and, of course, at the NTG site www.ntg.nl. The complete MAPS archive will be put on CDROM in PDF format along with the bibliographies in HTML and PDF format.

In this article I have tried to give an impression on how such a document is defined. As with many interactive documents that involve some manipulation of the content, there is an easy to follow definition part as well as a more complicated one dealing with fine tuning and manipulations. One can do wonderful things with TeX, but the more one wants, the more one ends up in dealing with typographic issues and low level programming. ConTeXt provides the user with enough structural mechanics to ease at least part of the task. Defining the layout, registers and lists is not that hard, but the SGML part doesn't always deserve a beauty prize. Even if not all of what is presented here is clear, at least it gives an impression of what kind of trickery is needed. I leave it up to the reader to imagine what nasty tricks would be needed if the MAPS articles themselves were to be coded in SGML.

One of the best things about TeX is that often one has to define the layout and manipulation macros only once. This is what distinguishes TeX from other systems, but at the same time it forces the user to develop some skills in typographic programming. Fortunately, the results are rewarding.