Tralics, a LaTeX to XML translator Partie II

José Grimm

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

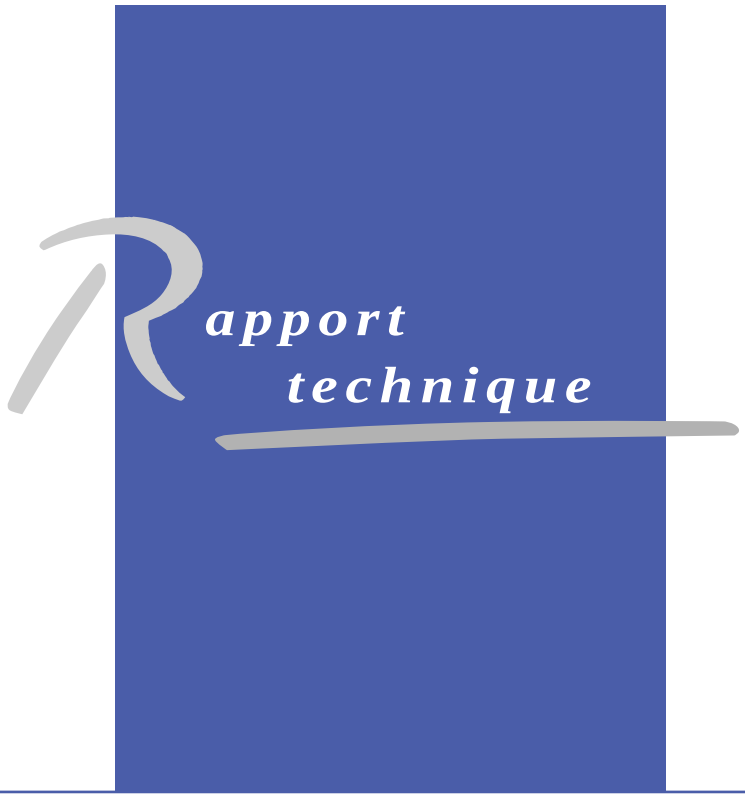
Tralics, a LaTeX to XML translator Part II

José Grimm

N° 310 — version 2

initial version September 2005 — revised version January 2007

Thème NUM



ISSN 0249-0803 ISRN INRIA/RT--310--FR+ENG

INRIA
SOPHIA ANTIPOLIS

Tralics, a LaTeX to XML translator
Part II

José Grimm*

Thème NUM — Systèmes numériques
Projet Apics

Rapport technique n° 310 — version 2 — initial version September 2005 — revised version January 2007 — 390 pages

Abstract: In this document we describe Tralics, a LaTeX to XML translator, and its application to the Raweb. There are two parts: the first part describes the translator, the second part the tools required for the Raweb.

This document has different chapters; we shall describe first how TeX can read an XML file and convert it to Pdf; in effect, we shall describe the xmltex, fotex and mathml packages, written by D. Carlisle et S. Rahtz, with some minor bug corrections and additions. We show how style sheets can be used to convert the XML source into XSL/FO or HTML, or even XML. Finally, we shall explain the Raweb DTD.

The second version of this report contains an additional chaptet that explains how to convert a Research Report or a PhD thesis in HTML using Tralics and an XML processor.

Key-words: Latex, XML, HTML, MathML, XSLT, PostScript, Pdf, stylesheet, formatting

* Email: Jose.Grimm@sophia.inria.fr

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Tralics, un traducteur de LaTeX vers XML
Partie II

Résumé : Dans cet rapport nous décrivons le logiciel Tralics, un traducteur de LaTeX vers XML, et son application au Raweb. La première partie de ce document décrit le traducteur lui-même, et la deuxième partie explique tous les outils nécessaires pour exploiter les fichiers XML.

Ce document contient plusieurs chapitres: on expliquera d'abord comment TeX peut interpréter du XML et produire du Pdf; il s'agit des packages xmltex, fotex, mathml2, écrits par D. Carlisle et S. Rahtz, avec quelques corrections et ajouts. On expliquera comment des feuilles de style permettent de convertir le XML en XSL/FO ou HTML, ou même en XML. Finalement, on expliquera la DTD Raweb.

La version 2 de ce rapport contient un chapitre supplémentaire qui explique comment convertit un document de type rapport de recherche ou thèse en HTML grâce à Tralics et un processeur XML.

Mots-clés : Latex, XML, HTML, MathML, XSLT, PostScript, Pdf, feuilles de style, formatage

Chapter 1

Introduction

This document is the second part of a report that explains Tralics and how it can be used for the Raweb. You will find a sequence of commented source files, most of them publicly available on the Web, followed by a large index (over twenty pages).

The index is sorted alphabetically; in the case of `<a>`, or `$c` or `\z@`, the first character is not used for sorting; the object `\@car` can be found with the character 'c'; some private TeX commands have an at-sign in their name, sometimes in the middle, and sometimes at the start.

If you look at the letter Y, you can see five occurrences of the word 'year'. The first item uses a normal font; we use such a font for an attribute value (for instance, in the Raweb DTD, line 292, you can see that the `<citation>` element has an attribute from whose value can be 'year'). The second item uses a sans-serif font, this is the name of an attribute (for instance of the main Raweb element has a year attribute). Other items in the index use a type-writer font, and additional characters. For instance `<year>` is the name of a HTML or XML element, `$year` is the name of a xsl variable, and `\year` is the name of a TeX command (this one is not used here).

If you look at the letter I, you will find id, `id`, and ID and `ID`. The case of `ID` is a bit special since it is the name of an attribute set (and only one such data structure is used); and ID is an attribute type. We use id for the name of an attribute, and `id` for an xsl template (whose purpose is to set the attribute). The index contains also entity names like `%list;`, such entities appear only in the DTD.

We do not index everything! Consider a source line, of the form

```
1371 \XMLNSAX{fo}{break-before}{\FObreakbefore}{auto}
```

The number in the left margin has nothing to do with the line number of the source document; it will be referenced to in some cases. This line is to be interpreted by TeX, it has 28 tokens. However, for a human reader, it contains five tokens, that could be indexed. The first token is the name of a TeX command that appears more than a hundred times (with and without the trailing X); it will not be indexed, as well as other common TeX commands like `\def`, `\gdef`, `\XMLelement`, or `<xsl:template>` in the part concerning style sheets, and `<!ELEMENT>` and `<!ATTLIST>` in the last chapter. The 'fo' in this example is a prefix that appears almost always after `\XMLNSA` or `\XMLNSAX`, thus it does not appear in the index. On the other hand, the attribute name, the associated TeX command and the default value will be found in the index. Consider a second example

```
1503 <xsl:variable name="Directory" select="concat($LeProjet,$year)"/>
```

Here, the line contains seven tokens. There is an element and two attributes. These will not be found in the index (the attribute name name will only be indexed if it appears in the HTML or XML document, not in the style sheet as xsl keyword). Xsl functions like 'concat' will not be indexed.

On the other hand you will find the three variables `$Directory`, `$LeProjet` and `$year` (in the line above, one variable is set, the other two variables are used).

The first chapter describes xmltex.tex. This is a TEX file whose purpose is to read and evaluate an XML file. The interpretation depends on some user commands, to be put in a .xmt file (the "user" here being the guy who designs the DTD of the XML file, as opposed to the author of the document, or the author of xmltex). The file contains a lot of commands of the form `\expandafter`, `\csname`, `\edef`, and the like, that are not described in standard LATEX books. If you understand this file, you can be called a TEX Master (according to the TEXbook, a Master is somebody who understand tables, a Grandmaster is somebody who can design output routines; the whole XML stuff described in this report is somewhere between these two levels). It is however a challenge for a software like Tralics to be able to read the xmltex.tex file.

Using xmltex is easy. For instance, Chapter 3 explains how maths can be interpreted (this is an extension of the work of Carlisle, the author of xmltex). We have added commands that interpret the picture environment, and some extensions; the only difficulty here is that the commands have an irregular syntax, so that the standard mode of evaluation cannot be used (for instance, if you say

```
<oval x='1.2' y='3.4' specs='lt'> Text</oval>
```

we must call the associated TEX command like this

```
\oval(1.2,3.4)[lt]{Text}
```

rather than

```
\oval{1.2}{3.4}{lt}{Text}
```

Perhaps, the easiest way would be to write an intermediate command. The code is only given as an example of what can be done; it is not completely tested, and not used for the Raweb at all, because we do not know how to convert it into HTML.

Chapter 8 is an addition to version 2 of this document. It explains how to convert a document (like a PhD thesis, or a technical report) into a HTML document, after conversion into XML. We show how to solve a non-trivial problem: there are are objects, similar to `<oval>` above, that cannot be rendered in HTML, and have to be replaced by images: these are obtained by creating an auxiliary XML file, evaluating it by LATEX, converting the dvi into a sequence of images.

A similar idea is used in the Kraken software by Nader Salman: in this case, the XML file contains `<math>` and `<cite>` elements, containing LATEX code; a script extracts this code, calls Tralics, and reinserts the math formulas; the `<cite>` elements are replaced by pointers to the bibliography, generated as a by-product by Tralics; this is a rather original way to produces an activity report, see http://www-sop.inria.fr/odyssee/.

Chapter 2

Interpreting XML in TeX

We shall describe here the xmltex.tex file. This is a piece of code written by David Carlisle as described in [1], it is a follow-up to typehtml, a package for typesetting HTML. The idea was to write a TeX file that interprets some XML code and typesets it, using rules defined in some other files (the so-called .xmt files), that depend on the DTD or namespaces. Some of these files are described in following chapters. This interpreter is used for the production of the Raweb (on figure 1 of the first part of this document, the arrows from 'xmlfo' to 'PDF' or 'PS' use this file). The XML file contains a lot of Unicode characters, that can be coded using iso-latin1 or UTF-8 encoding. Interpreting them in TeX is a real challenge. Here is an example:

```
<m:math overflow="scroll">
  <mrow xmlns="http://www.w3.org/1998/Math/MathML">
   <msup><mi>L</mi> <mn>2</mn> </msup><mo>&#x2192;</mo>
    <msup><mi>L</mi> <mi>&#x221E;</mi> </msup></mrow></m:math>
```

In this example, only ASCII characters are used, complicated things are written in the form `&#x221E;`, this is the same as `&#8734;`. The `<mrow>` element has an xmlns attribute, it is hence the same as `<m:mrow>`. The action associated to this element is stored in some command, so one question is: what's the name of this command? every Unicode character is allowed, i.e., much more than the 256 internal characters of TeX. We have another problem, it is that an element name could be entered as `<José>` (iso-latin1 encoding). A good encoding is UTF-8, since it allows encoding of all Unicode characters on 8 bits. For instance, the representation of a character looks like this: `\8:Ã©`, and that of the element is `\E:0:JosÃ©`. Here between the two colons we have the value of the namespace, a sequence of digits, where 0 represents the empty namespace, 3, the MathML namespace, etc. (namespaces are defined in [11]). All commands that are dynamically created start with a prefix. This is '`8:`' for a UTF-8 character, '`A:`' for the global attribute list, '`E:`' for the start of an element, '`E/:`' for the end of an element, '`Q:`' for a processing instruction like `<?xml?>`, '`XML:`' for a namespace. An example of such a namespace command is `\XML:http://www.w3.org/1998/Math/MathML`. Usual LaTeX commands contain only letters, reserved names may contain @; using a prefix with a character other than these reduces the risk of conflict with existing commands. We must use `\csname` for producing these commands. The math formula above is $L^2 \rightarrow L^\infty$.

In order to make the code easier to understand, we have invented some commands that are inlined in the real code (for efficiency reasons). The command `\XML:http.../MathML` (full name shown above) contains the unique identifier for the MathML namespace; this is in fact the number 3. It can be constructed via `\jg@NSuri`. In the case of `<m:math>`, the value of the 'm' prefix is the same number, it will be obtained via `\jg@namespace{m}`. In fact, when we parse an element, the prefix is in a global variable, so that we can use the parameterless command `\jg@this@namespace`.

```
\def\jg@NSuri#1{\csname XML:#1\endcsname}
\def\jg@namespace#1{\csname XMLNS@#1\endcsname}
\def\jg@this@namespace#1{\jg@namespace{\XML@this@prefix}}
```

In some cases, we need a canonical version of a string. We shall use the `\catxii` command for this: if `\val` is a command that expands to '`some/val`' then the expansion of '`\meaning\val`' is '`macro:->some/val`', this is a list of character tokens with category code 12 (except for spaces). The `\strip@prefix` command removes everything up to '`>`', it yields '`some/val`'. We need an `\expandafter` for changing the order of expansion.

```
\def\catxii#1{\expandafter\strip@prefix\meaning#1}
```

See the TEXbook [6] for details about `\expandafter`, category codes, the result of `\meaning`, what happens if `\meaning` produces no greater-than sign, etc. See the LATEX source code for `\strip@prefix`. See the Unicode book [8], paragraph 2.5, for the definition of encodings like UTF-8, UTF-16 and UTF-32, and the whole book for the significance of characters U+221E.

UTF-8 encoding is defined as follows. A character X will be represented using a variable number of bytes, say A, AB, ABC or ABCD. Let x be the integer value of X, and a, b, c and d the values of A, B, C, and D. The first byte indicates the length of the sequence: if $a < 128$, the sequence is of length one, and $x = a$. Otherwise, a starts with k bits 1 followed by a 0 bit, the sequence is of length k, the $k-1$ characters that follow start with a 1 and a 0 (and have 6 significant bits). These are the relations we shall use:

$$
\begin{array}{lll}
\text{If } 0 \le x < 2^7 & k=1 & x = x_1 \quad a = x \\
\text{If } 2^7 \le x < 2^{11} & k=2 & x = x_1 2^6 + x_2 \\
 & & a = 128 + 64 + x_1, b = 128 + x_2 \\
\text{If } 2^{11} \le x < 2^{16} & k=3 & x = x_1 2^{12} + x_2 2^6 + x_3 \\
 & & a = 128 + 64 + 32 + x_1, b = 128 + x_2, c = 128 + x_3 \\
\text{If } 2^{16} \le x < 2^{21} & k=4 & x = x_1 2^{18} + x_2 2^{12} + x_3 2^6 + x_4 \\
 & & a = 128 + 64 + 32 + 16 + x_1, b = 128 + x_2, c = 128 + x_3, d = 128 + x_4
\end{array}
$$

In all cases $0 \le x_i < 64$. The case $x \ge 2^{21}$ is not handled. As an example, the character with code 233 is coded as `Ã©`. Note: assume that X is an iso-latin1 character, if it fits on seven bits it is represented by itself. Otherwise, if $128 \le x < 128 + 64$, the first character is `Â`, the second is X, and if $x \ge 128 + 64$ the first character is `Ã`, the second has value $x - 64$ (note that most useful latin1 characters are in the range 192-255).

The file we consider starts like this:

```
1 %% Copyright 2000 David Carlisle, NAG Ltd.
2 %% re-released by Sebastian Rahtz June 2002
3 %% This file is distributed under the LaTeX Project Public License
4 %% (LPPL) as found at http://www.latex-project.org/lppl.txt
5 %% Either version 1.0, or at your option, any later version.
```

Unless told otherwise, newline characters are ignored in the xmltex file (in particular, on line 6, the space after the opening brace). More generally, lots of characters have category codes that depend on the context. We have not shown all these category changes; since definitions are in local groups, they are generally global (hence the `\gdef` here).

2.1 Constructing characters

Let's consider the following task: We have a character X, with code x, and x is in `\count@`. We want to find the bytes ABCD, with codes a, b, c and d. These quantities are obtained by writing x in base 64, with digits x_i, and we add 128 to everything. The first byte is a bit more complicated to compute. This piece of code uses two temporary registers `\@tempcnta` and `\@tempcntb` for the division. It replaces x by its quotient, and puts in the `\uccode` of `‘#1` the next byte (this assumes that the argument of the command is a character).

```
\gdef\XML@utfeight@a#1{
     \@tempcnta\count@
     \divide\count@64
     \@tempcntb\count@
     \multiply\count@64
     \advance\@tempcnta-\count@
     \advance\@tempcnta"80
     \uccode‘#1\@tempcnta
     \count@\@tempcntb}
```

This is the caller of the preceding command:

```
\gdef\XML@charref#1#2;{
  \begingroup
  \uppercase{\count@\if x\noexpand#1"\else#1\fi#2}\relax
  \ifnum\count@<"80\relax
    \uccode‘\~\count@
    \uppercase{
    \ifnum\catcode\count@=\active
      \gdef\XML@tempa{\utfeightay~}
    \else
      \gdef\XML@tempa{~}
    \fi}
  \else\ifnum\count@<"800\relax
     \XML@utfeight@a,
     \XML@utfeight@b C\utfeightb.,
  \else\ifnum\count@<"10000\relax
     \XML@utfeight@a;
     \XML@utfeight@a,
     \XML@utfeight@b E\utfeightc.{,;}
   \else
     \XML@utfeight@a;
     \XML@utfeight@a,
     \XML@utfeight@a!
     \XML@utfeight@b F\utfeightd.{!,;}
    \fi
    \fi
  \fi
  \endgroup}
```

There is a similar command, except that the test on lines 21-25 is assumed to be true, and code on line 22 is executed. It seems to be used only for reading auxiliary files in XML format; however, the .aux files contain currently no XML code.

```
\gdef\XML@charref@tex#1#2;{
  ...}
```

We shall see in the sequel some instances of 'black magic'. The result of '`\uppercase{xe9}`' is '`XE9`'. However, if you say '`\uppercase{\foo~}`' the result is '`\foo W`', where W is the character found in the uc table of the tilde character, and the category code of this character is the same as the tilde character (in general active). The substitution is done before `\foo` is evaluated[1]. In some cases, `\foo` is `\endgroup`. In our case, the group ends at line 41. The `\uppercase` on line 17 is not black magic. The idea is the following: imagine that we want to read something like '`&#233;`' or '`&#xe9;`', and that the ampersand and sharp characters have been read. Then `\XML@charref` reads all characters up to the semi colon. Arguments are 2, 33 in one case, x, e9 in the other case. A construction like `\count@="E9` puts 233 into `\count@`, upper case letters are needed. What `\uppercase` produces in our example is '`\count@\if X\noexpand X"\else X\fi E9`'; there is a `\relax` in the code whose purpose is to mark the end of the number (we do not want the `\ifnum` to be expanded before assignment is done); this `\relax` command could have been in the uppercase list. I don't know if `\noexpand` is needed here[2]. The effect of the conditional is just to replace the X by a `"` (you cannot do this using black magic, because the double quote has to be of category 12, so that the argument of the command must be of category code 12). What our code does is to put the number (say 233) into `\count@`. It chooses one of four alternatives on line 18, 26, 29 and 33; it corresponds to the number of bytes used to represent the Unicode character in UTF-8 format. In any case, the result is a definition of `\XML@tempa` as a command that start with `\UTF8?` (this is a shorthand for one of `\UTF8ax`, `\UTF8ay`, `\UTF8az`, `\UTF8b`, `\UTF8c`, or `\UTF8d`, the real name of the command is `\utfeightax`, etc.) followed by some characters (1, 1, 1, 2, 3, and 4 respectively).

Let's start with the case of one byte, lines 19-25. We have a special case here, because the UTF-8 character can be represented by a single TEX character; we use it, in the case where it is not expandable (i.e., is non active); the code on lines 41-42 does not use this simplification. As an example, if the number x is 65, then `\XML@tempa` will contain `A`; if it is 60, it will contain '`\utfeightay<`' (we assume that the less-than sign is of category code 12 when the code is read, this is needed on line 18, and of category 13 when the code is executed).

In the case where more than one byte is used, the idea is the following. We have to compute some integers a, b, c and d (two three or four values are required). These integers are in the range 1–255. If we store them in the uc-slot of A, B, C or D, then `\uppercase{ABCD}` will give a sequence of four characters, whose codes are the numbers a, b, c and d. Instead of these letters, point, exclamation point, comma and semi colon are used, in a random order. This is completely irrelevant since modifications are local (the group ends on line 41), and the `\uppercase` on line 47 sees only these character tokens, together with non-character tokens that are not affected. The code could be slightly optimized if, on one hand, we notice that a is always stored in '.' (point) and, on the other hand, that b could always be stored in '!' (exclamation point). On lines 26 to 37 we compute b, c and d, and call `\XML@utfeight@b` with four arguments. Argument `#3` is the character that will hold a, argument `#4` is the list of characters that are already set, argument `#2` is the command name, one of the `\UTF8?` commands mentioned above. The first argument is a C, E, or F. Remember that $a = x_1 + s$, where x_1 is in `\count@`, s depends on the number of bytes. It is sixteen times 12, 14 or 15 (in base 16, it is C0, E0 or F0). What the next function does is then obvious:

```
44 \gdef\XML@utfeight@b#1#2#3#4{
45     \advance\count@"#10\relax
46     \uccode`#3\count@
47     \uppercase{\gdef\XML@tempa{#2#3#4}}}
```

Assume that our number is 233 (or E9, in base 16). We have $x_1 = 3$ and $x_2 = 41$. This gives $b = 128 + 41$, stored in the `\uccode` of `#4`. This is the character ©. Here `#1` is C, `"#10` is 192.

[1] The `\uppercase` command cannot be expanded; however its evaluation is a sequence of tokens that will be read again, expanded, and evaluated.

[2] The first character should be X or a digit, otherwise, the XML source is invalid. If you replace xe9 by xy9 or ye9, strange errors may be signaled; the case of a non-ASCII character is worse

Thus we store 195 (this is the code of Ã) in the `\uccode` of `#3`. Thus, the effect of the uppercase is to define the command `\XML@tempa` (this is a temporary command name that any command may redefine), it takes no argument, expands to `\utfeightbÃ©`. The important point to remember: `\XML@charref` puts in `\XML@tempa` a list of tokens, this list is independent of the context, but the commands in the list have a meaning that depends on the context (redefined by the commands defined in the next paragraph).

2.2 Using UTF-8 characters

The piece of code that follows defines the six commands `\UTF8?` (there are other versions of the same commands). These definitions are useful in a context where we evaluate a piece of text.

```
\def\unprotect@utfeight{
  \let<\XML@lt@markup
  \let&\XML@amp@markup
  \def\utfeightax##1{
    \csname 8:\string##1\endcsname}
  \let\utfeightay\utfeightax
  \let\utfeightaz\utfeightax
  \def\utfeightb##1##2{
    \csname 8:##1\string##2\endcsname}
  \def\utfeightc##1##2##3{
    \csname 8:##1\string##2\string##3\endcsname}
  \def\utfeightd##1##2##3##4{
    \csname 8:##1\string##2\string##3\string##4\endcsname}}
```

For instance, `\utfeightbÃ©` expands to `\csname8:Ã\string ©\endcsname`. We shall see in a minute why all characters have to be protected, except the first one. If we expand this, we get the command with this strange name `\8:Ã©`. This command is assumed to typeset the Unicode character 233. Its definition could be, for instance, '`\ifmmode \acute{e}\else \'{e}\fi`'. Such a definition is valid only in a context where we typeset the object. Inside an `\edef`, the expansion of the conditional may give random results, inside a `\csname`, some tokens are illegal. Note that, in this command, less-than and ampersand are active, they scan something in the XML file; they should be input as '`<`' or '`&`' if you want a typeset `<` or `&`.

The next command looks funny:

```
\gdef\UnicodeCharacter#1#2{
   \begingroup
   \def\active{\catcode\count@}
   \XML@charref#1;
   \expandafter\expandafter\expandafter
   \expandafter\expandafter\expandafter
   \expandafter
    \gdef\XML@tempa{#2}
  \endgroup}
```

There are seven `\expandafter` in a row. Write `\E` instead, in order to gain space. Assume that we have a command `\A` that expands to `\B` that expands to `\C` that expands to `\D`. The expansion of '`\E\E\E\E\E\E\E\gdef\A`' is '`\E\E\E\gdef\B`'. This expands to '`\E\gdef\C`'. This expands to '`\gdef\D`'. Suppose that we say `\UnicodeCharacter{233}{\'e}`. In this case `\XML@charref` will define `\XML@tempa` as shown above. This is our `\A`. The expansion `\B` is `\utfeightbÃ©`. Its expansion `\C` is `\csname...`, its expansion `\D` is `\8:Ã©`. Hence, the code is `\def\8:Ã©{\'e}`. Thus, we know how to define every Unicode character. There is a little hack here (on line 63, you

see why?). Characters like A, B, C, typeset to themselves. But some other characters have to be defined. We say for instance

```
70 \UnicodeCharacter{94}{\textasciicircum}
71 \UnicodeCharacter{x5C}{\textbackslash}
72 \UnicodeCharacter{x5F}{\textunderscore}
73 \UnicodeCharacter{13}{ \ignorespaces}
74 \UnicodeCharacter{32}{ \ignorespaces}
75 \UnicodeCharacter{9}{ \ignorespaces}
```

These definitions come from the xmltex.tex file, and the Raweb redefines the character U+5C, so as to allow it in math mode also. The definition of characters 9, 13 and 32 (spaces) is a bit strange: the `\ignorespaces` command expands the next token, and removes it, if it is a space; hence spaces given in the form `&#32;` are not removed. Worse: '`\parindent = 12 cm`' becomes illegal if what follows the equals sign comes from an XML file. The xmltex.tex file also has these definitions[3].

```
76 \expandafter\def\csname8:\string<\endcsname{\ifmmode\langle\else\textless\fi}
77 \expandafter\def\csname8:\string>\endcsname{\ifmmode\rangle\else\textgreater\fi}
78 \expandafter\def\csname8:\string{\endcsname{\{}
79 \expandafter\def\csname8:\string}\endcsname{\}}
```

What does the test on line 21 do? it compares the category code of `\count@` with `\active`; this is 13, and the test is false in the cases shown above (well, the backslash may be active while reading the XML file, it is surely not while processing line 71). Redefining `\active` has as side effect that it will expand to '`\catcode\count@`' and this is the same as `\catcode\count@`. As a consequence '`\XML@tempa`' expands to '`\utfeightay^`' that expands to `\csname...` that expands to `\8:^`. Hence, line 70 defines the command `\8:^`. This is what is desired. Note: when the XML file is read, all characters with code ≥ 128 are active, those with code ≤ 31 have category 12 (in fact, they are invalid in XML1.0).

The xmltex.tex file starts like this (before category codes of usual characters have been changed).

```
80 \count@0
81 \catcode0=13
82 \gdef\XML@tempa{
83  \begingroup
84    \uccode0\count@
85   \uppercase{\endgroup
86     \edef^^@{
87       \ifnum\catcode\count@=11 %
88         \noexpand\utfeightay\else\noexpand\utfeightax\fi
89       \noexpand^^@}
90     \expandafter\edef\csname 8:\string^^@\endcsname{\string^^@}}
91  \ifnum\count@<127\advance\count@1 \expandafter\XML@tempa\fi}
92 \XML@tempa
93 \catcode0=9
```

Here we have real magic. There is a loop over all numbers x between 0 and 127. The number x is in `\count@`. For each x, code on lines 83–90 are executed. The null character (number zero) is active, and its uc value is x. In lines 86–90, it will be replaced by the character x. Note that this character is input as `^^@`. Assume for instance that $x = 65$, so that it represents the letter A, or that $x = 61$ (character '`=`'). The second `\edef` defines `\8:A` or `\8:=` to be A or = (note: the purpose of the `\edef` is to expand the `\string` in the body, so that the character in the body is a non-active character). Hence the effect is the same as `\UnicodeCharacter{65}{A}`.[4] The purpose

[3] These definitions of less-than and greater-than are wrong; they will be redefined later.

[4] The code on lines 70–79 overrides some of these settings

of the `\edef` on line 86 is the expansion of the conditional: we define A to be '`\utfeightay A`', and = to be '`\utfeightax=`'. The character after the command is active. Consider this:

```
\def\use@utfeightay{...}
\use@utfeightay ^^M ^_~%$#{}
```

We have simplified a bit the code. The idea is that, for the characters listed here, `\utfeightay` is used instead of `\utfeightax`. We shall see later that `\utfeightaz` is used for ampersand and less than in a case like `&` and `<`.

The following piece of code defines the commands `\UTF8?` (version two).

```
\def\utfeight@protect@internal{
  \let\utfeightax\noexpand
  \let\utfeightay\noexpand
  \def\utfeightaz{
    \noexpand\utfeightaz\noexpand}
  \let<\relax\let&\relax
  \def\utfeightb##1##2{
    \noexpand\utfeightb##1\string##2}
  \def\utfeightc##1##2##3{
    \noexpand\utfeightc##1\string##2\string##3}
  \def\utfeightd##1##2##3##4{
    \noexpand\utfeightd##1\string##2\string##3\string##4}}
```

What happens if a UTF8 character appears in an `\edef`? For instance, the character '`é`', represented as '`\utfeightb Ã©`' expands to the expansion of '`\noexpand\utfeightb Ã\string©`', namely '`\utfeightbÃ©`'. The only thing that might have changed is the category code of `©`. If it was active, it is now 12 (remember, the first character is never active). In the case `\utfeightay A`, the expansion is `A`, because `\utfeightay` is `\noexpand`. In the case of `\utfeightaz W`, the expansion is itself! Note that '`<`' and '`&`' are not modified.

This is version three:

```
\def\utfeight@protect@external{
  \def\utfeightax{
    \noexpand\noexpand\noexpand}
  \let\utfeightay\utfeighta@ref
  \let\utfeightaz\utfeighta@ref
  \edef<{\string<}
  \edef&{\string&}
  \def\utfeightb##1##2{
    ##1\string##2}
  \def\utfeightc##1##2##3{
    ##1\string##2\string##3}
  \def\utfeightd##1##2##3##4{
    ##1\string##2\string##3\string##4}}
```

In such a case, the expansion of '`\utfeightb Ã©`' is '`Ã©`' where both characters are of category code 12. This is very interesting in the case of `\write` that expands everything. The string `Ã©` is the UTF-8 representation of é, and can be read again without trouble.[5] The expansion of '`\utfeightax~`' is '`\noexpand~`'. It will become `~` after another expansion. In the case of '`\utfeightax A`', the expansion is '`&#65;`' because of the following lines:

```
\def\utfeighta@ref#1{
  \string&\string##\number\expandafter`\string#1\string;}
```

[5] The trouble is that, when the aux file contains `\newlabel`, its argument uses `\csname`, this expands to é, and this does not match the UTF-8 character.

Version four: this is the easy version: everything is converted into characters, of category code 12; in this case Unicode characters can be used inside a `\csname`.

```
\def\utfeight@protect@chars{
  \let\utfeightax\string
  \let\utfeightay\string
  \let\utfeightaz\string
  \def\utfeightb##1##2{
    ##1\string##2}
  \def\utfeightc##1##2##3{
    ##1\string##2\string##3}
  \def\utfeightd##1##2##3##4{
    ##1\string##2\string##3\string##4}}
```

2.3 Warnings

This piece of code is used in cases where we want to print something. It is the last definition of the `\UTF8?` series.

```
\def\utfeight@protect@typeout{
  \utfeight@protect@chars
  \let<\relax
  \let&\relax}
```

This is the piece of code that removes the traces.

```
\def\xmltraceoff{
  \global\let\XML@trace@warn\@gobble
  \global\let\XML@trace@warnNI\@gobble
  \global\let\XML@trace@warnE\@gobble
  \global\let\XML@attrib@trace\relax}
```

These are the commands that print a warning. We simplified a bit the code by removing (here) the body of some commands, and (elsewhere) calls to trace.

```
\def\XML@warnNI#1{
  {\let\protect\string\utfeight@protect@typeout\message{^^J#1}}}
\def\XML@warn#1{
  {\let\protect\string\utfeight@protect@typeout\message{^^J\XML@w@#1}}}
\def\XML@attrib@trace{...}
\def\XML@doattribute@warn#1#2#3{...}
\let\XML@trace@warn\XML@warn
\let\XML@trace@warnNI\XML@warnNI
\let\XML@trace@warnE\message
```

2.4 Reading the text

The next lines of code define a command `\nfss@catcodes`, such that, when executed, all characters have standard category codes. The @ character is a letter, quotes, less-than greater-than and equals-to are of category other.

```
 \def\nfss@catcodes{
  \catcode`\\0
  % Idem for {}%^@#"'<=>
  }
```

This changes even more category codes. Dollar, ampersand, hat, underscore, space have standard category codes, others have category 12.

```
\def\XML@reset{
  \nfss@catcodes
  % reset $&^_ space
  % reset :!=|
  \catcode`\~\active\def~{\nobreakspace{}}
  \let\XML@ns@a@\XML@ns@a@tex
  \let\XML@ns\XML@ns@tex}
```

The next lines of code define a command `\XML@catcodes`, such that, when executed, all characters have category codes useful for reading an XML file.

```
\def\XML@catcodes{
  \catcode`\ \active
  % same for: ^^M ^^I <>:[]%&"'=
  % same for: /!?-${}#_\~
  \def~{\utfeightay~}
  \let\XML@ns@a@\XML@ns@a@xml
  \let\XML@ns\XML@ns@xml
}
```

The following two commands are inlined for efficiency reasons. We have introduced them in order to gain space.

```
\def\Normalspace{\catcode`\^^I=10 \catcode`\^^M=10 \catcode`\ =10 }
\def\Activespace{\catcode`\^^I=13 \catcode`\^^M=13 \catcode`\ =13 }
```

This piece of code does a loop, starting with `\count@`, up to `\@tempcnta` (excluded). The loop puts the current number in the uc-code of tilde, and uppercasifies the value of `\XML@tempa`, to be defined later, in the form `\def\XML@tempa{{...}}`, double braces are needed because `\uppercase` want a brace-delimited list of tokens.

```
\gdef\utfeightloop{
  \uccode`\~\count@
  \expandafter\uppercase\XML@tempa
  \advance\count@\@ne
  \ifnum\count@<\@tempcnta
  \expandafter\utfeightloop
  \fi}
```

We leave it as an exercise to the reader to define a command `\XML@utfeight` whose expansion is '`utf-8`', all characters being of category code 12. This piece of code does nothing if the current encoding is '`utf-8`', otherwise it sets the current encoding to '`utf-8`', and does some action.

```
\gdef\XML@setutfeight{
  \ifx\XML@utfeight\XML@thisencoding
  \else
    \let\XML@thisencoding\XML@utfeight
    ...% see below
  \fi}
```

This is the action: for every character that is the first in a sequence of 2, 3 or 4 characters, it defines the character (for instance `Ã`) to take 1, 2 or 3 arguments. For instance `Ã` is defined as `\utfeightb Ã#1`. The first argument to `\utfeightb`, `\utfeightc`, or `\utfeightd` is not active! This works, because `\string~` is expanded to `~` of category code 12, where `~` is replaced by the `\uppercase` on line 174 by the character (for instance `Ã`), the `\utfeightb` command is not expanded since preceded by a `\noexpand`. The definition is in a double group (`\begingroup` on

line 185, braces on lines 188, 192, 194). The definition is visible outside the group because it is global: we use `\xdef`. We could replace `\gdef` by `\def` here, whether the temporary is restored or not after the loop is irrelevant.

```
185   \begingroup
186   \count@"C2
187   \@tempcnta"E0
188   \gdef\XML@tempa{{\xdef~####1{\noexpand\utfeightb\string~####1}}}
189   \utfeightloop
190   \count@"E0
191   \@tempcnta"F0
192   \gdef\XML@tempa{{\xdef~####1####2{\noexpand\utfeightc\string~####1####2}}}
193   \utfeightloop
194   \@tempcnta"F4  \gdef\XML@tempa{{\xdef~####1####2####3{%
195                               \noexpand\utfeightd\string~####1####2####3}}}
196   \utfeightloop
197   \endgroup
```

This defines a command named `\Q:xml`. It calls `\XML@xmldecl` after having changed the category code of white space.

```
198 \expandafter\gdef \csname Q:xml\endcsname{
199   \Normalspace
200   \XML@xmldecl}
```

This resets some category codes and calls `\XML@encoding`. The argument is something strange. The idea is that we parse `<?xml foo='bar'?>`. We read everything up to the end of the element, and provide a default encoding attribute. A `\relax` marker is put at the end.

```
201 \gdef\XML@xmldecl#1?>{
202   \Activespace
203   \XML@encoding#1 e="utf-8"\relax}
```

The XML norm (see for instance [9, 10, 3]) says (rules 23, 24, 32, 80) that in `<?xml?>` only the encoding attribute can start with the letter 'e'. This makes the loop easy. Note: other attributes are version, currently ignored (there are two versions of the XML standard, and the difference between them is tiny), and standalone (completely ignored).

```
204 \gdef\XML@encoding#1 #2{
205   \if\noexpand#2e
206     \expandafter\XML@encoding@aux
207   \else
208     \expandafter\XML@encoding
209   \fi}
```

We shall see later that `\XML@quoted\foo` reads `'bar'` or `"bar"`, and calls `\foo` with the value '`bar`'. The following piece of code grabs the attribute name, the equals sign, reads the attribute value and calls another command.

```
210 \gdef\XML@encoding@aux#1={
211   \XML@quoted\XML@setenc}
```

Here the `\lowercase` is no magic: the XML norm says (rule 80) that all characters should be ASCII characters (letters, digits, dot, underscore, dash), and case independent. In the case where the encoding is not UTF-8, some file is read; for instance iso-8859-1.xmt. On page 6, we have seen how to find the UTF-8 representation of a latin1 character. It depends on whether the character is smaller or larger than 192. Two easy loops suffice to define all characters like é as `\utfeightbÃ©`.

```
212 \def\XML@setenc#1#2\relax{
213   \lowercase{\gdef\XML@tempa{#1}}
```

```
214   \xdef\XML@tempa{\catxii\XML@tempa}
215   \ifx\XML@tempa\XML@thisencoding
216   \else
217     \ifx\XML@utfeight\XML@tempa
218       \XML@setutfeight
219     \else...% code not shown here
220     \fi
221   \fi}
```

2.5 Namespaces

You say `\XML@ns@alloc{foo}` in order to declare 'foo' as a namespace name; after that the value can be found by `\jg@NSuri {foo}`. In the case this command is defined, there is nothing to do. Otherwise, we allocate a number using the counter `\XML@ns@count`, say 3, and put this in the command. We define two other commands: `\jg@namespace{3}` will be 3, and `\A:3` will be empty (we shall see that this is the global attribute list of the namespace). Note: we use here the pseudo commands `\jg@NSuri`, so that a double indirection is needed; as a consequence `\expandafter3` should be replaced by a sequence of three `\expandafter` tokens.

```
224 \def\XML@ns@alloc#1{
225   \expandafter3\ifx\jg@NSuri{#1}\relax
226     \global\advance\XML@ns@count\@ne
227     \expandafter3\xdef\jg@NSuri{#1}{\the\XML@ns@count}
228     \global\expandafter3\let\csname A:\the\XML@ns@count\endcsname\@empty
229     \expandafter3\xdef\jg@namespace{\the\XML@ns@count} {\the\XML@ns@count}
230   \fi}
```

The namespace stuff is initialized like this; number 0 corresponds to the empty namespace. Note that the recommendations say: The prefix 'xml' is by definition bound to the namespace name: http://www.w3.org/XML/1998/namespace.

```
231 \XML@ns@count-1
232 \XML@ns@alloc{}
233 \XML@ns@alloc{http://www.w3.org/1998/xml}
234 \def\XMLNS@xml{1}
235 \XML@ns@alloc{http://www.dcarlisle.demon.co.uk/xmltex}
```

The next piece of code is standard trick to convert '`foo:bar`' into `{foo}{bar}` and '`foo`' into `{}{foo}`. The auxiliary command sees '`bar`' or '`\@`' as second argument, argument 3 is junk. This works only if `\\` does not appear in the argument, moreover, it is recommended that at most one colon appears, and no `\@`, otherwise, two many tokens are considered as junk.

```
236 \gdef\XML@ns@xml#1{\expandafter\XML@ns@a@xml#1:\@:\\}
237 \gdef\XML@ns@a@xml#1:#2:#3\\{
238   \ifx\@#2 \XML@ns@b{}{#1}
239   \else    \XML@ns@b{#1}{#2}
240   \fi}
```

The function above depends on the category code of the colon character. We define an alternative version of the command and its helper, and install `\XML@ns@a@` to be the TEX variant, but it may be redefined (see lines 160 and 167).

```
241 \def\XML@ns@tex#1{...}
242 \def\XML@ns@a@tex#1:#2:#3\\{...}
243 \let\XML@ns@a@\XML@ns@a@tex
244 \let\XML@ns\XML@ns@tex
```

What this code does is just to expand everything (in order to get a canonical form). Thus `\XML@ns`, as well as all its variants, take a sequence like '`foo:bar`', puts it in a canonical form, and puts '`foo`' in `\XML@this@prefix`, '`bar`' in `\XML@this@local`.

```
\def\XML@ns@b#1#2{
  \begingroup
  \utfeight@protect@chars
  \xdef\XML@tempa{#1}
  \xdef\XML@tempb{#2}
  \endgroup
  \let\XML@this@prefix\XML@tempa
  \let\XML@this@local\XML@tempb
  }
```

2.6 Redefining \protect

In order to prevent premature expansion, you can insert `\protect` before a command; this makes it "robust"; the `\protect` command is defined in LaTeX, its value depends on the context. It may be `\@unexpandable@protect`, that is `\noexpand\protect\noexpand`. Hence `\protect\foo` expands to itself in an `\edef`. On line 96, we define a command so that `\utfeightb Ã©` (the internal representation of é) also expands to itself. In this section, we modify all context switch commands in order to make all UTF-8 characters naturally robust.

We start with a modified `\xdef` in which `\protect` and UTF-8 characters are left unchanged. This works well in a group because the end of the group restores the old value.

```
\def\unrestored@protected@xdef{
   \utfeight@protect@internal
   \let\protect\@unexpandable@protect
   \xdef
}
```

Another extension to LaTeX: Here everything is done in a group, the definition is global, the modifications to `\protect` and `\UTF8?` are local; the group is terminated after the `\xdef` because of the `\afterassignment`.

```
\def\protected@xdef{
   \begingroup
   \utfeight@protect@internal
   \let\protect\@unexpandable@protect
   \afterassignment\endgroup
   \xdef}
```

Yet another one: No group is used here, and `\afterassignment` gets another token as argument. This is useful if we do not want an `\xdef` (for instance, `\refstepcounter` uses this to define the current label). The meaning of UTF-8 characters is not restored, but reset to XML mode.

```
\def\protected@edef{
   \let\@@protect\protect
   \let\protect\@unexpandable@protect
   \utfeight@protect@internal
   \afterassignment\restore@protect
   \edef
}
```

We have to restore `\protect` and some other commands.

```
\def\restore@protect{\let\protect\@@protect
   \unprotect@utfeight}
```

We have to redefine `\protected@write`. This is a command that takes 3 arguments. It writes the last argument on the file defined by the first argument. Protection works as follows: there is an `\edef` that will expand all tokens but the protected ones, the current page reference (i.e., `\thepage`), including side-effects that come from evaluating the second argument. For instance, in Chapter 4, line 2445, there is an example where `\jgFOlabel` is set to `\relax`; the `\addtocontents` command defines `\label`, `\index` and `\glossary` to gobble their arguments.

```
\long\def \protected@write#1#2#3{
      \begingroup
       \let\thepage\relax
       #2
       \utfeight@protect@external
       \let\protect\@unexpandable@protect
       \edef\reserved@a{\write#1{#3}}
       \reserved@a
      \endgroup
      \if@nobreak\ifvmode\nobreak\fi\fi
}
```

We must also redefine this (it is used by `\typeout`).

```
\def\set@display@protect{
  \let\protect\string
  \utfeight@protect@typeout}
```

2.7 The catalogue

The catalogue is an association list, a sequence of the form `\key{val1}{val2}`. We have mentioned elsewhere that adding something at the end of a token list is not obvious. Here we proceed as follows. Consider

```
\edef\val{\noexpand\the\list\noexpand\key{\catxii\val}}
```

If we assume that `\list` is a command that cannot be expanded and `\val` expands to '`some/val`', the code above puts `\the\list\key{some/val}` into `\val`. Assume now that `\list` is a reference to a token list, and that we say

```
\list\expandafter\expandafter\expandafter{\val{aux}}
```

Since `\list` is a reference to a token list, the code above is an assignment, after `\list` we have a token list, and the first token is expanded to see if it is a left brace. Because of the `\expandafter` the code is equivalent to

```
\list\expandafter{\the\list\key{some/val}{aux}}
```

Now, the token that follows `\list` can be expanded; hence the result is the same as

```
\list{<value of the list>\key{some/val}{aux}}
```

We can also say something like

```
\list\expandafter{\the\expandafter\list\expandafter\key\val{aux}}
```

Here the effect of `\expandafter` is to expand `\the`; this expands the token that follows, namely the `\expandafter`, so that `\val` is expanded. This the result is the same as

```
\list{<value of the list>\key some/val{aux}}
```

The catalogue is a token list defined by sequence of assignments like this:

```
\SYSTEM   {http://www.oucs.ox.ac.uk/dtds/tei-oucs.dtd} {tei.xmt}
\NAMESPACE{http://www.w3.org/1998/Math/MathML}         {mathml2.xmt}
\NAMESPACE{http://www.dcarlisle.demon.co.uk/sec}       {sec.xmt}
\NAME{langtest}                                        {langtest.xmt}
\NAME{TEI.2}                                           {tei.xmt}
\NAME{html}                                            {html.xmt}
\NAMESPACE{http://www.w3.org/1999/XSL/Format}          {fotex.xmt}
```

Here the last item on each line is the name of a TEX file to load in some cases. There are five different items in the catalogue, thus five commands that put things in it, and five other commands that extract something. The action of `\FOO{A}{B}` is essentially to add `\XML@@FOO{A}{B}` at the end of the token list.

Let's start with the `\PUBLIC` command. It takes two arguments, an URI and a file name.

```
\def\PUBLIC#1#2{
 \xdef\XML@tempa{#1}
 \xdef\XML@tempa{\noexpand\the\XML@catalogue\noexpand\XML@@PUBLIC
             {\catxii\XML@tempa}}
 \global\XML@catalogue\expandafter\expandafter\expandafter{
   \XML@tempa{#2}}}
```

Same idea here.

```
\def\SYSTEM#1#2{
 \xdef\XML@tempa{#1}
 \xdef\XML@tempa{\noexpand\the\XML@catalogue\noexpand\XML@@SYSTEM
             {\catxii\XML@tempa}}
 \global\XML@catalogue\expandafter\expandafter\expandafter{
   \XML@tempa{#2}}}
```

In the case of a namespace, for instance MathML, we compute the namespace number of it, and the catalogue associates to this number the file in which everything is defined.

```
\def\NAMESPACE#1#2{
  \utfeight@protect@chars
  \XML@ns@alloc{#1}
  \edef\@tempa{{\jg@NSuri{#1}}}
  \global\XML@catalogue\expandafter{\the\expandafter\XML@catalogue
     \expandafter\XML@@NAMESPACE\@tempa{#2}}
  \unprotect@utfeight}
```

This is the easiest of all commands, since we do not have to do anything with the arguments.

```
\def\NAME#1#2{
 \global\XML@catalogue\expandafter{\the\XML@catalogue\XML@@NAME{#1}{#2}}}
```

You run the catalogue by evaluating it. For instance, if you put '`foo`' into `\XML@PUBLIC`, then the value associated to `foo` by the `\PUBLIC` command will be put in `\XML@use`.

```
\def\XML@@PUBLIC#1#2{
  \gdef\XML@tempa{#1}
  \ifx\XML@tempa\XML@PUBLIC \def\XML@use{#2}\fi}
```

Same action for `SYSTEM`. The temporary variable has a different name.

```
\def\XML@@SYSTEM#1#2{
  \def\@tempa{#1}
  \ifx\@tempa\XML@SYSTEM \def\XML@use{#2}\fi}
```

Same action for `NAMESPACE`.

```
\def\XML@@NAMESPACE#1#2{
  \def\@tempa{#1}
  \ifx\@tempa\XML@NAMESPACE  \def\XML@use{#2}\fi}
```

Same action for `NAME`.

```
\def\XML@@NAME#1#2{
  \def\@tempa{#1}
  \ifx\@tempa\XML@NAME  \def\XML@use{#2}\fi}
```

You say `\XMLNS{html}{http://www.w3.org/1999/xhtml}`. The effect is to associate to the name `html` the namespace value of the second argument, for instance 17.

```
\def\XMLNS#1#2{
  \utfeight@protect@chars
  \XML@ns@alloc{#2}
  \edef\@tempa{{#1}{\jg@NSuri{#2}}}
  \global\XML@catalogue\expandafter{\the\expandafter\XML@catalogue
     \expandafter\XML@@XMLNS\@tempa}
  \unprotect@utfeight}
```

This piece of code is a bit strange; it might produce unexpected results. The idea is the following. The command `\XML@checkknown` will run the catalogue in case of unknown elements. In the case of `<TEI.2>` or `<html>`, where no namespace prefix is given, the command `\XML@NAME` is set, and `\XML@@NAME` may define `\XML@use`. However, if `\XMLNS` has been defined as above, this piece of code is also executed: it defines a default namespace, in particular it could replace `<0:html>` by `<17:html>`. The last action is to define `\XML@NAMESPACE`, and we are ready to run the catalogue again.

```
\def\XML@@XMLNS#1#2{
  \def\@tempa{#1}
  \ifx\@tempa\XML@NAME
    \edef\XMLNS@{#2}
    \edef\XML@this@element{\XMLNS@\noexpand:\XML@this@local}
    \let\XML@NAMESPACE\XMLNS@
  \fi}
```

2.8 Reading elements

Let's start slowly. This piece of code is executed whenever we see a less-than sign, that is the start of an element. We have to distinguish between `</foo>`, `<?foo>`, `<!foo>` and `<foo>`. The procedure reads one character. What makes everything interesting is that `\fi` tokens are missing. On the other hand, we have inserted a `\@` marker, whose purpose is to skip easily over all useless tokens.

```
\def\XML@lt@markup#1{
  \Normalspace
  \ifx/#1\XML@getend
  \else\ifx!#1\XML@getdecl
  \else\ifx?#1\XML@getpi
  \else\XML@getname#1\@}
```

The function that follows is defined in an environment where space, newline, and tabulation are active characters (remember that `\endlinechar` is -1, so that newline characters are produced only via `^^M`). The code makes these characters active, and defines them; this action is local to a group (the groups ends on line 352). When we typeset some text, it is wise to activate these characters; on the other hand, spaces have normal category code when scanning attributes. In any case, space characters disappear at end of line, this explains the need of the `%` signs here. This

piece of code is called when the XML file contains `<foo>`; we have read the less than sign, and the letter that follows; the letter is in `#1` (if you look at line 340, you see that `#1` is nothing else than the first argument of `\XML@lt@markup`, because this cannot be `\@`. It could be `Ã`, i.e., the first byte of a Unicode character). We know that this is not slash, not an exclamation point, not a question mark, and we close these conditionals. The reader should take some time, in order to understand how `\XML@tempa` is defined.

```
\gdef\XML@getname#1\@{
\fi\fi\fi
\begingroup
\Activespace
\def {\iffalse{\fi}\XML@getname@}
\let^^M %
\let^^I %
\def/{\iffalse{\fi}\XML@getname@/}
\def>{\iffalse{\fi}\XML@getname@>}
\unrestored@protected@xdef\XML@tempa{\iffalse}\fi#1}
```

The last line contains `\unrestored@protected@xdef`; this is a command that modifies the behavior of some UTF-8 characters; it assumes to be in a group (thus the 'unrestored'); it evaluates to `\xdef` (see line 257). After TEX has seen the opening brace, all tokens are expanded; as a result the '`\iffalse}\fi`' is ignored; note that the '`\iffalse{\fi`' that appears in the definition of space, tabulation, newline, slash, greater-than sign disappears also; the full expansion of these commands is: a closing brace, `\XML@getname@`, and maybe one character. It is this closing brace that terminates the `\xdef`; said otherwise, `\XML@tempa` will contain everything up to these characters. In the case `<foo>`, `<foo/>`, `<foo a='b'>`, it will contain '`foo`'. In the case of `<José>`, in a document with latin1 encoding, it will contain '`JosÃ©`', where the `Ã` is an active character.

The `\XML@getname@` command is defined below. It closes the group in which space and other characters have a funny definition. The `\XML@begingroup` is a hack that saves some stack space. It has the same features as `\begingroup`. The `\XML@w@` command contains N spaces (where N is the current level). It is used for debugging, and argument grabbing. What the code does is: Put in `\begintag` the name of the element, in `\XML@parent` the current element (the parent of this one), initialize the current attribute list `\XML@attribute@toks` to the empty list, and parse the attributes.

```
\def\XML@getname@{
  \endgroup
  \XML@begingroup
  \edef\XML@w@{ \XML@w@}
  \let\begintag\XML@tempa
  \let\XML@parent\XML@this@element
  \XML@attribute@toks{}
  \XML@getattrib}
```

All these `\expandafter` in the code have as purpose to pop the conditional stack (said otherwise, if the command takes an argument, the argument will be what follows the `\fi`, not the `\fi` itself)[6]. There are two cases to consider: there is an attribute, or there is none. In the case where there is no attribute, there are two subcases: the element can be empty or not. If you say `<foo␣␣bar='gee'>`, the first space was active, and read by the magic above; the second one has category code 10, and is discarded because the argument to this command is not a delimited argument.

```
\def\XML@getattrib#1{
  \ifx#1/
    \expandafter\XML@endempty
```

[6] Guess: why is there no `\expandafter` after the second `\else`?

```
362   \else
363   \ifx#1>
364      \expandafter\expandafter\expandafter\XML@startelement
365   \else
366     \XML@getattrib@a#1
367   \fi
368   \fi}
369 \let\XML@@getattrib\XML@getattrib
```

In the case of `<foo bar='1'/>`, when the slash is seen, the greater sign is read, and `</foo>` is pushed back in the input stream. After that, we proceed as if there were no slash. This means that `<foo/>` is the same as `<foo></foo>`.

```
370 \def\XML@endempty#1>{
371   \expandafter\XML@startelement
372   \expandafter<\expandafter/\begintag>}
```

Here is a little trick: in the case where we are reading `<foo bar='1'>`, line 366 contains the command `\XML@getattrib@a`, followed by the letter b, followed by `\fi\fi`, followed by `ar='1'` (still unread). What we do is a trick to read an optional space before the equals sign (the space after the equals sign disappears because `\XML@quoted` uses an undelimited argument; a space in the attribute value will not disappear, because `\XML@qq` uses a delimited argument). We save the attribute name in a variable, and read the value.

```
373 \gdef\XML@getattrib@a#1\fi\fi#2={
374   \fi\fi
375    \XML@set@this@attribute#1#2 \@
376    \XML@quoted\XML@attribval}
377
378 \def\XML@set@this@attribute#1 #2\@{
379   \def\XML@this@attribute{#1}}
```

You say `\XML@quoted\foo'bar'` or `\XML@quoted\foo"bar"`. In both cases, `\foo` is called with `bar` as argument. In general, error handling is very poor. The purpose of `\ERROR` here is not to report an error in the case of wrong syntax. It will be used on line 550.

```
380 \def\XML@quoted#1#2{
381    \ifx#2"\expandafter\XML@qq
382    \else\ifx#2'\expandafter\expandafter\expandafter\XML@q
383    \else
384      \ERROR#2
385    \fi \fi #1}
386 \def\XML@qq#1#2"{#1{#2}}
387 \def\XML@q#1#2'{#1{#2}}
```

In order to make things easier to understand, write `\Att` instead of `\XML@this@attribute`, this is the attribute to analyze. Write `\AL` instead of `\XML@attribute@toks`, this is the resulting list to which tokens will be added. Before we forget it: this command terminates on line 407, with `\XML@getattrib`, hence continues parsing the attribute list. The normalized attribute value is compared to `\XML@ns@decl`, a command that contains '`xmlns`' with category codes 12. If the attribute name is '`xmlns`', this defines the default namespace, if the attribute is '`xmlns:foo`', this defines the namespace prefix 'foo' for this element and its content. In both cases, we call `\XML@ns@uri`. Otherwise if the attribute is foo:bar = 'gee' we add `\XML@doattribute{foo}{bar}{gee}` to the token list.

```
388 \def\XML@attribval#1{
389   \xdef\XML@tempa{\catxii\XML@this@attribute}
390   \ifx\XML@tempa\XML@ns@decl
```

```
391      \XML@ns@uri{}{#1}
392    \else
393      \XML@ns\XML@this@attribute
394      \edef\XML@this@prefix{\catxii\XML@this@prefix}
395      \ifx\XML@this@prefix\XML@ns@decl
396        \XML@ns@uri\XML@this@local{#1}
397      \else
398       \begingroup
399       \utfeight@protect@internal
400        \xdef\XML@tempa{
401           \the\XML@attribute@toks
402           \noexpand\XML@doattribute{\XML@this@prefix}{\XML@this@local}{#1}}
403        \endgroup
404        \XML@attribute@toks\expandafter{\XML@tempa}
405       \fi
406     \fi
407   \XML@getattrib}
```

Assume that we have xmlns:foo = 'http://www.w3.org/1998/Math/MathML'. This piece of code allocates a number for the URI if not already done. Let's assume that this number is 3. It then defines `\XMLNS@foo` to be 3. In the case `xmlns='...'` it defines `\XMLNS@`, the default namespace.

```
408 \def\XML@ns@uri#1#2{
409   \utfeight@protect@chars
410   \XML@ns@alloc{#2}
411   \expandafter3\edef\jg@namespace{#1}{\jg@NSuri{#2}}
412   \unprotect@utfeight}
```

The next macro is called when we see the greater-than sign that closes the opening tag of an element. We first do something with default attributes, this will be explained later. After that, we split the element name into '`foo:bar`'. Assume that the namespace number of '`foo`' is 4, we put in '`\XML@this@element`' the tokens '`4:bar`'. We execute a piece of code that can possibly load a file in which the element's behavior is defined, and then, we execute the associated code. We shall see later that there is a command `\xmlgrab` that reads everything (including subelements) up to some end tag. This command redefines `\XML@doelement`. This explains why `\XML@doelement` is not inlined.

```
413 \gdef\XML@startelement{
414   \XML@default@attributes
415   \Activespace
416   \XML@ns\begintag
417   \edef\XML@this@element{
418     \jg@namespace{\XML@this@prefix\expandafter}\noexpand:\XML@this@local}
419   \XML@checkknown
420   \XML@doelement}
```

The action associated to `<m:math>` is just to call `\E:3:math`. We shall see later how this command can be defined.

```
421 \def\XML@doelement{
422   \csname E:\XML@this@element \endcsname}
```

Assume that we want to evaluate `\E:3:math`. This routine does nothing if the command exists. Otherwise, it "runs the catalogue", and does a check: a warning is printed in case where the command does not exist. Assume first that the prefix is empty (has number 0). In this case, the catalogue can have an entry for the name (defined via the `\NAME` command), defining a file to load. Otherwise, the catalogue should have an entry for the namespace (here 3) defined via

`\NAMESPACE`. In any case, the catalogue should define `\XML@use` the name of a file to be loaded. We simplified a bit the code by introducing the `\jg@this@namespace` command; note that we could use `\XML@this@element`. Important note: on line 353, there is a command `\XML@begingroup`, so that all definitions from the included file are local to this group. If the current element is not the root element, and if you want the definitions to apply to all elements (and not only the descendants of this one), the definitions should better be `\global`.

```
\def\XML@checkknown{
  \expandafter\ifx
    \csname E:\jg@this@namespace:\XML@this@local\endcsname
    \relax
   \let\XML@use\@empty
   \ifnum0=\jg@this@namespace
     \let\XML@NAME\XML@this@local
     \the\XML@catalogue
   \else
     \edef\XML@NAMESPACE{\jg@this@namespace}
     \fi
   \let\XML@NAME\relax
   \the\XML@catalogue
   \inputonce\XML@use
   \expandafter\ifx\csname E:\jg@this@namespace\csname
       :\XML@this@local\endcsname\relax
     \XML@trace@warnE{Undefined}
   \fi
 \fi}
```

2.9 End of element

Look at lines 337-340. When we are reading `</foo >`, the `\XML@getend` command is called after the slash has been read. This little piece of code grabs all tokens, until the end of the command (argument `#1`, unused), and everything up to the greater-than sign (argument `#2`). It closes the conditional, and calls another command (the purpose of the call is to get rid of the final space). Why this changes the category code of space and not tabulation is beyond me.

```
\def\XML@getend#1\@#2>{
  \fi
  \catcode`\ \active
  \XML@getend@a#2 \@}
```

We have now read `</foo >`, and `\endtag` contains '`foo`'. We extract the namespace part, and call a command (that may be redefined in case of grab).

```
\gdef\XML@getend@a#1 #2\@{
  \Activespace
  \def\endtag{#1}
  \XML@ns\endtag
  \XML@doend}
```

The action associated to `</m:math>` is to call the command named `\E/:3:math`. After that, we have to close a group (opened on line 353).

```
\gdef\XML@doend{
  \csname E/:\jg@this@namespace:\XML@this@local  \endcsname
```

```
453   \XML@endgroup
454   \Activespace}
```

2.10 Using attributes

Consider `<X:elt foo:bar='gee' color='red' xmlns:X='myX'/>`. We have already seen that the effect of the xmlns:X attribute is to define X as a namespace for this element and its children. In the case foo:bar, the definition of 'foo' could come later. For this reason, when we parse the attribute list, we construct a list of the form `\do{a}{b}{c}`[7], in our case it contains `\do{foo}{bar}{gee}` and `\do{}{color}{red}`. This list is in `\XML@attribute@toks`. There is another list that contains terms of the form `\Att name\relax\cmd{val}action`[8], it depends on the behavior of the element `<X:elt>`. Imagine that any element in the `X` namespace has an attribute some:background, with a default value of black, and `<X:elt>` has an attribute color, with some default value, and that some action is associated to it. This second list is the argument of the macro whose definition follows. It is constructed by `\XMLelement`; this construction can occur because of autoloading of some package, and this depends on the current namespace, i.e., `myX`. What follows `\Att` is the name with its namespace, for instance `25:background` or `0:color`; it is followed by `\relax` and the name of the command in which the element can get the value; it is followed by the default value (in braces) and an action (a sequence of commands, generally empty). Let's assume that there is no action for the background, but `\checkcolor` for the color.

This piece of code evaluates both lists, with a double `\relax` at the end. We have to evaluate all namespaces, but there is no default namespace for attributes. For this reason we set `\XMLNS@` to 0.

```
455 \def\XML@setattributes#1{
456   \let\XMLNS@@\XMLNS@
457    \def\XMLNS@{0}
458    \the\expandafter\XML@attribute@toks#1\relax\relax
459    \let\XMLNS@\XMLNS@@}
```

Let's assume that our big list is the following `\do {foo} {bar} {gee} \do {} {color} {red} \Att 25:background\relax \setbg {black} \Att 0:color\relax \setcol {blue} \checkcolor \relax \relax`. We give here the definition of the command that evaluates the `\do`. It reads the three token lists that follow and defines a command that will be used twice. If this command is `\T`, then `\T\foo` applies `\foo` to some argument, namely `\Att name\relax`, where the character string between the two commands is the full name, for instance `17:bar` or `0:color`.

```
460 \def\XML@doattribute#1#2#3{
461   \xdef\XML@tempa##1{\noexpand##1{
462        \noexpand\XML@attrib\jg@namespace{#1}:#2\relax}}
463   \XML@tempa\XML@attrib@x{#3}
464   \XML@tempa\XML@attrib@y}
```

The command that follows is called twice in our example, with arguments '`\Att17:bar\relax`' and '`gee`', then '`\Att0:color\relax`' and '`red`'. The action is to define a command `\XML@tempb`, whose action is to read some tokens, and define some command to be '`gee`' or '`red`'. We shall see in a moment how this command is used. Putting a `\def` in a `\def` in a `\def` is unusual. There is a priori no reason why the second one should be global. The inner one has to be local.

```
465 \def\XML@attrib@x#1#2{
466     \gdef\XML@tempb##1#1##2##3##4\relax\relax{
```

[7] Here `\do` is short for `\XML@doattribute`
[8] Here `\Att` is short for `\XML@attrib`

```
467     \def##2{#2}
468     ##1##4\relax\relax}}
```

The command that follows is simple (its body has one line) but a bit subtle. Remember the long list of tokens shown above. It is of the form `\do...\do...\Att...\Att...\relax\relax`. We have read the `\do...`, and constructed a `\Att...`, which is in `#1`. Everything else is in `#2`. We apply `\XML@tempb` to the list, where the first `\do...` is removed and a new `\Att...` is added. Remember that `\Att` is followed by a name, then `\relax`, a command, a value, and maybe action. In `#1` we have only the name and `\relax`; we provide here `\XML@temp@l` as command name, and 6 as value. We show the code, then the explanations.

```
469 \def\XML@attrib@y#1#2\relax\relax{
470   \XML@tempb#2#1\XML@temp@l{6}\relax\relax}
```

Consider first the case where `#1` is '`\Att17:bar\relax`'. The command `\XML@tempb` takes four arguments, the first argument is delimited by `#1`, and the last by a double `\relax`. Our element `<X:elt>` knows nothing about 'foo:bar', so that the `#1` is the one provided on line 470. Thus, the first argument is `#2`, the second argument is `\XML@temp@l`, the third argument is '6', the last is empty. The effect of line 467 is to define `\XML@temp@l`, this is a dummy command, its definition is irrelevant. The effect of line 468 is to evaluate the long list again.

Consider now the case where `#1` is '`\Att0:color\relax`'. This is found in the long list because the element accepts the color attribute. Hence the arguments of `\XML@tempb` are the following: The first argument is '`\Att 5:background\relax \setbg {black}`' (in general, it starts with all the unhandled `\do...` commands), the second argument is '`\setcol`', the third is '`blue`', the last is '`\checkcolor \Att 0:color\relax \XML@temp@l {6}`'. The concatenation of arguments 1 and 4 is the long list, with the `\Att...` of color removed, and re-inserted at the end, with `\XML@temp@l` as command and 6 as value. The action is to define `\setcol` to '`red`'. The definition is local to the group started on line 353, ended on line 453. The action associated to `<X:elt>`, `</X:elt>` and descendants can see this value.

After all these `\do...` have been evaluated, our long list reduces to '`\Att 25:background\relax \setbg {black} \checkcolor \Att 0:color\relax \XML@temp@l {6} \relax \relax`'. This is the list constructed by `\XMLelement`, possibly re-ordered, where the command name associated to attributes that have a value is replaced by a dummy name. Note the placement of `\checkcolor` in this list: when it is evaluated, `\setcol` is defined, either to the value of the XML file or the default value. The definition of `\Att` is given here: it sets `\setbg` to black, and `\XML@temp@l` to 6. The only subtlety is that, if the default value is `\inherit` nothing happens. Note: the initial value should always be defined; however, the code checks this, replacing undefined by `\relax`.

```
471 \def\XML@attrib#1\relax#2#3{
472   \ifx\inherit#3\relax% #3 might be empty
473     \ifx#2\@undefined
474       \def#2{\relax}
475     \fi
476   \else
477     \def#2{#3}
478   \fi}
479 %
480 \let\inherit\XML@attrib % just some random name
```

2.11 Processing instructions

We consider here parsing `<?xml?>`. The layout here is awful, for the same reason as `\XML@getname`. We use however a different trick: we use `\csname` and define the delimiters (space, question mark), to evaluate to `\endcsname` (no namespace hacking needed here).

```
\gdef\XML@getpi#1\@{
\fi\fi\fi
\begingroup
\utfeight@protect@chars
\Activespace
\def?{\endcsname?}
\let \endcsname
\let^^M\endcsname
\let^^I\endcsname
\expandafter\XML@getpi@\csname
Q:}
```

Hence, in the case `<?xml something?>`, the `\XML@getpi@` command is called, with as argument the token `\Q:xml`. What we do is close the current group, activate spaces, evaluate the command. If the command is not defined, we put a `\XML@getpi@x` before it. Note that, if a command is constructed by `\csname`, its value is `\relax` instead of undefined; this is a local assignment, after `\endgroup`, the undefined value is restored. Note that the action associated to `<?xml?>` is defined on line 198.

```
\def\XML@getpi@#1{
  \endgroup
  \Activespace
  \ifx#1\@undefined
    \expandafter\XML@getpi@x
  \fi
  #1}
```

In the case where `<?foo something?>` is seen, and the command `\Q:foo` is undefined, we read everything, and call `\XML@dopi` with innocent arguments.

```
\def\XML@getpi@x#1#2?>{
  \XML@dopi{Undefined}{}}
```

In the case `<?xmltex something?>` we use an auxiliary command that grabs the content with the right category codes. This allows TEX commands, with TEX syntax.

```
\expandafter\def \csname Q:xmltex\endcsname{
  \begingroup
  \XML@reset
  \catcode`\>\active
  \XML@xmltexpi}
```

The code of the command is trivial. We call `\XML@dopi`.

```
\gdef\XML@xmltexpi#1?>{
  \endgroup
  \XML@dopi{xmltex}{#1}}
```

The default action is trivial also. Not inlined because of grabbing.

```
\def\XML@dopi#1#2{
  #2}
```

2.12 Declarations

In this paragraph, we consider declarations, things that start with `<!`. Here the `\@` has as purpose to read in `#1` all tokens up to the end of `\XML@lt@markup`. We read the first two characters after the `<!` and decide what to do. The `@` at the end makes reading of arguments easy.

```
\def\XML@getdecl#1\@#2#3{
\fi\fi
  \if-\noexpand#2\XML@comment     %   --
  \else\if N\noexpand#3\XML@entity%   EN TITY
  \else\if L\noexpand#3\XML@dec@e%    EL EMENT
  \else\if A\noexpand#2\XML@dec@a%    AT TLIST
  \else\if D\noexpand#2\XML@doctype%  DO CTYPE
  \else\if C\noexpand#3\XML@cdata%    [C DATA
  \else        \XML@dec@n%            NO TATION
@}
```

Easy part: Element declarations are ignored. In fact, elements can only be defined via an xmt file.

```
\def\XML@dec@e#1@#2>{
  \fi\fi\fi
  \XML@checkend@subset}
```

In the case of `<!ATTLIST...>` declarations, we will do something. We start with closing all conditionals. After that, we read the element name and save it somewhere. Then we parse the list.

```
\def\XML@dec@a#1 #2 {
  \fi\fi\fi\fi
  \protected@xdef\XML@tempa{#2}
  \XML@dec@a@x}
```

The XML production number 52 says that we should have 'AttDef*', optional space and close tag; thus the code fails if no attribute is declared. Let's hope that the list is not empty. Production 53 says that 'AttDef' is space, name, space, 'AttType', space, 'DefaultDecl'. We read the name and store it in `\XML@tempb`. After that we look at the character that follows. It could be an open parenthesis, or something else. The type of the attribute is ignored.

```
\gdef\XML@dec@a@x#1 #2{
  \protected@xdef\XML@tempb{#1}
   \if(\noexpand#2
     \begingroup
     \catcode`\(\active
     \expandafter\XML@dec@a@brack
   \else
      \expandafter\XML@dec@a@type
   \fi}
```

Rule 59 says that the type can be a list enclosed by parentheses.

```
\gdef\XML@dec@a@brack#1){
  \endgroup
  \XML@dec@a@hash}
```

According to rules 54, 55, and 56, the type can be `CDATA`, `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NMTOKEN` or `NMTOKENS`. It could also be `NOTATION` followed by a list. Here we skip over the word, and continue parsing.

```
\def\XML@dec@a@type#1 {
  \XML@dec@a@hash}
```

Rule 60 says that we should have `#REQUIRED`, `#IMPLIED`, or a default value, optionally preceded by `#FIXED`. We consider three cases: If we see a `#`, we read it via `\XML@dec@a@type`; this will call this function. Said otherwise, when we are here, we might have read the `#FIXED`, and are ready for the value, or we might have read `#REQUIRED`, and a `>` sign is OK, as well as another attribute declaration; for this reason, we redefine `\ERROR`: this command is called when the character that follows is neither a single quote nor a double quote.

```
\gdef\XML@dec@a@hash$1{
  \if\noexpand$1#
    \expandafter\XML@dec@a@type
  \else
    \ifx$1>
      \let\ERROR\@undefined
      \expandafter\expandafter\expandafter\XML@checkend@subset
    \else
      \let\ERROR\XML@dec@a@nodef
      \XML@dec@a@def$1
    \fi
  \fi}
```

When we come here, we have finished our 'AttDef' and we are ready for the next one.

```
\gdef\XML@dec@a@nodef#1\fi\fi#2{
  \fi\fi
  \XML@dec@a@x#1}
```

When we come here, we have a default value for the attribute.

```
\def\XML@dec@a@def#1\fi\fi{
 \fi\fi
  \XML@quoted\XML@dec@a@default#1}
```

This code adds `\XML@add@attrib{name}{att}{val}` to a global list, where 'name' is the name of the element, 'att' the name of the attribute and 'val' the default value of the attribute. It continues parsing the declaration.

```
\def\XML@dec@a@default#1#2{
  \ifx\XML@default@attributes\relax
    \let\XML@default@attributes\@empty
  \fi
  \toks@\expandafter{\XML@default@attributes}
  \protected@xdef\XML@default@attributes{
    \the\toks@\noexpand\XML@add@attrib{\XML@tempa}{\XML@tempb}{#1}}
  \XML@dec@a@hash#2}
```

Remember line 414: there was `\XML@default@attributes`. This list was constructed by the code above. It consists of a sequence of `\XML@add@attrib ABC`. What we do here is to evaluate in a context where `\begintag` is the element to be evaluated. If it matches, we call `\XML@attribval`. The effect is as if the user gave B = 'C'; in the case B = 'something' is on the attribute list, this is evaluated first, and the B = 'C' is useless.

```
\def\XML@add@attrib#1#2#3{
  \gdef\XML@tempa{#1}
  \ifx\XML@tempa\begintag
   \def\XML@this@attribute{#2}
    \let\XML@getattrib\relax
```

```
573     \XML@attribval{#3}
574     \let\XML@getattrib\XML@@getattrib
575   \fi}
```

This reads a comment. The code is trivial. An intermediary command is needed for the case where we want to grab something.

```
576 \def\XML@comment#1@#2-->{
577   \fi
578   \Activespace
579   \XML@comment@}
```

This is the intermediary command.

```
580 \def\XML@comment@{\XML@checkend@subset}
```

2.13 Entities

We have to distinguish between `<!ENTITY foo ...>` and `<!ENTITY % foo ...>`. If a percent character is present, this is a parameter entity, and it can be used only in a DTD. Moreover, it is always a parsed entity (no NDATA allowed in the declaration). The command defined here takes as argument some junk, and what follows, the percent sign or a name. Here in the code, we have two versions of `\XML@input`. This command will be explained later. We continue parsing with `\XML@p@ent` or `\XML@ent`.

```
581 \gdef\XML@entity#1 #2 {
582   \fi\fi
583   \ifx%#2
584   \def\XML@input{
585     \ifx\XML@use\XML@SYSTEM\expandafter\@gobble\else
586       \noexpand\inputonce\fi}
587   \expandafter\XML@p@ent
588   \else
589   \def\XML@input{\noexpand\xmlinput}
590   {\utfeight@protect@chars\xdef\XML@ename{&#2}}
591   \expandafter\XML@ent
592    \fi}
```

We have to distinguish between `<!ENTITY % foo "val">`, `<!ENTITY % foo2 SYSTEM "val">`, and `<!ENTITY % foo3 PUBLIC "file" "val">`. Here we put in `\XML@ename` the entity name '`%foo1`' and look at the first character of what follows.

```
593 \gdef\XML@p@ent#1 #2{
594   {\utfeight@protect@chars\xdef\XML@ename{%#1}}
595   \if\noexpand#2P\XML@E@public
596   \else\if\noexpand#2S\XML@E@system
597    \else\XML@E@internal#2}
```

We have to make the same distinctions in the case `<!ENTITY foo1 ...>`. Here we have put the entity name '`&foo1`' in `\XML@ename` and look at the first character of what follows. This looks like above, but `NDATA` is allowed here.

```
598 \def\XML@ent#1{
599   \if\noexpand#1P\XML@E@public
600   \else\if\noexpand#1S\XML@E@system
601    \else\XML@E@internal#1}
```

This handles the case `<!ENTITY % foo1 "val">` or `<!ENTITY foo2 'val'>`. In `\XML@ename` we have '`%foo1`' or '`&foo2`'. We have read the opening quote. What we do on line 607 is to redefine it to be `</`, so that the parser will see `val</>`. We will read the argument via `\xmlgrab`. This command will be explained later; the important point is that it will read everything up to the end of the current element (i.e. up to the `</>`), and call the command associated to the current element, defined on line 608. The effect is to call `\XML@E@internal@x` after the assignment, which is the `\gdef` that defines `\+%foo1` or `\+&foo2` with as body all the grabbed text. The whole difficulty is that the declaration could be something like `<!ENTITY ier "<hi rend='sup'>er</hi>">`, so that the attribute list has to be parsed, but it is too early for namespace processing. For this reason, some commands have to be redefined.

```
602 \gdef\XML@E@internal#1{
603   \fi\fi
604   \begingroup
605   \let\XML@endgroup\endgroup  % use real groups instead of faked ones.
606   \let\XML@begingroup\begingroup
607   \def#1{</}
608   \expandafter\def\csname E\string/:\endcsname{
609     \afterassignment\XML@E@internal@x
610     \expandafter\gdef\csname+\XML@ename\endcsname}
611   \begingroup
612   \let\XML@ns@decl\relax% stop xmlns 'attribute' being recognised
613   \let\XML@this@local\@empty
614   \def\XML@this@prefix{*} % set up special prefix to gobble colon
615   \let\XML@checkknown\relax % disable these
616   \def\XML@ns##1{% hobble namespace code to put all name in local part.
617     \protected@edef\XML@this@local{##1}
618     \def\XML@this@prefix{*}}
619   \xmlgrab}
```

This closes the group started line 611. After that, we execute three tokens after the current group, namely `\XML@trace@warn` (for debug), `\+&foo2` (argument of previous) and `\fihack`. The group ends because of the token at line 812 (the `\XML@endgroup` redefined above).

```
620 \def\XML@E@internal@x{
621    \endgroup
622     \aftergroup\XML@trace@warn
623     \expandafter\aftergroup\csname+\XML@ename\endcsname
624     \aftergroup\fihack
625 }
```

The `\fi` comes from line 813. This piece of code just ignores the conditional. It continues parsing of the element.

```
626 \def\fihack#1\fi{\expandafter\XML@checkend@subset}
```

When we see `<!ENTITY foo PUBLIC 'pub-part' 'system-part'>`, this piece of code finishes reading the `PUBLIC` token, then reads `pub-part`, and calls `\XML@E@pubid`.

```
627 \def\XML@E@public#1 {
628    \fi
629    \XML@quoted\XML@E@pubid}
```

After that, all characters in `pub-part` are converted to category 12, using a classical method, the result is stored in `\XML@E@pubid`, and `system-part` is read.

```
630 \def\XML@E@pubid#1{
631   \def\XML@PUBLIC{#1}
```

```
  \edef\XML@PUBLIC{\catxii\XML@PUBLIC}
  \XML@quoted\XML@E@systemid}
```

The case `<!ENTITY foo SYSTEM 'system-part'>` is similar, but there is no public part.

```
\def\XML@E@system#1 {
   \fi\fi
   \def\XML@PUBLIC{}
   \XML@quoted\XML@E@systemid}
```

Now we run the catalogue. This sets `\XML@use` to either the value associated to the 'pub-part' in a `PUBLIC` item of the catalogue, or the value associated to the 'system-part' in a `SYSTEM` item, or `\XML@SYSTEM` if nothing is found. Assume that this is X; we call `\XML@E@internal@` with `\XML@input{X}` as first argument, the second argument being the unread character, that should be a greater-than sign (but junk is silently ignored). An unparsed entity contains `NDATA Y` for some Y. In this case, the command is called with `{Y}{X}` instead, and `\XML@E@ndata` is used to read Y.

```
\def\XML@E@systemid#1#2{
  \def\XML@SYSTEM{#1}
  \let\XML@use\XML@SYSTEM
  \the\XML@catalogue
  \if\noexpand#2N
   \expandafter\XML@E@ndata
  \else
    \afterfi
    \XML@E@internal@{\XML@input{\XML@use}}#2
  \fi}
```

In the case of `NDATA`, we hack a bit. All characters up to the greater-than sign are read, this gives three lists: the unread part of 'NDATA', the value that follows, optional junk.

```
\def\XML@E@ndata#1 #2>{\XML@ndata@#2 >}
\def\XML@ndata@#1 #2>{
    \XML@E@internal@{{#1}{\XML@use}}>}
```

The command takes two arguments: some action, and everything that remains on the current element. Assume that we consider `<!ENTITY % foo SYSTEM "bar">`. In this case (parameter entity), we define `\+%foo`; its body will be the expansion of `#1`. This is `\XML@input{\XML@use}`, where the argument is what is found in the catalogue, and the command is defined on lines 584 or 589. In the case of `foo`, without percent sign (general entity), the command `\XML@input` is the same as `\xmlinput`. Otherwise, it is `\inputonce` (except: it ignores the argument if not found in the catalogue).

```
\def\XML@E@internal@#1#2>{
   \expandafter\protected@xdef\csname+\XML@ename\endcsname{#1}
  \XML@checkend@subset}}
```

This is done at the end of an `<!ENTITY...>` declaration.

```
\gdef\XML@checkend@subset{
  \Normalspace
  \XML@checkend@subset@}
```

This is a bit strange: why does the command take these four arguments? The definition of the percent character is to make it a normal Unicode character (end of local DTD).

```
\gdef\XML@checkend@subset@#1#2#3#4{
  \ifx]#1
  \let\XML@w@\@empty
  \gdef%{\utfeightay%}
```

```
  \let\XML@checkend@subset\relax
  \expandafter\XML@loaddoctype
  \fi
  #1#2#3#4}
```

In the case where the `<!DOCTYPE>` element specifies a DTD, we load the file.

```
\def\XML@loaddoctype#1#2{
  \Activespace
  \ifx\XML@D@dtd\relax\else
    \inputonce\XML@D@dtd
  \fi}
```

2.14 Interpreting the Doctype element

Consider now that case of `<!DOCTYPE TEI.2 SYSTEM "teilite.dtd">`. This piece of code finishes reading the `DOCTYPE` name, then the name of the document element, it puts it in `\documentelement`. It then checks if what follows is `PUBLIC`, `SYSTEM`, an internal subset, or the end of the element. The command closes all `\fi` that are open. There is an `@` here that makes it easy to skip over the conditionals defined here.

```
\gdef\XML@doctype#1 #2 #3{
 \fi\fi\fi\fi\fi
  \def\documentelement{#2}
  \let\XML@D@dtd\relax
  \if\noexpand#3P\XML@D@public
  \else\if\noexpand#3S\XML@D@system
  \else\ifx#3[\XML@D@internal
  \else%must be > the end
    \XML@D@empty
  @}
```

If nothing is given, we have nothing to do.

```
\gdef\XML@D@empty @{
   \fi\fi\fi}
```

In the case of `PUBLIC`, we parse the public value, and do something with it.

```
\gdef\XML@D@public#1 {
   \fi
   \XML@quoted\XML@pubid}
```

In the case of `<!DOCTYPE foo PUBLIC "aaa" "bbb">`, we put the `aaa` part in `\XML@PUBLIC`, change all category codes to 12, and read the '`bbb`' part.

```
\gdef\XML@pubid#1{
  \def\XML@PUBLIC{#1}
  \edef\XML@PUBLIC{\catxii\XML@PUBLIC}
  \XML@quoted\XML@systemid}
```

In the case of `<!DOCTYPE foo SYSTEM "bbb">`, it is as above, but the public part is empty.

```
\gdef\XML@D@system#1 {
   \fi\fi
   \def\XML@PUBLIC{}
   \XML@quoted\XML@systemid}
```

We put the `bbb` part in `\XML@SYSTEM`, change all category codes to 12, run the catalogue, and put the result in `\XML@D@dtd` for later use by `\XML@loaddoctype`.

```
\gdef\XML@systemid#1{
  \protected@edef\XML@SYSTEM{#1}
  \edef\XML@SYSTEM{\catxii\XML@SYSTEM}
  \let\XML@use\@empty
  \the\XML@catalogue
  \let\XML@D@dtd\XML@use
  \XML@D@internal@}
```

When we have no `PUBLIC` and no `SYSTEM` part, but only a local DTD, we call this: it pops the conditional stack, and pushes back the open bracket. The action is the same as if a `PUBLIC` or `SYSTEM` part had be given.

```
\gdef\XML@D@internal#1@{
  \fi\fi\fi
  \XML@D@internal@[}
```

We parse the internal DTD in the same fashion as everything else. However `%foo;` evaluates to something, so that the percent sign must be activated. The vertical bar is used for comments.

```
\gdef\XML@D@internal@#1{
  \ifx[#1
    \let%\XML@pcent
    \edef\XML@w@{ \XML@w@}
     \expandafter\XML@checkend@subset
  \else
      | it had better be the closing >
   \fi}
```

Inside a local DTD, the parameter entity '`%foo;`' evaluates to `\+%foo`.

```
\gdef\XML@pcent#1;{
  \csname+%#1\endcsname
  \XML@checkend@subset}
```

When you say `&#foo;` the `\XML@charref` command is called to parse the entity; the result is in `\XML@tempa`. It will in general be evaluated right now. On the other hand '`&foo;`' evaluates to `\+&foo`, there is no intermediate command.

```
\let&\XML@amp@markup
\gdef\XML@amp@markup$1$2;{
  \ifx#$1\@empty
   \XML@charref$2;
   \XML@tempa
  \else
   \begingroup\utfeight@protect@chars
   \expandafter\aftergroup
   \csname+\string&$1$2\expandafter\endcsname
   \endgroup
  \fi}
```

In the case of `<![CDATA xxx ]]>`, this reads up to the first space.

```
\gdef\XML@cdata #1[{
 \fi\fi\fi\fi\fi\fi
  \Activespace
  \XML@cdata@a}
```

And this reads everything up to the special end marker `]]>`. Less-than sign and ampersand are not active.

```
729 \gdef\XML@cdata@a#1]]>{
730   \begingroup
731   \edef<{\noexpand\utfeightaz\string<}
732   \edef&{\noexpand\utfeightaz\string&}
733   \XML@docdata{#1}}
```

The action here is trivial. We need an intermediary command, in the case of grab.

```
734 \def\XML@docdata#1{#1\endgroup}
```

The only thing done here is to skip over everything, until the end.

```
735 \def\XML@dec@n#1N #2 #3 {
736  \fi\fi\fi\fi\fi\fi
737   \XML@quoted\XML@notation
738   }
739
740 \def\XML@notation#1#2{
741   \ifx>#2
742    \expandafter\XML@checkend@subset
743   \else
744     \afterfi
745     \XML@quoted\XML@notation#2
746   \fi}
```

2.15 Grabbing content

The normal behavior of `<foo>text</foo>` is like `\begin{foo}text\end{foo}`. This method is the most efficient concerning memory space. In some cases, we prefer the equivalent of `\def \arg {text}`, `\foo {arg}`. The interesting point is that, assuming that `<foo>` takes two children, and that we do not care about cases with incorrect syntax, we can manage everything so that the user function sees these two children, for instance in the form `\split\arg\first\second` followed by `\foo\first\second`. In fact, instead of calling a single command, we call two commands, as `\foofirst\first`, `\foosecond`.

The following piece of code is the definition of the `<msup>` element in the MathML namespace. We shall explain the syntax of `\XMLelement` later. Line 10002 says: we do not care about attributes, line 10003 says that we want to grab the content of the element. Line 10004 says: there are two children, and we want to apply some commands to them.

```
10000   \XMLelement{m:msup}
10001     {}
10002     {\xmlgrab}
10003     {\xmltextwochildren\@firstofone\sp#1}
```

We shall define `\xmlgrab` below. It will read the content of the element, and use `\@empty` as element separator (see code line 819); remember that `\@empty` expands to nothing, hence is harmless. This marker allows easy splitting. In the case of the example of the start of the chapter, the tokens are

```
\xmltextwochildren\@firstofone\sp
+<3:mi^^I>L</3:mi>\@empty  <3:mn^^I>2</3:mn>\@empty  +
```

The second line is printed by TEX, when we ask for the value of `+#1+`, we have inserted the plus signs, this being the easiest way to see that each `\@empty` is followed by a space (they are in the input file), and for each opening tag, a tabulation between the tag name and the attribute list (which is empty in this case). With the definition below, the arguments of `\xmltextwochildren` will be

```
#1=\@firstofone
#2=\sp
#3=<3:mi>L</3:mi>
#4=<3:mn>2</3:mn>
```

(we did not show the tabulations, nor the spaces). Tabulations are read again by the parser when looking for attributes, spaces are ignored, as usual, in math mode. The effect of the command is to apply the first argument to the third, the second to the fourth. If the arguments are, say, `\A`, `\B`, `CC` and `DD`, the result is `\A{CC}\B{DD}`. In our case, we want `CC^{DD}`, so that `\A` is just a command that removes useless braces, and `\B` is the TeX primitive for superscripts.

```
\def\xmltextwochildren#1#2#3\@empty#4\@empty{
  #1{#3}#2{#4}}
\def\xmltexthreechildren#1#2#3#4\@empty#5\@empty#6\@empty{
  #1{#4}#2{#5}#3{#6}}
```

This is a small function that returns everything before the `\@empty`. You have to use `\@` to mark the end of the child list (only first child is used here).

```
\def\xmltexfirstchild#1\@empty#2\@{
  #1}
```

If you say `\xmltexforall\cmd{list}`, where the second argument is a list of tokens with `\@empty` between tokens, this applies `\cmd` to each item. Moreover, the quantity `\xml@name` contains the name of the element. The end of the loop relies on the fact that no element name starts with a space.

```
\def\xmltexforall#1#2{
  \xmltexf@rall#1#2< >\@empty}

\def\xmltexf@rall#1#2<#3 #4>#5\@empty{
  \ifx\relax#3\relax
  \else
  \def\xml@name{#3}#1{<#3 #4>#5}
  \expandafter\xmltexf@rall\expandafter#1
  \fi}
```

The action associated to `<foo>` consists in two parts: first all attributes are scanned, and some commands are instantiated (see section 2.10), and then the start code is executed (in the example, line 10002). When we see `</foo>`, we execute the end code (line 10003, in the example). In the special case where the initial action is `\xmlgrab`, the command gets an argument. This argument is computed on line 811 as the value of the token list `\XMLgrabtoks`. Thus, the `\xmlgrab` command must read all tokens, up to the end tag; it must handle namespaces properly (as the example shows, all namespaces, even the default ones, are replaced by integers).

The idea is to redefine temporarily all commands `\XML@do...`, for the case `<foo>`, `<?foo>`, `<!foo>`, and `</foo>`, and ask them to put the result in the list. We store in `\XML@next@level` the value of `\XML@w@` at the next level. It is thus possible to check, for a given element, if it is a child (and not merely a descendant) of the current element, so that we know where to insert the `\@empty` markers. The main routine here is `\grab@`.

```
\def\xmlgrab{
  \begingroup
  \global\XMLgrabtoks{}
  \let\XML@this@level\XML@w@
  \edef\XML@next@level{ \XML@w@}
  \let\XML@doelement\XML@grabelement
  \let\XML@doend\XML@grabend
  \let\XML@docdata\XML@grabcdata
```

```
  \let\XML@comment@\XML@grabcomment@
  \let\XML@dopi\XML@grabpi
  \XMLgrab@}
```

This uses the same magic as `\XML@getname`. The idea is to read everything until the next less-than sign, putting all tokens in the command `\XML@tempa`.

```
\def\XMLgrab@{
  \utfeight@protect@internal
  \def<{\iffalse{\fi}\XMLgrab@@}
  \xdef\XML@tempa{\iffalse}\fi}
```

When `\XMLgrab@` has read everything between tags, it puts the grabbed tokens in the token register `\XMLgrabtoks`, and then evaluates the less-than sign.

```
\def\XMLgrab@@{
  \global\XMLgrabtoks\expandafter{\the\expandafter\XMLgrabtoks\XML@tempa}
  \XML@lt@markup}
```

This command is called when we grab the content of an element. Assume that we have seen `<mi>L</mi>`. When we are here, we have seen the first less-than sign. And we know that `\XML@this@element` is '`3:mi`'. We add to the token list `<3:mi atts>`. There is a tabulation after the element name, this is obtained by uppercasing the tilde. Attributes are added by a call to `\the\XML@attribute@toks`, with a temporary redefinition of `\XML@doattribute`, and the default namespace is neutralized.

```
\uppercase{
\gdef\XML@grabelement{
   \Activespace
  \global\XMLgrabtoks\expandafter{
    \the\expandafter\XMLgrabtoks
      \expandafter<\XML@this@element~}
   \begingroup
   \let\XML@doattribute\XML@grabattribute
   \def\XMLNS@{0}
   \expandafter\let\csname XMLNS@0\endcsname\XMLNS@
   \the\XML@attribute@toks
   \endgroup
   \Activespace
  \global\XMLgrabtoks\expandafter{
    \the\XMLgrabtoks
    >}
   \XMLgrab@}
}
```

Assume that foo:bar = 'gee' is in the attribute list of the current element, and assume that `foo` has namespace number 4. We add `4:bar="gee"` and a space to the token list.

```
\gdef\XML@grabattribute#1#2#3{
  \protected@xdef\XML@tempa{\jg@namespace{#1}:#2}
  \global\XMLgrabtoks\expandafter{
    \the\expandafter\XMLgrabtoks
    \XML@tempa="#3" }}
```

Let's assuming that we are grabbing something and we see `</foo>`. There are two cases to consider. If this element is the one we are looking for, we close the group open by `\xmlgrab`, and we execute the command `\E/:3:msup` (there are some hacks here; the `\uppercase` command replaces dot and star by slash and colon with the right category code). We pass the grabbed token list as argument. This is achieved by putting an `\expandafter` just before the `\endcsname`, it will

expand `\the`, i.e., replace `\XMLgrabtoks` by its value (since we want braces around this token list, another `\expandafter` is needed). After execution of the command, we have to close our XML group and hack with category codes. On the other hand, in the case where the element does not end grabbing, we add `</4:foo>` to the end of the `\XMLgrabtoks` token list. If this is a direct child, we add also a `\@empty` marker. We continue grabbing via a call to `\XMLgrab@`.

```
\uppercase{
\gdef\XML@grabend{
  \ifx\XML@this@level\XML@w@
    \endgroup
    \csname
      E.*\jg@this@namespace
        *\XML@this@local
    \expandafter\endcsname\expandafter{
      \the\XMLgrabtoks}
    \XML@endgroup
    \ifnum\catcode`\^^M=10  \Activespace \fi
  \else
    \xdef\XML@tempa{\noexpand<\noexpand/
      \jg@this@namespace\noexpand: %%%% \expandafter omitted [jg]
            \XML@this@local
    \noexpand>
    \ifx\XML@next@level\XML@w@\noexpand\@empty\fi}
    \global\XMLgrabtoks\expandafter{
      \the\expandafter\XMLgrabtoks
      \XML@tempa}
    \XML@endgroup
  \expandafter
    \XMLgrab@
  \fi}}
```

When we want to grab something and see `<?PI etc?>`, we add all these tokens to our list.

```
\gdef\XML@grabpi#1#2{
  \global\XMLgrabtoks\expandafter{
  \the\XMLgrabtoks<?#1^^I#2?>}
  \XMLgrab@}
```

If you say `\NDATAEntity\att\A\B`, if the expansion of `\att` is something like `foo`, and `\&+foo` expands to `\bar` and `\gee`, this piece of code applies `\A` to `\bar` and `\B` to `\gee`[9].

```
\gdef\NDATAEntity#1{
  \expandafter\expandafter\expandafter
  \XML@ndataentity\csname+&#1\endcsname}

\gdef\XML@ndataentity#1#2#3#4{
  #3{#1}#4{#2}}
```

Grabbing `CDATA` is easy: what we do is re-insert the content of the element, and continue grabbing. Ampersands and less-than signs are replaced by the equivalent of `&` and `<`, said otherwise, inactive characters.

```
\def\XML@grabcdata#1{
  \utfeight@protect@internal
  \edef<{\noexpand\utfeightaz\string<}
  \edef&{\noexpand\utfeightaz\string&}
```

[9] This is unclear to me

```
841   \xdef\XML@tempa{#1}
842   \endgroup
843    \expandafter\XMLgrab@\XML@tempa}
```

When we grab a comment, the only thing we need to do is continue grabbing.

```
844 \def\XML@grabcomment@{
845   \XMLgrab@}
```

2.16 Defining actions

This is for use in a .xmt file, the file that defines actions for each element. After `\XMLentity{foo}{bar}`, the XML entity `&foo;` evaluates to `bar`.

```
846 \gdef\XMLentity#1#2{
847   \expandafter\gdef\csname+&#1\endcsname{#2}}
```

These are the predefined entities:

```
848 \XMLentity{amp}{\utfeightaz&}
849 \XMLentity{quot}{\utfeightax"}
850 \XMLentity{apos}{\utfeightax'}
851 \XMLentity{lt}{\utfeightaz<}
852 \XMLentity{gt}{\utfeightax>}
```

The `\XMLelement` command appears in a .xmt file. It takes four arguments: the first one is the name of an element. We have seen an example above. Assume that the name is `m:msup`. Assume that the 'm' prefix stands for MathML and this corresponds to the number 3. We define two commands `\E:3:msup` and `\E/:3:msup`. The body of these commands is argument `#3` and `#4`. In the case where `#3` is `\xmlgrab` then the `\E/:3:msup` command takes an argument (see previous section). Argument `#2` explains what to do with attributes. It should contain a sequence of `\XMLattribute` commands. Note: all attributes declared for the current namespace (found in `\A:3`) are added to the list. A call to evaluation of the attribute list is inserted in the body of `\E:3:msup` before the user code. On line 860, 861, the purpose of all these `\expandafter` is to expand the `\the`, so as to insert the token list (and not a reference to it) in the body of the definition.

```
853 \long\def\XMLelement#1#2#3#4{
854   \XML@ns{#1}
855   \xdef\XML@tempc{:\jg@this@namespace
856        :\XML@this@local}
857   \toks@\expandafter{\csname A:\jg@this@namespace
858     \endcsname}
859   #2
860   \expandafter\gdef\csname E\XML@tempc\expandafter\endcsname
861   \expandafter{\expandafter\XML@setattributes\expandafter{\the\toks@}#3}
862   \gdef\XML@tempa{#3}
863   \ifx\XML@tempa\XML@xmlgrab
864     \expandafter\gdef\csname E\string/\XML@tempc\endcsname##1
865     {#4}
866   \else
867     \expandafter\gdef\csname E\string/\XML@tempc\endcsname
868     {#4}
869   \fi}
870 \def\XML@xmlgrab{\xmlgrab}
```

When you say `\XMLattribute {form} {\mycmd} {inline}` this puts in `\XML@tempa` the quantity `\XML@attrib 0:form\relax \mycmd {inline}`. If '`form`' is replaced by '`m:form`' and the namespace value of '`m`' is 3, then '`3:form`' will be used instead of '`0:form`'. This works by redefining locally the default namespace to be the empty namespace. The value of `\XML@tempa` will be added to the end of `\toks@`, the token list used in `\XMLelement` or `\XMLnamespaceattribute`. For some strange reason the second argument is put in `\XML@tempa`, but not the last one (this implies that the third argument is not expanded; the single token of the second argument is not expanded, because of the `\noexpand`; if the second argument has more than one token, you lose).

```
\long\def\XMLattribute#1#2#3{
  {\def\XMLNS@{0}
  \XML@ns{#1}
  \xdef\XML@tempa{\noexpand\XML@attrib
      \jg@this@namespace
        :\XML@this@local\relax\noexpand#2}}
  \toks@\expandafter{\the\expandafter\toks@\XML@tempa{#3}}}
```

Like above, with a little hack. Assume that the attribute has to be stored in `\foo`. Then `\utfeight@chardef\foo` is executed (some time after a value has been stored).

```
\long\def\XMLattributeX#1#2#3{
  {\def\XMLNS@{0}
  \XML@ns{#1}
  \xdef\XML@tempa{\noexpand\XML@attrib
      \jg@this@namespace
        :\XML@this@local\relax\noexpand#2}}
  \toks@\expandafter{\the\expandafter\toks@\XML@tempa{#3}\utfeight@chardef#2}}
```

This is the action associated to the special setting used above. The command is fully expanded, in a group where this is harmless, globally put in a temporary, and then, outside the group, the temporary is put again in the command. There seems to be a problem here: what if the argument contains ampersands and less-than signs?

```
\def\utfeight@chardef#1{
\begingroup
\utfeight@protect@chars
\xdef\x@temp{#1}
\endgroup
\let#1\x@temp}
```

In case `\XMLnamespaceattribute{foo}{bar}{gee}{etc}`, if the namespace number of `foo` is 4, this code will define the action associated to the attribute defined by bar, gee, etc, and put it in `\A:4`. This command should be used at toplevel, not inside `\XMLelement`.

```
\long\def\XMLnamespaceattribute#1#2#3#4{
   \toks@\expandafter\expandafter\expandafter{\csname A:%
    \jg@namespace{#1}\endcsname}
  \XMLattribute{#2}{#3}{#4}
  \expandafter\xdef\csname A:\jg@namespace{#1}\endcsname{\the\toks@}}
```

Idem, expanded.

```
\long\def\XMLnamespaceattributeX#1#2#3#4{
   \toks@\expandafter\expandafter\expandafter{\csname A:%
    \jg@namespace{#1}\endcsname}
  \XMLattributeX{#2}{#3}{#4}
  \expandafter\xdef\csname A:\jg@namespace{#1}\endcsname{\the\toks@}}
```

If you say `\XMLname{foo:bar}{\gee}` this puts in `\gee` something like `4:bar`.

```
\long\gdef\XMLname#1#2{{
  \XML@ns{#1}
  \xdef#2{\jg@this@namespace\noexpand:\XML@this@local}}}
```

If you say `\XMLstring\foo<>bar</>` this piece of code reads the `<>`, then calls `\xmlgrab`, which reads everything until the `</>`, and executes the code associated to the end tag: this defines `\foo`, and closes the group.

```
\gdef\XMLstring#1#2<>{
  \begingroup
  \let\XML@endgroup\endgroup
  \let\XML@this@local\@empty
  \let\XML@this@prefix\@empty
  \expandafter\def\csname E/:\XMLNS@:\endcsname{\gdef#1}
  \XML@catcodes
  \xmlgrab}
```

Same code, expanded.

```
\gdef\XMLstringX#1#2<>{
  \begingroup
  \let\XML@endgroup\endgroup
  \let\XML@this@local\@empty
  \let\XML@this@prefix\@empty
  \expandafter\def\csname E/:\XMLNS@:\endcsname{\xdef#1}
  \XML@catcodes
  \utfeight@protect@chars
  \xmlgrab}
```

2.17 Other commands

Public version of `\XML@setenc`.

```
\let\DeclareNamespace\XML@ns@uri  %  version for xmt files
\def\FileEncoding#1{\XML@setenc{#1}\relax}
\newtoks\XML@attribute@toks
\newcount\XML@ns@count
\newtoks\XMLgrabtoks
```

The next function reads an XML file. The idea is to restore the current encoding. We have not shown the source of `\XML@xmlinput`.

```
\def\xmlinput#1{
 \IfFileExists{#1}
  {\expandafter\XML@xmlinput\expandafter
    \XML@setenc\expandafter{\XML@thisencoding}\relax
  }{\XML@warn{No file: #1}}}

\def\XML@xmlinput{...}
```

This reads a TEX file only once.

```
\def\inputonce#1{
  \expandafter\ifx\csname xmt:#1\endcsname\relax
  \global\expandafter\let\csname xmt:#1\endcsname\@ne
  \begingroup
  \XML@reset
  \def\XMLNS@{0}
```

```
939   \input{#1}
940   \endgroup
941   \fi}
```

In case you want some characters to be active.

```
942 \gdef\ActivateASCII#1{
943   \uppercase{\count@"0\if x\noexpand#1\relax\else\count@#1\fi\relax}
944   \toks@\expandafter{\nfss@catcodes}
945        \xdef\nfss@catcodes{
946        \catcode\the\count@=\the\catcode\the\count@\relax\the\toks@}
947   \toks@\expandafter{\XML@catcodes}
948      \xdef\XML@catcodes{
949        \catcode\the\count@\active\the\toks@}
950   \expandafter\ifx\csname8:"\endcsname\relax
951     \expandafter\gdef\csname8:"\endcsname{"}
952   \fi}
953 \ActivateASCII{94}% ^ for tex ^^ notation in aux files
954 \ActivateASCII{x5C}% \
955 \ActivateASCII{x5F}%  underscore [jg]
956 \ActivateASCII{123}% {
957 \ActivateASCII{125}%  close brace [jg]
```

We redefine `\obeyspaces` and `\obeylines`. The idea is to redefine the action associated to space and newline character.

```
958 \expandafter\def\expandafter\obeylines\expandafter{
959 \expandafter\def\csname 8:\string^^M\endcsname{\leavevmode\hfil \break\null}}
960
961 \expandafter\def\expandafter\obeyspaces\expandafter{
962 \expandafter\def\csname 8: \endcsname{\nobreakspace}}
```

This line is a bit strange:

```
963 \expandafter\def\csname 8:\string^^I\expandafter\endcsname
964       \expandafter{\csname 8: \endcsname}
```

The end of the xmltex.tex file is like this (we simplified a bit the code, by assuming that we do not want to dump a format, and that `\xmlfile` is defined). Essentially, we reset the end-of-line character to its normal meaning, we set the category codes, and we load the XML file.

```
965 \def\XML@tempa{\catcode`\-12\relax\input\xmlfile\relax}
966 \endlinechar`\^^M \expandafter\XML@catcodes\XML@tempa
```

2.18 Example

Assume that we have a file thesis.tex containing the following lines.

```
\def\xmlfile{these.xml}
\def\LastDeclaredEncoding{T1}
\input{xmltex.tex}
\end{document}
```

When TeX processes this file, it loads xmltex.tex, the file described in this chapter, because of line 1003. This defines a lot of commands; however the last line (line 966) contains some action, consisting essentially into setting some variables (end-of-line character, category codes) to values useful for typesetting. There are two hooks, not shown here. First, if the file xmltex.cfg is found, it will be loaded. The default file contains some Unicode character definitions, and the catalogue

shown earlier. Second, if thesis.cfg is found, it will be loaded. After that the XML file is loaded, this is defined on line 1001. Let's assume that the root element of the XML file is `<fo:root>`, and that the name space associated to 'fo' is declared in the catalogue, and loads fotex.xmt. This file is described in Chapter 4, and the action associated to `</fo:root>` is `\end{document}`, so that line 1004 is not really needed. Line 1002 is required in some cases 9but it is not clear which ones).

Chapter 3

Interpreting MathML and related stuff in TeX

In the previous chapter, we have seen that TeX is able to read and evaluate an XML file. The Raweb is not typeset directly: there is a stylesheet that converts it into XSL/Format, see Chapter 7. The purpose of the next chapter is to explain how XSL/Format has to be interpreted by TeX; in this chapter we explain everything else, the big part being MathML, described in [2].

We describe here the file raweb-cfg.sty. It starts with the following lines

```
\makeatletter
\immediate\write20{Loading mathml support and raweb extensions
$ Revision: 2.14 $}
```

It is not a real style file, so that the category code of the '@' character is not known, and we do not want to use the `\ProvidesPackage` command for the identification. The revision number given above is computed by RCS, it is likely that the Tralics distribution comes with another version of this file.

This file was obtained by some additions to a file named mathml2.xmt, and it starts with the following copyright notice.

```
% patch to xmltex.tex plus mathml2.xmt plus other stuff
%% Copyright 2000 David Carlisle, for the original mathml part
%% Copyright 2004, 2006 Jos\'e Grimm

%% This file is distributed under the LaTeX Project Public License
%% (LPPL) as found at http://www.latex-project.org/lppl.txt
%% Either version 1.0, or at your option, any later version.
```

We have seen in the previous chapter that, if xmltex.tex is loaded and LaTeX reads an XML file, then the first occurence of an element in a name space 'xx' provokes loading of file yy.xmt, assuming that 'xx' and 'yy' are associated in the catalogue. In our case, the root element is in the 'fo' namespace, some file is loaded, and the interpretation of the root element provokes loading of all required packages, as well as `\begin{document}`. After that, it is not possible to load other packages, so that the yy.xmt file cannot be a real LaTeX package.

For some unknown reason, in some cases, this autoloading mechanism did not work with the file mathml2.xmt, so that we decided to load it systematically, using a hook. At the end of the previous chapter we have explained that, if your file is called foo.tex, then foo.cfg is loaded. Thus this file is loaded if it has foo.cfg as alternate name (in Chapter 8, we give an example where a Perl

script makes a symbolic link from foo.cfg to raweb-cfg.sty). As a consequence, the file is loaded before anything else (in particular before the class file), but we can use the begin-document hook.

3.1 Local patches to xmltex.tex

Redefinition of `\utfeight@protect@chars`. Compare this with the definition on page 12, line 123. The idea is that using `\let` rather than `\def` should be more efficient. Some tests show that the speedup factor is nearly 10 percent.

```
\def\utfeight@protect@chars{%
  \let\utfeightax\string
  \let\utfeightay\string
  \let\utfeightaz\string
  \let\utfeightb\utfeightb@jg@
  \let\utfeightc\utfeightc@jg@
  \let\utfeightd\utfeightd@jg@}
```

Three auxiliary definitions used above.

```
\def\utfeightb@jg@#1#2{#1\string#2}
\def\utfeightc@jg@#1#2#3{#1\string#2\string#3}
\def\utfeightd@jg@#1#2#3#4{#1\string#2\string#3\string#4}
```

We redefine this also (original definition on page 11, line 96).

```
\begingroup
\catcode`<\active
\catcode`&\active
\gdef\utfeight@protect@internal{%
  \let\utfeightax\noexpand
  \let\utfeightay\noexpand
  \let\utfeightaz\utfeightaz@jg@int
  \let<\relax\let&\relax
  \let\utfeightb\utfeightb@jg@int
  \let\utfeightc\utfeightc@jg@int
  \let\utfeightd\utfeightd@jg@int}
```

These are the auxiliary commands.

```
\def\utfeightaz@jg@int{\noexpand\utfeightaz\noexpand}
\def\utfeightb@jg@int#1#2{\noexpand\utfeightb#1\string#2}
\def\utfeightc@jg@int#1#2#3{\noexpand\utfeightc#1\string#2\string#3}
\def\utfeightd@jg@int#1#2#3#4{\noexpand\utfeightd#1\string#2\string#3\string#4}
```

This is redefined (original on page 39, line 885); we locally change the meaning of the ampersand character, using the command `\XML@amp@markup` shown below.

```
\gdef\utfeight@chardef#1{%
  \begingroup
  \utfeight@protect@chars
  \let&\XML@amp@markup@jg      % <-  modified
  \xdef\x@temp{#1}%
  \endgroup
  \let#1\x@temp}
\endgroup
```

The fotex.xmt file contains a declaration of the form
'`\XMLnamespaceattributeX {fo} {external-destination} {FOexternaldestination} {}`'

The 'X' after the command name means that the argument is evaluated in an `\xdef`. In the case where we want a `&` in an URL, this does not work. Thus we redefine the meaning of `&`: in the case of `&` the expansion is `&`, otherwise the standard behavior is used.

```
\def\XML@amp@markup@jg#1;{%
  \XML@amp@markup@jgw#1;amp;\@nil}
\def\XML@amp@markup@jgw#1amp;#2\@nil{%
  \ifx b#2b\XML@amp@markup#1;\else\string&\fi}
```

3.2 Support for MathML

We pretend here that mathml2.xmt has been loaded (we do not want the original file to be loaded later).

```
\global\expandafter\let\csname xmt:mathml2.xmt\endcsname\@ne
```

We declare the namespaces.

```
\DeclareNamespace{m}{http://www.w3.org/1998/Math/MathML}
\DeclareNamespace{fotex}{http://www.tug.org/fotex}
\DeclareNamespace{fo}{http://www.w3.org/1999/XSL/Format}
```

Let's start with simple things: `<m:mrow>`*text*`</m:mrow>` translates to `\bgroup` *text* `\egroup`. This differs from the original coding, allowing constructions like `${x^2}^3$`.

```
\XMLelement{m:mrow}
  {}
  {\bgroup}
  {\egroup}
```

Processing of `<m:msub>` *base subscript* `</m:msub>`. The translation is *base*`_{`*subscript*`}`. The MathML standard says that there is a subscriptshift attribute that specifies the minimum amount to shift the baseline of subscript down. This is not implemented.

```
\XMLelement{m:msub}
  {}
  {\xmlgrab}
  {\xmltextwochildren\@firstofone\sb#1}
```

Processing of `<m:msup>` *base supscript* `</m:msup>`. The translation is *base*`^{`*supscript*`}`. The attribute superscriptshift defined by the standard is ignored.

```
\XMLelement{m:msup}
  {}
  {\xmlgrab}
  {\xmltextwochildren\@firstofone\sp#1}
```

Processing of `<m:msubsup>` *base subscript superscript* `</m:msubsup>`: the translation is *base*`_{`*subscript*`}^{`*supercript*`}`. The attributes subscriptshift and superscriptshift defined by the standard are ignored.

```
\XMLelement{m:msubsup}
  {}
  {\xmlgrab}
  {\xmltexthreechildren\@firstofone\sb\sp#1}
```

Processing of `<m:mstyle` *atts*`>` *text* `</m:mstyle>`. The translation is *text*. However, the attributes of the element influence typesetting of it. The attributes veryverythinmathspace, verythinmathspace, thinmathspace, mediummathspace, thickmathspace, verythickmathspace, and veryverythickmathspace can be modified (default value k/18em, for k=1 to 7). TeX provides only three values: thin, medium and thick. The attribute scriptsizemultiplier gives the quotient of the

font size at level N and $N+1$ and scriptminsize indicates the minimum size (TEX has three levels of math fonts: text, script, and scriptscript; there is no such limit in MathML, however there is a limit on the size). The default value of the multiplier is 0.71. If the main font is a 10pt, then the script size will be 7pt, and scriptscript size will be 5pt (these are the default values or TEX; however the default scriptminsize is 8pt). We ignore the background attribute.

In this piece of code, we consider the value of the displaystyle attribute. If it is true, we evaluate `\displaystyle`. If it is neither true nor false, nothing happens. If it is false, we look at the scriptlevel attribute. We execute `\textstyle`, or `\scriptstyle`, or `\scriptscriptstyle`, if it is 0, 1 or 2. If the attribute is not provided, we use 0 as default value; if the value is out of range, we use 2.

```
\XMLelement{m:mstyle}
 {\XMLattribute{displaystyle}{\XML@att@displaystyle}{foo} %
   \XMLattribute{scriptlevel}{\XML@scriptlevel}{0}%
 }
 {\xmlgrab}
 {{\ifx\XML@att@displaystyle\att@true\displaystyle
   \else\ifx\XML@att@displaystyle\att@false
    \ifx\XML@scriptlevel\att@dzero\textstyle
     \else\ifx\XML@scriptlevel\att@done\scriptstyle
      \else \scriptscriptstyle\fi\fi
   \fi %do nothing if neither true nor false
   \fi#1}}
```

Processing of `<m:mroot>` *base index* `</m:mroot>`. We call `\mathmlroot` with *base* and *index* as arguments (note that the empty group `{}` disappears when this calls the intermediate command). The MathML norm says that the scriptlevel of *index* should be increased by two (thus, it will be as small as the TEX version).

```
\XMLelement{m:mroot}
  {}
  {\xmlgrab}
  {\xmltextwochildren\mathmlroot{}#1}

\def\mathmlroot#1#2{\root#2\of{#1}}
```

Processing of `<m:msqrt>` *base* `</m:msqrt>`. We just call `\sqrt`.

```
\XMLelement{m:msqrt}
  {}
  {\xmlgrab}
  {\sqrt{#1}}
```

Processing of `<m:mtext>...</m:mtext>`. That's easy, we use `\text`.

```
\XMLelement{m:mtext}
  {}
  {\xmlgrab}
  {\text{#1}}
```

Processing of `<m:ms>...</m:ms>`. Not yet implemented. The MathML doc says: The `<m:ms>` element is used to represent "string literals" in expressions meant to be interpreted by computer algebra systems.

```
% \XMLelement{m:ms} {} {"} {"}
```

Processing of `<m:mn>...</m:mn>`. The MathML doc says: Unlike `<mi>`, `<mn>` elements are (typically) rendered in an unslanted font by default, regardless of their content.

```
92 \XMLelement{m:mn}
93   {}
94   {\xmlgrab}
95   {\mathrm{#1}}
```

Processing of `<m:mi mathvariant=xx>...</m:mi>`. The MathML doc says that the fontstyle attribute is deprecated, and the mathvariant attribute can take one of the following values: normal, bold, italic, bold-italic, double-struck, bold-fraktur, script, bold-script, fraktur, sans-serif, bold-sans-serif, sans-serif-italic, sans-serif-bold-italic, monospace.

```
96 \XMLelement{m:mi}
97   {\XMLattribute{mathvariant}{\XML@mathmlvariant}{normal}}
98   {\xmlgrab}
99   {\mi@test#1\relax}
```

As you can see, not all variants are implemented here. On the other hand, the test done on line 107 is false in the case where the content of the element is a letter plus some space. (The MathML norm does not say how to get a single letter using an upright variant).

```
100 \gdef\mi@test#1#2\relax{
101  \ifx\XML@mathmlvariant\att@mathml@bold
102   \mathbf{#1#2}\else
103  \ifx\XML@mathmlvariant\att@mathml@sansserif
104   \mathsf{#1#2}\else
105  \ifx\XML@mathmlvariant\att@mathml@tt
106   \texttt{#1#2}\else
107   \ifx\mi@test#2\mi@test
108    \expandafter#1
109   \else
110    \mathrm{#1#2}
111   \fi\fi\fi\fi}
```

These are used in the code above.

```
112 \XMLstring\att@mathml@bold<>bold</>
113 \XMLstring\att@mathml@sansserif<>sans-serif</>
114 \XMLstring\att@mathml@tt<>monospace</>
```

Processing of `<m:mspace atts></m:mspace>`. The MathML doc says that attributes width, height, depth, linebreak are allowed. Here we consider only the width. The `\@defaultunits` command is called for the case where a dimension is given without a unit.

```
115 \XMLelement{m:mspace}
116   {\XMLattribute{width}{\XML@mspacewidth}{0}}
117   {}
118   {\@defaultunits\dimen@\XML@mspacewidth pt\relax\@nnil
119    \ifnum\dimen@=\z@\else\kern\dimen@\fi}
```

3.2.1 Fences

Processing of `<m:mfenced open=A close=B separators=...>content</m:mfenced>`. We ignore the separators. The translation is `\left A content \right B`. However, what `\left` and `\right` want are numbers (to be precise, slots into math fonts that indicate how large delimiters can be made from smaller pieces); what TeX wants is a character with a `\delcode`, what the XML file contains is a character. We hack a bit, because we cannot expand all these Unicode characters.

```
120 \XMLelement{m:mfenced}
121   { \XMLattribute{open}{\XML@fenceopen}{(}
```

```
122       \XMLattribute{close}{\XML@fenceclose}{)}
123     }
124     {\jg@hacko\left\XML@fenceopen}
125     {\jg@hackc\right\XML@fenceclose}
```

This code is easy: for instance, if the character is U+2329, we replace it by `\langle`.

```
126 \def\jg@hacko{%
127   \ifx\XML@fenceopen\jg@lt\let\XML@fenceopen\langle\fi
128   \ifx\XML@fenceopen\jg@lbra\let\XML@fenceopen\langle\fi
129   \ifx\XML@fenceopen\jg@xlbra\let\XML@fenceopen\langle\fi
130   \ifx\XML@fenceopen\jg@gt\let\XML@fenceopen\rangle\fi
131   \ifx\XML@fenceopen\jg@rbra\let\XML@fenceopen\rangle\fi
132   \ifx\XML@fenceopen\jg@xrbra\let\XML@fenceopen\rangle\fi
133   \ifx\XML@fenceopen\jg@verbar\let\XML@fenceopen\|\fi
134   \ifx\XML@fenceopen\jg@Verbar\let\XML@fenceopen\|\fi
135   \ifx\XML@fenceopen\jg@lfloor\let\XML@fenceopen\lfloor\fi
136   \ifx\XML@fenceopen\jg@rfloor\let\XML@fenceopen\rfloor\fi
137   \ifx\XML@fenceopen\jg@lceil\let\XML@fenceopen\lceil\fi
138   \ifx\XML@fenceopen\jg@rceil\let\XML@fenceopen\rceil\fi
139   \ifx\XML@fenceopen\jg@lmoustache\let\XML@fenceopen\lmoustache\fi
140   \ifx\XML@fenceopen\jg@rmoustache\let\XML@fenceopen\rmoustache\fi
141   \ifx\XML@fenceopen\jg@lgroup\let\XML@fenceopen\lgroup\fi
142   \ifx\XML@fenceopen\jg@rgroup\let\XML@fenceopen\rgroup\fi
143   \ifx\XML@fenceopen\jg@uparrow\let\XML@fenceopen\uparrow\fi
144   \ifx\XML@fenceopen\jg@downarrow\let\XML@fenceopen\downarrow\fi
145   \ifx\XML@fenceopen\jg@Uparrow\let\XML@fenceopen\Uparrow\fi
146   \ifx\XML@fenceopen\jg@Downarrow\let\XML@fenceopen\Downarrow\fi
147   \ifx\XML@fenceopen\jg@updownarrow\let\XML@fenceopen\updownarrow\fi
148   \ifx\XML@fenceopen\jg@Updownarrow\let\XML@fenceopen\Updownarrow\fi
149 }
```

Same code for closing delimiters.

```
150 \def\jg@hackc{%
151   \ifx\XML@fenceclose\jg@gt\let\XML@fenceclose\rangle\fi
152   \ifx\XML@fenceclose\jg@rbra\let\XML@fenceclose\rangle\fi
153   \ifx\XML@fenceclose\jg@xrbra\let\XML@fenceclose\rangle\fi
154   \ifx\XML@fenceclose\jg@lt\let\XML@fenceclose\langle\fi
155   \ifx\XML@fenceclose\jg@lbra\let\XML@fenceclose\langle\fi
156   \ifx\XML@fenceclose\jg@xlbra\let\XML@fenceclose\langle\fi
157   \ifx\XML@fenceclose\jg@verbar\let\XML@fenceclose\|\fi
158   \ifx\XML@fenceclose\jg@Verbar\let\XML@fenceclose\|\fi
159   \ifx\XML@fenceclose\jg@lfloor\let\XML@fenceclose\lfloor\fi
160   \ifx\XML@fenceclose\jg@rfloor\let\XML@fenceclose\rfloor\fi
161   \ifx\XML@fenceclose\jg@lceil\let\XML@fenceclose\lceil\fi
162   \ifx\XML@fenceclose\jg@rceil\let\XML@fenceclose\rceil\fi
163   \ifx\XML@fenceclose\jg@lmoustache\let\XML@fenceclose\lmoustache\fi
164   \ifx\XML@fenceclose\jg@rmoustache\let\XML@fenceclose\rmoustache\fi
165   \ifx\XML@fenceclose\jg@lgroup\let\XML@fenceclose\lgroup\fi
166   \ifx\XML@fenceclose\jg@rgroup\let\XML@fenceclose\rgroup\fi
167   \ifx\XML@fenceclose\jg@uparrow\let\XML@fenceclose\uparrow\fi
168   \ifx\XML@fenceclose\jg@downarrow\let\XML@fenceclose\downarrow\fi
169   \ifx\XML@fenceclose\jg@Uparrow\let\XML@fenceclose\Uparrow\fi
170   \ifx\XML@fenceclose\jg@Downarrow\let\XML@fenceclose\Downarrow\fi
```

```
  \ifx\XML@fenceclose\jg@updownarrow\let\XML@fenceclose\updownarrow\fi
  \ifx\XML@fenceclose\jg@Updownarrow\let\XML@fenceclose\Updownarrow\fi
}
```

Here we put each delimiter in a string command.

```
\XMLstring\jg@verbar<>&#x2016;</>
\XMLstring\jg@Verbar<>&#8214;</>
\XMLstring\jg@lfloor<>&#x230A;</>
\XMLstring\jg@rfloor<>&#x230B;</>
\XMLstring\jg@lceil<>&#x2308;</>
\XMLstring\jg@rceil<>&#x2309;</>
\XMLstring\jg@lmoustache<>&#x23B0;</>
\XMLstring\jg@rmoustache<>&#x23B1;</>
\XMLstring\jg@lgroup<>&#x3014;</>
\XMLstring\jg@rgroup<>&#x3015;</>
\XMLstring\jg@uparrow<>&#x2191;</>
\XMLstring\jg@Uparrow<>&#x21D1;</>
\XMLstring\jg@downarrow<>&#x2193;</>
\XMLstring\jg@Downarrow<>&#x21D3;</>
\XMLstring\jg@updownarrow<>&#x2195;</>
\XMLstring\jg@Updownarrow<>&#x21D5;</>
```

The Unicode standard says that U+2329 and U+232A should not be used for math. For this reason, we add U+27E8 and U+27E9 in this list.

```
\XMLstring\jg@lbra<>&#x2329;</>
\XMLstring\jg@rbra<>&#x232A;</>
\XMLstring\jg@xlbra<>&#x27E8;</>
\XMLstring\jg@xrbra<>&#x27E9;</>
```

3.2.2 Accents

The translation by Tralics of `$\vec x$` is `<mover accent='true'> <mi>x</mi> <mo>&rightarrow; </mo> </mover>`. A non obvious point is how to translate this back. This idea is that we evaluate the accent in a context (defined by `\notinover`), this will set the two commands `\cur@mo@content` and `\jg@cur@acc`. Let's declare them here.

```
\let\notinover\relax
\def\cur@mo@content{TOTO}
\let\jg@cur@acc\relax
```

Processing of `<m:munderover accent=`*bool* `accentunder=`*bool*`>` *base underscript overscript* `</m:munderover>`. This is wrong. Moreover, attributes are ignored.

```
\XMLelement{m:munderover}
  { }
  {\xmlgrab}
  {\xmltexthreechildren\@firstofone\sb\sp#1}
```

Processing of `<m:mover accent=`*bool*`>` *base overscript* `</m:mover>`. We use an intermediate command. If no accent, we use `\stackrel`. Otherwise, we call a command, say `\jg@bindings`, evaluate the accent in an environment where `\notinover` is `\over`, then execute two commands `\cur@mo@content` and `\jg@cur@acc` (`\X` and `\Y` for simplicity). Evaluating the overscript should define `\X`, and evaluating `\X` should define `\Y`, in such a way that `\Y` applied to the base should put an accent on it.

```
\XMLelement{m:mover}
  {\XMLattribute{accent}{\XML@overaccent}{none}}
  {\xmlgrab}
  {\xmltextwochildren\xml@implement@over{}#1}
```

The auxiliary function.

```
\def\xml@implement@over#1#2{%
 \ifx\XML@overaccent\att@true {%
   \jg@bindings
   \let\notinover\over #2\let\notinover\relax \cur@mo@content\jg@cur@acc{#1}%
 }\else\stackrel{#2}{#1}\fi}
```

Processing of `<m:munder accentunder=`*bool*`>` *base underscript* `</m:munder>`. We use an intermediate command: same idea as above. If no accent, we use `\underset`.

```
\XMLelement{m:munder}
  {\XMLattribute{accentunder}{\XML@underaccent}{none}}
  {\xmlgrab}
  {\xmltextwochildren\xml@implement@under{}#1}
```

```
\def\xml@implement@under#1#2{%
 \ifx\XML@underaccent\att@true {%
   \jg@ubindings
   \let\notinover\over #2\let\notinover\relax \cur@mo@content\jg@cur@acc{#1}%
 }\else \underset{#2}{#1}\fi}
```

Processing of `<m:mo` *atts*`>...</m:mo>`. The MathML norm says that the following attributes are recognized: form, fence, separator, lspace, rspace, stretchy, symmetric, maxsize, minsize, largeop, movablelimits, and accent. We implement only two attributes. We assume that, if form = 'prefix' is given, that our operator is like log or sin, and never used as an accent. Thus, we generate a `\mathop` with an operator font. We set `\limits` or `\nolimits`, depending on the value of the movablelimits attribute. Otherwise, some command is called.

```
\XMLelement{m:mo}
  {\XMLattribute{form}{\XML@mathmlform}{inline}%
   \XMLattribute{movablelimits}{\XML@movablelimits}{false}}
  {\xmlgrab}
  {\ifx\XML@mathmlform\att@PREFIX
     \ifx\XML@movablelimits\att@true
       \mathop{\operator@font #1}\limits
     \else
       \mathop{\operator@font #1}\nolimits
     \fi
   \else\special@mo{#1}\fi}
```

For some strange reason, the default code does not produce a less than sign. Hence we hack a bit. In the case of accent, we have two commands, let's call them `\X` and `\Y`. We define `\X`, so that it will define `\Y`. The definition is global (it must be seen by the caller of the `<mo>` element). In the case of grave accent, what we have here is a 7bit character, and we have to set `\Y` directly. In the case of 3 or 4 dots, we have to set this command.

```
\def\special@mo#1{%
     \def\jg@tck{#1}% save the argument
     \ifx\notinover\over% we cannot typeset here
        \ifx\jg@tck\jg@accgrave % strange
          \global\let\jg@cur@acc\jg@grave@acc
          \global\let\cur@mo@content\relax
```

```
236         \else \ifx\jg@tck\jg@accdddot % strange
237           \global\let\jg@cur@acc\dddot
238           \global\let\cur@mo@content\relax
239         \else \ifx\jg@tck\jg@accddddot % strange
240           \global\let\jg@cur@acc\ddddot
241           \global\let\cur@mo@content\relax
242         \else\gdef\cur@mo@content{#1}\fi\fi\fi
243       \else % typeset the argument, handle < and > in the correct way
244       \ifx\jg@tck\jg@gt\string>\else
245       \ifx\jg@tck\jg@lt\string<\else
246       #1\fi\fi\fi}
```

Some declarations, needed above.

```
247 \XMLstring\jg@lt<>&lt;</>
248 \XMLstring\jg@gt<>&gt;</>
249 \XMLstring\jg@accgrave<>`</>
250 \XMLstring\jg@accdddot<>&#x20DB;</>
251 \XMLstring\jg@accddddot<>&#x20DC;</>
```

This is the big hack. Remember that, in the case of `\vec`, the accent is an arrow, in fact, a Unicode character, that typesets as an arrow. What we do is to redefine `\rightarrow`, in order to put in `\Y` (in fact in `\jg@cur@acc`) a command that behaves like `\vec`. In the case of a breve accent, the character 2D8 is defined to be `\ifmmode \u\else \textasciibreve \fi`, so that we have to redefine `\u`. In the case of the character 302, the definition is `\ifmmode \hat{}\else \^{}\fi`, so that we have to redefine `\hat` in such a way that it reads an argument, and evaluates to something that looks like `\hat`.

```
252 \def\jg@bindings{%
253    \def\texttildelow {\global\let\jg@cur@acc\jg@tilde@acc}%
254 %   \def\textasciimacron {\global\let\jg@cur@acc\jg@cur@accB}%
255     \def\textasciicircum{\global\let\jg@cur@acc\jg@hat@acc}
256    \def\textasciicaron {\global\let\jg@cur@acc\jg@check@acc}%
257    \def\u{\global\let\jg@cur@acc\jg@breve@acc}%
258    \def\hat##1{\global\let\jg@cur@acc\jg@hat@acc}%
259    \def\dot##1{\global\let\jg@cur@acc\jg@dot@acc}%
260    \def\mathring##1{\global\let\jg@cur@acc\jg@ring@acc}%
261    \def\textasciidieresis{\global\let\jg@cur@acc\jg@ddot@acc}%
262    \let\JGG@orig@rarrow\rightarrow
263    \let\JGG@orig@larrow\leftarrow
264    \def\rightarrow{\global\let\jg@cur@acc\jg@overRarrow@acc}%
265    \def\leftarrow{\global\let\jg@cur@acc\jg@overLarrow@acc}%
266    \def\textoverbrace{\global\let\jg@cur@acc\overbrace}
267    \def\textunderbrace{\global\let\jg@cur@acc\underbrace}
268    \def\textasciiacute{\global\let\jg@cur@acc\acute}
269    \def\textasciimacron{\global\let\jg@cur@acc\overline}
270    \def\ring{\global\let\jg@cur@acc\mathring}
271 }
```

The same idea is used in the case of underaccent.

```
272 \def\jg@ubindings{%
273 \let\JGG@orig@rarrow\rightarrow
274 \let\JGG@orig@larrow\leftarrow
275    \def\texttildelow {\global\let\jg@cur@acc\jg@tilde@acc}%
276    \def\textasciimacron {\global\let\jg@cur@acc\underline}%
```

```
    \def\textasciicaron {\global\let\jg@cur@acc\jg@check@acc}%
    \def\u{\global\let\jg@cur@acc\jg@breve@acc}%
    \def\hat##1{\global\let\jg@cur@acc\jg@hat@acc}%
    \def\dot##1{\global\let\jg@cur@acc\jg@dot@acc}%
    \def\textasciidieresis{\global\let\jg@cur@acc\jg@ddot@acc}%
    \let\JGG@orig@rarrow\rightarrow
    \let\JGG@orig@larrow\leftarrow
    \def\rightarrow{\global\let\jg@cur@acc\jg@underRarrow@acc}%
    \def\leftarrow{\global\let\jg@cur@acc\jg@underLarrow@acc}%
    \def\textoverbrace{\global\let\jg@cur@acc\overbrace}
    \def\textunderbrace{\global\let\jg@cur@acc\underbrace}
    \def\jgunderline{\global\let\jg@cur@acc\underline}
}
```

This defines braces as delimiters. The plain.tex file says: N.B. { and } should NOT get delcodes; otherwise parameter grouping fails!

```
\global\delcode`{"66308
\global\delcode`}"67309
```

This is the original definition of the accent commands.

```
\def\jg@tilde@acc{\mathaccent"707E }
\def\jg@check@acc{\mathaccent"7014 }
\def\jg@breve@acc{\mathaccent"7015 }
\def\jg@hat@acc{\mathaccent"705E }
\def\jg@dot@acc{\mathaccent"705F }
\def\jg@ddot@acc{\mathaccent"707F }
\def\jg@grave@acc{\mathaccent"7012 }
\def\jg@ring@acc{\protect \mathaccentV {mathring}017 }
```

Note that when `\jg@overRarrow@acc` is called, `\rightarrow` does something strange. We must redefine it here, because `\rightarrowfill@` uses it.

```
\def\jg@overRarrow@acc{\let\rightarrow\JGG@orig@rarrow
  \mathpalette{\overarrow@\rightarrowfill@}}
\def\jg@overLarrow@acc{\let\leftarrow\JGG@orig@larrow
  \mathpalette{\overarrow@\leftarrowfill@}}
\def\jg@underRarrow@acc{\let\rightarrow\JGG@orig@rarrow
   \mathpalette{\underarrow@\rightarrowfill@}}
\def\jg@underLarrow@acc{\let\leftarrow\JGG@orig@larrow
  \mathpalette{\underarrow@\leftarrowfill@}}
```

3.2.3 More math

This is the main math element. Only the display attribute is taken into account. It can be 'inline' or 'block'. If it is block, we use brackets, otherwise parentheses. The original code defined a command `\GATHER`.

```
\XMLelement{m:math}
  {\XMLattribute{display}{\XML@mathmlmode}{inline}}
  {
     \ifx\XML@mathmlmode\att@BLOCK\[\else\(\fi
   }
  {
     \ifx\XML@mathmlmode\att@BLOCK\]\else\)\fi
  }
```

Processing of `<m:mfrac linethickness=A numalign=B denomalign=C bevelled=D>num denom</m:mfrac>`. We ignore the bevelled attribute. In the current version, we also ignore the horizontal alignment attributes.

```
\XMLelement{m:mfrac}
  {\XMLattribute{linethickness}{\XML@linethickness}{true}%
   \XMLattribute{numalign}{\XML@numalign}{center}%
   \XMLattribute{denomalign}{\XML@denomalign}{center}%
  }
  {\xmlgrab}
  {\xmltextwochildren\xml@implement@frac{}#1}
```

The auxiliary command.

```
\def\xml@implement@frac#1#2{%
  \ifx\XML@linethickness\att@true\frac{#1}{#2}%
  \else \genfrac{}{}\XML@linethickness{}{#1}{#2}\fi
```

3.2.4 Tables

Processing of `<m:mtable atts>body</m:mtable>`. The MathML specification says that the following attributes are allowed: align, rowalign, columnalign, groupalign, alignmentscope, columnwidth, width, rowspacing, columnspacing, rowlines, columnlines, frame, framespacing, equalrows, equalcolumns, displaystyle, side, and minlabelspacing. They are currently all ignored. New implementation, dated January 2005.

```
\XMLelement{m:mtable}
  {}
  {\xmlgrab }
  {\jg@start@mtable#1\jg@end@mtable }
```

Processing of `<m:mtr atts>body</m:mtr>`. The attributes are: rowalign, columnalign, and groupalign.

```
\XMLelement{m:mtr}
  {}
  {\xmlgrab }
  {\jg@start@mtr#1}
```

Processing of `<m:mtd atts>body</m:mtd>`. The attributes are: rowspan, columnspan, rowalign, columnalign, and groupalign.

```
\XMLelement{m:mtd}
 {\XMLattribute{columnalign}{\XML@mtdalign}{center}
  \XMLattribute{columnspan}{\XML@mtdspan}{1}}
 {\xmlgrab}
 {\jg@start@mtd{#1}}
```

The current implementation of math tables uses two token lists, one holds the table, the other one holds the current row. The same holds for normal tables, but since normal tables can contain math tables, we need four lists.

```
\newtoks\jg@mtable@toks
\newtoks\jg@mrow@toks
\newtoks\jg@table@toks
\newtoks\jg@row@toks
```

There is a command `\addto@hook` that inserts some tokens at the end of a token list; it is similar to `\gaddto@hook` without the `\global`. The command `\merge@toks` appends the current row to the table (action has to be global, because it is executed from the group that defines the

row). We also need a command that adds the content of a command to the back of the token list, this is used by normal table when constructing the optional argument of \\.

```
\newcommand\merge@toks[2]{%
  \expandafter\gaddto@hook\expandafter#1\expandafter{\the#2}}
\long\def\gaddto@hook#1#2{\global#1\expandafter{\the#1#2}}
\newcommand\merge@cmd[2]{%
  \expandafter\gaddto@hook\expandafter#1\expandafter{#2}}
```

When we see the start of a table, we initialize the token list. What we will finally evaluate is an array, declared as `*{99}{c}`. In the current implementation, it is possible to count the number of columns, and replace 99 by a better value, but this would cost more than constructing a preamble of length 99. We shall see later that, for normal tables, we count the number of columns. Note that, in amsmath, matrices are declared in this way, using the counter `MaxMatrixCols`, with initial value 10, instead of the constant 99. We initialize the row token list to be empty, and `\ifStartTable` to true (this means that the next row is the first one).

```
\newif\ifStartTable\newif\ifStartRow
\newif\ifStartRowx\newif\ifStartTablex
\newcommand\jg@start@mtable{%
 \jg@mtable@toks{\begin{array}{*{99}{c}}}%
 \jg@mrow@toks{}%
 \StartTabletrue}
```

When the end of the array is seen, we insert the last row and the `\end{array}` in the token list, then evaluate it.

```
\newcommand\jg@end@mtable{%
 \merge@toks\jg@mtable@toks\jg@mrow@toks
 \addto@hook\jg@mtable@toks{\end{array}}%
 \the\jg@mtable@toks}
```

When we see the start of a row, we insert the previous row into the table; we reset it; we define the command `\ifStartRow` to true (this means that the next cell is the first in the row). In the case where `\ifStartTable` is true, we set it to false, otherwise we add a \\, the row separator, in the array.

```
\newcommand\jg@start@mtr{%
  \merge@toks\jg@mtable@toks\jg@mrow@toks
  \ifStartTable
     \glbal\StartTablefalse
  \else \gaddto@hook\jg@mtable@toks{\\}\fi
  \jg@mrow@toks{}%
  \StartRowtrue}
```

The main action associated to a cell consists in putting the content in the `\jg@row@toks` token list; in the case where `\ifStartRow` is true, we set it to false, otherwise we add a `&`, the cell separator, in the token list. In `\temp`, we put the content of our cell, plus some `\hfill` in the case where alignment is left or right; if alignment is 'left', we insert an empty group after the `\hfill` (the preamble of the array for our cell is `\hfil\ignorespaces#\unskip\hfil`). In the general case, there is no need to use a temporary command: we could add the tokens directly into the token list; however, in the case where the span is greater than one, the action is a bit more complicated, in this case, we use a temporary token register.

```
\newcommand\jg@start@mtd[1]{%
   \ifStartRow
     \global\StartRowfalse
   \else\gaddto@hook\jg@mrow@toks{\tabcellsep}\fi
   \ifx\XML@mtdalign\att@mtd@left
```

```
      \def\temp{#1\hfill{}}%
    \else\ifx\XML@mtdalign\att@mtd@right
      \def\temp{\hfill#1}%
        \else \def\temp{#1}\fi\fi%
  \ifnum\XML@mtdspan=1\else
   \toks0=\expandafter{\temp}%
   \edef\temp{\noexpand\multicolumn{\XML@mtdspan}{c}{\the\toks0}}%
  \fi
  \expandafter\gaddto@hook\expandafter\jg@mrow@toks\expandafter{\temp}}
```

3.3 Other commands

3.3.1 Pictures

In this section, we define some elements that Tralics constructs when evaluating a command from the epic package. These elements are in the default namespace with 'pic-' prefix; in a future version, we will move them in another namespace.

Processing of `<picture width=A height=B xpos=C ypos=D>...</picture>`. This defines the picture environment. Translation is `\begin{picture} (A,B)(C,D) ... \end{picture}`.

```
\XMLelement{picture}
  {\XMLattribute{width}{\XML@width}{1}
   \XMLattribute{height}{\XML@height}{1}
   \XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{ypos}{\XML@ypos}{0}
  }
  {\begin{picture}(\XML@width,\XML@height)(\XML@xpos,\XML@ypos)}
  {\end{picture}}
```

Processing of `<pic-put xpos=A ypos=B>C</pic-put>`. We translate this into `\put(A,B){C}`.

```
\XMLelement{pic-put}
  {\XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{ypos}{\XML@ypos}{0}}
  {\xmlgrab}
  {\put(\XML@xpos,\XML@ypos){#1}}
```

Processing of `<pic-arc xpos=A ypos=B angle=C></pic-arc>`. We translate this in an obvious manner into `\arc(A,B){C}`.

```
\XMLelement{pic-arc}
  {\XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{angle}{\XML@angle}{0}
   \XMLattribute{ypos}{\XML@ypos}{0}}
  {\xmlgrab}
  {\arc(\XML@xpos,\XML@ypos){\XML@angle}}
```

Processing of `<pic-scaleput xpos=A ypos=B xscale=xs yscale=ys xscaley=xsy yscaley=ysy>C</pic-scaleput>`. We translate this into `\scaleput(A,B){C}`. Other attributes are put in variables named `\xscale`, `\yscale`, `\xscaley`, and `\yscalex`.

```
\XMLelement{pic-scaleput}
  {\XMLattribute{xscale}{\xscale}{1.0}
   \XMLattribute{yscale}{\yscale}{1.0}
   \XMLattribute{xscaley}{\xscaley}{0.0}
   \XMLattribute{yscalex}{\yscalex}{0.0}
```

```
    \XMLattribute{xpos}{\XML@xpos}{0}
    \XMLattribute{ypos}{\XML@ypos}{0}}
  {\xmlgrab}
  {\scaleput(\XML@xpos,\XML@ypos){#1}}
```

Processing of `<pic-thicklines/>` and `<pic-thinlines/>`. We call a command, that is executed just after the group. In the case of `<pic-linethickness size=V/>`, the command takes an argument; we have to expand the value of the attribute, put this in a global variable, and ask for that variable to be expanded after the group. In a first version, we redefined the three commands `\thinlines`, `\thicklines` and `\linethickness`. The current code is simpler.

```
\XMLelement{pic-thicklines}
  {}{}{\aftergroup\thicklines}
```

Case of thin lines.

```
\XMLelement{pic-thinlines}
  {}{}{\aftergroup\thinlines}
```

Case of line thickness.

```
\XMLelement{pic-linethickness}
  {\XMLattribute{size}{\XML@size}{1pt}}
  {}
  {\xdef\temp{\noexpand\linethickness{\XML@size}}\aftergroup\temp}
```

Processing of `<pic-multiput xpos=A ypos=B repeat=C dx=D dy=E>obj</pic-multiput>`. We translate this as `\multiput(A,B)(D,E){C}{obj}`.

```
\XMLelement{pic-multiput}
  {\XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{ypos}{\XML@ypos}{0}
   \XMLattribute{repeat}{\XML@repeat}{1}
   \XMLattribute{dx}{\XML@dx}{1}
   \XMLattribute{dy}{\XML@dy}{1}}
  {\xmlgrab}
  {\multiput(\XML@xpos,\XML@ypos)(\XML@dx,\XML@dy){\XML@repeat}{#1}}
```

Processing of `<pic-bezier a1=A1 a2=A2 b1=B1 b2=B2 c1=C1 c2=C2 repeat=D/>`. Translation is `\qbezier[D](A1,A2)(B1,B2),(C1,C2)`.

```
\XMLelement{pic-bezier}
  {\XMLattribute{a1}{\XML@ai}{0}
   \XMLattribute{a2}{\XML@aii}{0}
   \XMLattribute{b1}{\XML@bi}{0}
   \XMLattribute{b2}{\XML@bii}{0}
   \XMLattribute{c1}{\XML@ci}{0}
   \XMLattribute{c2}{\XML@cii}{0}
   \XMLattribute{repeat}{\XML@repeat}{0}}
  {}
  {
   \qbezier[\XML@repeat](\XML@ai,\XML@aii)(\XML@bi,\XML@bii)(\XML@ci,\XML@cii)
  }
```

Processing of `<pic-line xdir=A ydir=B width=C/>`. The translation is `\line(A,B){C}`.

```
\XMLelement{pic-line}
  {\XMLattribute{xdir}{\XML@xdir}{0}
   \XMLattribute{ydir}{\XML@ydir}{0}
   \XMLattribute{width}{\XML@width}{0}}
```

```
  {}
  {\line(\XML@xdir,\XML@ydir){\XML@width}}
```

Processing of `<pic-vector xdir=A ydir=B width=C/>`. The translation is `\vector(A,B){C}`.

```
\XMLelement{pic-vector}
  {\XMLattribute{xdir}{\XML@xdir}{0}
   \XMLattribute{ydir}{\XML@ydir}{0}
   \XMLattribute{width}{\XML@width}{0}}
  {}
  {\vector(\XML@xdir,\XML@ydir){\XML@width}}
```

Processing of `<pic-curve unit-length=A>B</pic-curve>`. The translation is `\curve(B)`. There are two hacks here: one is that `\curve` modifies `\unitlength` globally, so that we have to reset it. The other hack is that `\ignorespaces` must be ignored: the argument of `\curve` is a sequence of numbers, converted to dimensions by using `\unitlength`. Between a sequence of digits and a unit of measure, spaces are allowed (but here, spaces are active, there expansion is: space plus `\ignorespaces`).

```
\newdimen\jgunitlength
\newcommand\withulength[1]{%
  {\def\ignorespaces{}%
   \jgunitlength=\unitlength \setlength\unitlength{\XML@ulength pt}%
   #1\global\unitlength\jgunitlength}}

\XMLelement{pic-curve}
  {\XMLattribute{unit-length}{\XML@ulength}{1}}
  {\xmlgrab}
  {\withulength{\curve(#1)}}
```

Processing of `<pic-closecurve unit-length=A>B</pic-closecurve>`. With the same hack as above, the translation is `\closecurve(B)`.

```
\XMLelement{pic-closecurve}
  {\XMLattribute{unit-length}{\XML@ulength}{1}}
  {\xmlgrab}
  {\withulength{\closecurve(#1)}}
```

Processing of `<pic-tagcurve unit-length=A>B</pic-tagcurve>`. With the same hack as above, the translation is `\tagcurve(B)`.

```
\XMLelement{pic-tagcurve}
  {\XMLattribute{unit-length}{\XML@ulength}{1}}
  {\xmlgrab}
  {\withulength{\tagcurve(#1)}}
```

Processing of `<pic-frame>X</pic-frame>`. Translation is `\frame{X}`.

```
\XMLelement{pic-frame}
 {}
 {\xmlgrab}
 {\frame{#1}}
```

Processing of `<pic-circle size=A full=B/>`. Translation is `\circle{A}` or `\circle*{A}`. A star is used B is given and not 'false'.

```
\XMLelement{pic-circle}
  {\XMLattribute{size}{\XML@size}{1}
   \XMLattribute{full}{\XML@full}{false}}
  {}
  {\ifx\XML@full\att@false\circle{\XML@size}\else \circle*{\XML@size}\fi}
```

Processing of `<pic-circle size=A unit-length=B/>`. Translation is `\bigcircle{A}`. We locally change `\unitlength`.

```
\XMLelement{pic-bigcircle}
  {\XMLattribute{size}{\XML@size}{1}
   \XMLattribute{unit-length}{\XML@ulength}{1}}
  {}
  {\withulength{\bigcircle{\XML@size}}}
```

The strings defined here are needed in the next command.

```
\XMLstring\att@l<>l</>
\XMLstring\att@r<>r</>
\XMLstring\att@t<>t</>
\XMLstring\att@b<>b</>
\XMLstring\att@lt<>lt</>
\XMLstring\att@lb<>lb</>
\XMLstring\att@rt<>rt</>
\XMLstring\att@rb<>rb</>
\XMLstring\att@tl<>tl</>
\XMLstring\att@bl<>bl</>
\XMLstring\att@tr<>tr</>
\XMLstring\att@br<>br</>
```

Given a string A, in `\XML@pos`, we construct a string B, that has the same characters (order is irrelevant), with category code 11. Moreover, if we have a command `\bar` and an argument `gee`, we evaluate `\bar[B]{gee}` (in fact, we construct a command that does this; this command will be evaluated after all conditionals have been closed.)

```
\def\@att@to@rtb#1#2{%
  \ifx\XML@pos\att@l \def\jg@tmp{#1[l]{#2}}
  \else\ifx\XML@pos\att@r \def\jg@tmp{#1[r]{#2}}%
  \else\ifx\XML@pos\att@t \def\jg@tmp{#1[t]{#2}}%
  \else\ifx\XML@pos\att@b \def\jg@tmp{#1[b]{#2}}%
  \else\ifx\XML@pos\att@lt \def\jg@tmp{#1[lt]{#2}}%
  \else\ifx\XML@pos\att@lb \def\jg@tmp{#1[lb]{#2}}%
  \else\ifx\XML@pos\att@rt \def\jg@tmp{#1[rt]{#2}}%
  \else\ifx\XML@pos\att@rb \def\jg@tmp{#1[rb]{#2}}%
  \else\ifx\XML@pos\att@tl \def\jg@tmp{#1[lt]{#2}}%
  \else\ifx\XML@pos\att@bl \def\jg@tmp{#1[lb]{#2}}%
  \else\ifx\XML@pos\att@tr \def\jg@tmp{#1[rt]{#2}}%
  \else\ifx\XML@pos\att@br \def\jg@tmp{#1[rb]{#2}}%
  \else \def\jg@tmp{#1{#2}}
 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
 \jg@tmp
}
```

Processing `<pic-framebox width=A height=B position=C framed=D>obj</pic-framebox>`. The translation is `\framebox(A,B)[CC]{obj}` (it could be `\makebox` if the attribute framed is false). Here CC is the value of C, processed as indicated above.

```
\XMLelement{pic-framebox}
  {\XMLattribute{width}{\XML@width}{0}
   \XMLattribute{height}{\XML@height}{0}
   \XMLattribute{position}{\XML@pos}{}
   \XMLattribute{framed}{\XML@framed}{false}
  }
```

```
 {\xmlgrab}
 {\let\cmd\framebox\ifx\XML@framed\att@false\let\cmd\makebox\fi
  \@att@to@rtb{\cmd(\XML@width,\XML@height)}{#1}}
```

Processing `<pic-dashbox width=A height=B position=C dashdim=D> obj </pic-dashbox>`. The translation is `\dashbox{D}(A,B)[CC]{obj}`, where CC is as above.

```
\XMLelement{pic-dashbox}
  {\XMLattribute{width}{\XML@width}{0}
   \XMLattribute{height}{\XML@height}{0}
   \XMLattribute{position}{\XML@pos}{w}
   \XMLattribute{dashdim}{\XML@dashdim}{1pt}
  }
 {\xmlgrab}
 {\@att@to@rtb{\dashbox{\XML@dashdim}(\XML@width,\XML@height)}{#1}}
```

Processing of `<pic-oval xpos=A ypos=B specs=C>obj</pic-oval>`.
The translation is `\oval(A,B)[CC]{obj}`, where CC is as above.

```
\XMLelement{pic-oval}
  {\XMLattribute{xpos}{\XML@xpos}{0}
   \XMLattribute{specs}{\XML@pos}{}
   \XMLattribute{ypos}{\XML@ypos}{0}}
  {\xmlgrab}
  {\@att@to@rtb{\oval(\XML@xpos,\XML@ypos)}{#1}}
```

3.3.2 Titlepage

We found no easy way to describe the title page of the Raweb using XSL/Format; thus additional elements were invented and described here. The `\ra@atxy` command adds a box (that occupies no space) at a give position in the page (the position is absolute).

```
\newbox\ra@atxybox
\def\ra@atxy(#1,#2)#3{\global\setbox\ra@atxybox=\hbox
 {\unhbox\ra@atxybox
  \vtop to 0pt{\kern #2\hbox to 0pt{\kern #1\relax #3\hss}\vss}}}
```

We use `\@texttop` (a command that does nothing)[1] for insertion of our box. After that, we kill our command. We have to shift horizontally by 1in plus `\hoffset` and the current margin, vertically by 1in, plus `\voffset` plus `\headheight` plus `\headsep` plus `\topmargin`.

```
\def\ra@useatxy{{%
  \let\@themargin\oddsidemargin
  \vtop to 0pt{\kern-\headsep \kern-\topmargin \kern-\headheight
               \kern-1in \kern-\voffset
    \hbox to 0pt{\kern-\@themargin \kern-1in \kern-\hoffset
    \unhbox\ra@atxybox \hss}\vss}}%
    \global\let\@texttop\relax}
```

The command `\firstchar` is nowadays useless. In the times when Inria themes were 1A, 1B, 2A, 2B etc, we used it to remove the letter. Nowadays themes are NUM, COG, etc, and we hope people give only the theme.[2] We insert the logo, and, above it, something that looks like

_____ THEME NUM _____.

[1] This is redefined by `\raggedbottom` to be empty; you have to find a different trick if you want to insert something at a given position using this code as a model.

[2] and not the subtheme; we could use a command that picks up the first three characters.

```
\def\foratheme#1{\vskip8cm \vfil
  \ra@atxy(74mm,175mm) {\hbox to 70mm{%
       \hrulefill\hspace{8mm}
       \def\firstchar##1##2\relax{##1##2} % ducon
       \href{http://www.inria.fr/recherche/equipes/listes/theme%
         _\firstchar#1\relax.en.html}{THEME \uppercase{#1}}%
       \hspace{8mm}\hrulefill}}}
```

Start of the title page.

```
\def\foinria{%
  \ra@atxy(7.8cm,2.5cm){\includegraphics[width=5.7cm]{Logo-INRIA-couleur}}%
  \ra@atxy(55mm,173mm){\includegraphics{LogoRA\ra@year}}%
 \setbox0\hbox to 14cm{%
     \noindent\hskip3cm\hfill
     {\fontencoding{T1}\fontfamily{ptm}\fontseries{m}%
    \fontshape{n}\fontsize{10pt}{12pt}\selectfont
\href{http://www.inria.fr}{INSTITUT NATIONAL DE RECHERCHE EN %
     INFORMATIQUE ET EN AUTOMATIQUE}}%
      \hskip-5cm\hfill}%
  \null\vskip0.7cm \leavevmode\hskip-3.5cm\box0\null\vskip2cm\vfil}
```

This is a hack: we just want `\cleardoublepage`. This is not used anymore.

```
\XMLelement{cleardoublepage}
 {} {\cleardoublepage} {}
```

This is a hack: we want a rule below the page headings. We found no other way. This is not used anymore.

```
\XMLelement{pagestylehrule}
 {} {\hbox to0pt{\rule[-1ex]{\textwidth}{.03cm}\hss}} {}
```

3.3.3 Images

For the HTML version of the Raweb, we convert all math formulas into images. The idea is to construct an XML file that contains a `<formula>` for each object to convert. This file is translated by TEX into a dvi file, containing one formula per page; after that the dvi is converted to PostScript then png, or some other image format recognized by HTML.

A non trivial problem is the size of the image. When TEX translates the formula, it creates a box, and prints the dimension of the box on the transcript file. We want the baseline of the image to be aligned with the baseline of the text, but HTML provides only 'center' or 'bottom' as alignment options; for this reason, if the alignment cannot be 'bottom', said otherwise, if the depth of the box is non-zero, we add some blank space so that the height and depth will be the same. If two much space is added, this is not a good thing to do.

Assume that x is (a bit more than) the maximum of the height and depth of `\sizebox`. We modify the height and depth to be x; we insert a vertical invisible rule of height x and depth x. This code is not used any more.

```
\def\@centerinlinemath{%
  \dimen1=\ifdim\ht\sizebox<\dp\sizebox \dp\sizebox\else\ht\sizebox\fi
  \advance\dimen1by.5pt \vrule width0pt height\dimen1 depth\dimen1
 \dp\sizebox=\dimen1\ht\sizebox=\dimen1\relax}
```

We noticed that if the images contains a single table, it is not vertically centered. In fact, the following code

```
$\vcenter{\vrule height 0pt depth5pt width2pt}$
```

produces a box of height 5pt, and depth 0pt (for details, see TeXbook, appendix G, rule 8: the distance between the center of the box and the baseline is parameter σ_{22} of the current math font). In fact, the `\vcenter` command is designed for math mode only, and the the distance between the center of the box and the math axis is zero.

In a first version, we used the 'E' option of dvips; this produces a set of PostScript files, one for each formula, with the correct bounding box. This bounding box is computed by dvips, according to characters and rules of the dvi file. If the TeX file contains a `\special` command that asks a PostScript interpreter to draw a circle of radius R, and an empty box that has the size of the circle, followed by some text, then you will see the circle followed by the text when viewing the PostScript file; but dvips is not a PostScript interpreter and has no idea of the size of the circle.

We solve this problem by adding two rules, one at the left of the image and one below. If everything works correctly, conversion from PostScript to png will remove them, but this is not always the case. The code is taken from `latex2html`, it has a strange feature. In the case of overflow, only the vertical rule is added. This is the code that adds rules to the box in `\sizebox`.

```
\hbox{\vrule width1pt\kern-.5pt\vtop{\vbox{%
 \kern1pt\kern0.6 pt\hbox{\hglue1.7pt\copy\sizebox\hglue0.6 pt}}\kern.3pt%
 \ifdim\dp\sizebox>0pt\kern1pt\fi \kern0.6 pt%
 \ifdim\hsize>\wd\sizebox \hrule depth1pt\fi}}%
 \ifdim\ht\sizebox<\vsize
     \ifdim\wd\sizebox<\hsize\hfill\fi
     \vfill
 \else\vss\fi
```

Assume that (x,y) is the position of the upper left corner of the image, (X,Y) are the dimensions of the image. If ϵ is the width of the rule plus the distance to the image, then the bounding box of the page will be $x-\epsilon$, $x+X$, y, $y+Y+\epsilon$. These quantities can be computed by dvips; it is then obvious to remove the two rules.

We now use a different method. The bounding box is not computed anymore by dvips; we call an external program with the parameters x, y, X and Y. This program has a comment that says: *should (x,y) be (78,72) or (72,72) ?* but it uses (62,41); if `pstoimg` is unavailable, then `convert` is used with (64, 44). Remember that TeX puts its origin at 1in (this is 72pt) from the upper left corner, but we have no idea what the current margins could be. If we ask TeX what it puts on the dvi, we see that the main vertical box contains 16pt of glue, and a box shifted by 62pt, this one starts with a header, then 25pt of glue. After that we see a vbox containing some kerns: -25pt, -16pt, -12pt, and two kerns with sum 0, and absolute value 72.27pt. We also noticed that the message *Underfull \hbox (badness 10000) has occurred while \output is active []*[][][] printed on each page is a consequence of loading the hyperref package, that tries to put something in the header (which is empty, of course); as a consequence, we have patched the file, so that the hyperref package is not loaded in the case were we just want to convert a piece of XML into an image.

The result is cropped. This means: if Y is too big, a smaller value will be used; in particular this will remove the additional blank space added (as explained above for vertical centering). We simplified a bit the algorithm: no rules are printed, the size is no more adjusted.

There are two elements `<formula>` and `<tree>` that produce images. A formula contains a `<math>` element, and a tree contains a table (that defines the nodes and their layout) together with the connections. When we see a formula in a table, we simply ignore it (we typeset the math). Nothing special has to be done for math formulas, because they contain no tables, no trees (only MathML stuff).

Here we translate `<formula id=A>math</formula>`; the id attribute is the number of the image.

```
5 \XMLelement{formula}
6  {\XMLattribute{id}{\XML@formid}{none}}
```

```
 {\formula@start}
 {\formula@end}
```

The two commands `\@inlinemathA` and `\@inlinemathZ` are used to start and finish a math image; originally, they were used directly in the code of `<formula>`. When we evaluate a `<tree>`, we must change these two commands.

```
\def\formula@start{\@inlinemathA{\XML@formid}}
\let\formula@end\@inlinemathZ
```

There are two differences between a tree and a formula. To start with, we do not have a tree number, so that we invent one; in reality, the tree number is irrelevant. The second difference is that nodes in a tree can be connected by non-straight lines; as a consequence the size needed by the tree is bigger than the size of the table. We allow a kern at the left of the table and below the table.

```
\XMLelement{tree}
 {\XMLattribute{hpos}{\XML@hpos}{0pt}
   \XMLattribute{vpos}{\XML@vpos}{0pt}
 }
 {
 \stepcounter{treecounter}
 \let\formula@start\relax
 \let\formula@end\relax
 \startdimen=\XML@hpos
 \enddimen=\XML@vpos
 \@inlinemathA{\thetreecounter}}
 {\@inlinemathZ}
```

It may happen that space must be added around the object. Currently, we need space on the left and bottom, thus we declare these two variables. Since assignment is local, there is no need to reset them.

```
\newdimen\startdimen
\newdimen\enddimen
```

For some strange reasons, the fotex.sty file says that eps files should be included by converting them into pdf; this works for pdfTEX, but not for LATEX. We make sure here that this rule is never applied.

```
\newbox\sizebox
\AtBeginDocument{
\@namedef{Gin@rule@.eps}#1{{eps}{.eps}{#1}}
```

We redefine `\normalsize`, in the same way as `latex2html`. This is really, really, strange.

```
\let\realnormalsize=\relax
\def\adjustnormalsize{%
  \def\normalsize{\mathsurround=0pt \realnormalsize
   \parindent=0pt\abovedisplayskip=0pt\belowdisplayskip=0pt}%
  \def\phantompar{\csname par\endcsname}%
  \normalsize}}
```

The action at the start of a formula is the following: we remember the id somewhere, we start a new page, and we construct a box; this box will start with a strut and some space.

```
\def\@inlinemathA#1{%
  \xdef\@mathenv{#1}%
  \adjustnormalsize \newpage\clearpage
  \setbox\sizebox=\hbox\bgroup
```

```
  \vrule height1.5ex width0pt
  \kern\startdimen }
```

This is done at the end. The action is the following: we terminate the box, we modify the depth of the box in the case additional vertical space is needed, and print the dimensions on the log file. We insert the box on the current page and start a new one.

```
\def\@inlinemathZ{%
  \egroup
  \ifdim\enddimen>0pt
    \dimen1=\dp\sizebox\advance\dimen1by \enddimen\dp\sizebox=\dimen1
  \fi
  \typeout{l2hSize %
    :\@mathenv:\the\ht\sizebox::\the\dp\sizebox::\the\wd\sizebox.}
  \box\sizebox
  \clearpage
}
```

3.3.4 Trees

A tree is defined by a set of nodes, together with connections. The elements here are handled by the tree-dvips package, using commands described in part I of this report. All elements, except <node> have an empty content. Translation of element <foo> is in general \foo, with arguments depending on the attributes.

The content of a <node> is typeset normally, it has an a name attribute, that identifies it uniquely (On each page in the PostScript file, there is a unique node with a given name).

```
\XMLelement{node}
 {\XMLattribute{name}{\XML@name}{somename}}
 {\xmlgrab}
 {\node{\XML@name}{#1}}
```

A <nodepoint> element has a name, and two optional attributes xpos, ypos.

```
\XMLelement{nodepoint}
 {\XMLattribute{name}{\XML@name}{somenode}
  \XMLattribute{xpos}{\XML@xpos}{0pt}
  \XMLattribute{ypos}{\XML@ypos}{0pt}}
 {\xmlgrab}
 {\nodepoint{\XMl@name}[\XML@xpos][\XML@ypos]}
```

A <nodebox> element has a nameA attribute. The effect is to add a frame around the node defined by the name.

```
\XMLelement{nodebox}
 {\XMLattribute{nameA}{\XML@nameA}{somenode}
 }
 {\xmlgrab}
 {\nodebox{\XML@nameA}}
```

A <nodeoval> element has a nameA attribute. The effect is to put an oval around the node defined by the name.

```
\XMLelement{nodeoval}
 {\XMLattribute{nameA}{\XML@nameA}{somenode}
 }
 {\xmlgrab}
 {\nodeoval{\XML@nameA}}
```

A <nodecircle> element has a nameA attribute. The effect is to put an circle around the node defined by the name. There is another attribute depth that specifies a parameter of the circle.

```
\XMLelement{nodecircle}
 {\XMLattribute{nameA}{\XML@nameA}{somenode}
  \XMLattribute{depth}{\XML@depth}{0pt}
 }
 {\xmlgrab}
 {\nodecircle[\XML@depth]{\XML@nameA}}
```

A <nodetriangle> element has two names, nameA and nameB. The effect is to insert a triangle between the nodes defined by the names.

```
\XMLelement{nodetriangle}
 {\XMLattribute{nameA}{\XML@nameA}{nodeA}
  \XMLattribute{nameB}{\XML@nameB}{nodeB}
 }
 {\xmlgrab}
 {\nodetriangle{\XML@nameA}{\XML@nameB}}
```

Both elements <nodeconnect> and <anodeconnect> have a nameA and nameB attributes; the effect is to connect the nodes defined by the names; if the first letter of the element is a, there will be an arrow. The attributes posA and posB explain the positions of the connection, it can be one of top, bottom, left, right, or a combination. For instance 'br' stands for bottom right. This is the reverse order of that chosen for pictures in section 3.3.1, but we check nothing here.

```
\XMLelement{nodeconnect}
 {\XMLattribute{nameA}{\XML@nameA}{nodeA}
  \XMLattribute{nameB}{\XML@nameB}{nodeB}
  \XMLattribute{posA}{\XML@posA}{b}
  \XMLattribute{posB}{\XML@posB}{t}
 }
 {\xmlgrab}
 {\nodeconnect[\XML@posA]{\XML@nameA}[\XML@posB]{\XML@nameB}}

\XMLelement{anodeconnect}
 {\XMLattribute{nameA}{\XML@nameA}{nodeA}
  \XMLattribute{nameB}{\XML@nameB}{nodeB}
  \XMLattribute{posA}{\XML@posA}{b}
  \XMLattribute{posB}{\XML@posB}{t}
 }
 {\xmlgrab}
 {\anodeconnect[\XML@posA]{\XML@nameA}[\XML@posB]{\XML@nameB}}
```

Both elements <barnodeconnect> and <abarnodeconnect> have a nameA and nameB attributes; the effect is to connect the nodes defined by the names; if the first letter of the element is a, there will be an arrow. The attribute depth is optional.

```
\XMLelement{barnodeconnect}
 {\XMLattribute{nameA}{\XML@nameA}{nodeA}
  \XMLattribute{nameB}{\XML@nameB}{nodeB}
  \XMLattribute{depth}{\XML@depth}{5pt}
 }
 {\xmlgrab}
 {\barnodeconnect[\XML@depth]{\XML@nameA}{\XML@nameB}}

\XMLelement{abarnodeconnect}
```

```
 {\XMLattribute{nameA}{\XML@nameA}{nodeA}
  \XMLattribute{nameB}{\XML@nameB}{nodeB}
  \XMLattribute{depth}{\XML@depth}{5pt}
 }
 {\xmlgrab}
 {\abarnodeconnect[\XML@depth]{\XML@nameA}{\XML@nameB}}
```

Both elements <nodecurve> and <anodecurve> have a nameA and nameB attributes; the effect is to connect the nodes defined by the names; if the first letter of the element is a, there will be an arrow. The attributes depthA and depthB are optional.

```
\XMLelement{nodecurve}
 {\XMLattribute{nameA}{\XML@nameA}{nodeA}
  \XMLattribute{nameB}{\XML@nameB}{nodeB}
  \XMLattribute{posA}{\XML@posA}{b}
  \XMLattribute{posB}{\XML@posB}{t}
  \XMLattribute{depthA}{\XML@depthA}{5pt}
  \XMLattribute{depthB}{\XML@depthB}{5pt}
 }
 {\xmlgrab}
 {\nodecurve[\XML@posA]{\XML@nameA}[\XML@posB]{\XML@nameB}{\XML@depthA}[\XML@depthB]}

\XMLelement{anodecurve}
 {\XMLattribute{nameA}{\XML@nameA}{nodeA}
  \XMLattribute{nameB}{\XML@nameB}{nodeB}
  \XMLattribute{posA}{\XML@posA}{b}
  \XMLattribute{posB}{\XML@posB}{t}
  \XMLattribute{depthA}{\XML@depthA}{5pt}
  \XMLattribute{depthB}{\XML@depthB}{5pt}
 }
 {\xmlgrab}
 {\anodecurve[\XML@posA]{\XML@nameA}[\XML@posB]{\XML@nameB}{\XML@depthA}[\XML@depthB]}
```

3.3.5 Tables

We use the same method for normal tables as for math table, said otherwise, we read everything, construct the code, and finally evaluate it. This piece of code is only used for trees, so that we assume that cells have no borders. We could handle all attributes.

A <table> has a unique attribute vpos, currently unused.

```
\XMLelement{table}
  {\XMLattribute{vpos}{\XML@vpos}{b}}
  {\xmlgrab }
  {\jg@start@table#1\jg@end@table }
```

A <row> can have lots of attributes, but we consider only one; it defines the vertical space after the current row.

```
\XMLelement{row}
  {\XMLattribute{spaceafter}{\XML@spaceafter}{0pt}}
  {\xmlgrab }
  {\jg@start@tr#1}
```

Attributes right-border, left-border, halign and rows are declared for a <cell>, but only cols is used (this is the number of effective columns spanned by the cell).

```
\XMLelement{cell}
 {\XMLattribute{cols}{\XML@cols}{1}
  \XMLattribute{rows}{\XML@rows}{1}
  \XMLattribute{halign}{\XML@halign}{center}
  \XMLattribute{right-border}{\XML@rightborder}{false}
  \XMLattribute{left-border}{\XML@leftborder}{false}
 }
 {\xmlgrab}
 {\jg@start@td{#1}}
```

Action when we start a table. We kill the two token lists, and say that we are at the start of the table. For our application, we need a lot of space between rows, so that `\arraystretch` is redefined to some value; an intermediate macro is used, it can be redefined, for instance at begin-document, by a config file. The redefinition is local.

```
\def\@myarraystretch{1.7}%
\newcounter{localcolcounter}
\newcounter{globalcolcounter}

\newcommand\jg@start@table{%
 \let\arraystretch\@myarraystretch
 \jg@table@toks{}%
 \jg@row@toks{}%
 \setcounter{globalcolcounter}{1}%
 \StartTablextrue}
```

Action when we see the end of a table. Since we are at the end of the table, we know the number of columns of the last row, so that we can compute the number of columns of the table. After that, we insert the last row to the table. We insert the `\begin` and `\end` commands, then evaluate everything.

```
\newcommand\jg@end@table{%
 \ifnum\value{localcolcounter}>\value{globalcolcounter}%
 \setcounter{globalcolcounter}{\value{localcolcounter}}\fi
 \merge@toks\jg@table@toks\jg@row@toks
 \addto@hook\jg@table@toks{\end{tabular}}%
 \edef\foo{\noexpand\begin{tabular}{*{\theglobalcolcounter}{c}}%
   \the\jg@table@toks}%
 \foo}
```

Action when we see the start of a row. We insert the content of the row token list to the table, and clear it. If this is not the first row of the table, we insert a double backslash, plus its optional argument; the current column counter is the number of columns of this row, and from this we deduce the current number of columns of the table. In any case, we compute the optional argument to be inserted at the end of the row, we say that the number of columns of this row is zero, and we are at the start of the row.

```
\newcommand\jg@start@tr{%
  \merge@toks\jg@table@toks\jg@row@toks
  \ifStartTablex
     \global\StartTablexfalse
  \else
    \gaddto@hook\jg@table@toks{\\}%
    \merge@cmd\jg@table@toks\Rowsep
    \ifnum\value{localcolcounter}>\value{globalcolcounter}%
    \setcounter{globalcolcounter}{\value{localcolcounter}}\fi
```

```
  \fi
  \jg@row@toks{}%
  \ifdim\XML@spaceafter=0pt  \gdef\Rowsep{}\else
  \xdef\Rowsep{[\XML@spaceafter]}\fi
  \setcounter{localcolcounter}{0}%
  \global\StartRowxtrue}
```

Action when we see a cell (in the argument of the command). If this not the first cell of the row, we insert an ampersand character (in `\tabcellsep`). We increment the current column number by N, the span; if this is not one, we must use `\multicolumn`.

```
\newcommand\jg@start@td[1]{%
   \ifStartRowx\global\StartRowxfalse
   \else \gaddto@hook\jg@row@toks{\tabcellsep}\fi
  \addtocounter{localcolcounter}{\XML@cols}%
   \ifx\XML@mtdalign\att@mtd@left
      \def\temp{#1\hfill{}}%
    \else\ifx\XML@mtdalign\att@mtd@right
      \def\temp{\hfill#1}%
        \else \def\temp{#1}\fi\fi%
  \ifnum\XML@cols=1 \else
   \toks0=\expandafter{\temp}%
   \edef\temp{\noexpand\multicolumn{\XML@cols}{c}{\the\toks0}}%
  \fi
  \expandafter\gaddto@hook\expandafter\jg@row@toks\expandafter{\temp}}
```

3.3.6 Other commands

The `<hi>` element has a `rend` attribute that indicates how to typeset the content; we use `\csname` for dispatching.

```
\XMLelement{hi}
 {\XMLattribute{rend}{\XML@rend}{rm}}
 {\xmlgrab}
 {\csname rend@\XML@rend\endcsname{#1}}
```

In order to be complete, we should implement all font commands.

```
\let\rend@it\textit
\let\rend@bf\textbf
\let\rend@rm\textrm
\def\rend@small#1{{\small #1}}
\def\rend@sub#1{\textsubscript{#1}}
\let\rend@sup\textsuperscriptscript
```

We want TeX and LaTeX to typeset properly.

```
\XMLelement{TeX}
  {}{\TeX{}}{}

\XMLelement{LaTeX}
  {}{\LaTeX{}}{}
```

We use this for producing little images.

```
\XMLelement{preview}
  {}
```

```
  {\begin{preview}}
  {\end{preview}}
```

Some attribute definitions.

```
\XMLstring\att@true<>true</>
\XMLstring\att@false<>false</>
\XMLstring\jg@OverBrace<>&#xFE37;</>
\XMLstring\jg@UnderBrace<>&#xFE38;</>
\XMLstring\jg@OverBar<>&#xAF;</>
\XMLstring\jg@UnderBAr<>&#x332;</>
\XMLstring\att@mtd@left<>left</>
\XMLstring\att@mtd@right<>right</>
\XMLstring\att@mtd@center<>center</>
\XMLstring\att@none<>none</>
\XMLstring\att@dzero<>0</>
\XMLstring\att@done<>1</>
\XMLstring\att@dtwo<>2</>
\XMLstring\att@mathml@rm<>rm</>
\XMLstring\att@BLOCK<>block</>
\XMLstring\att@PREFIX<>prefix</>
\XMLstring\att@EQUATION<>equation</>
\XMLstring\att@sub<>sub</>
```

Some commands must be redefined at start of document. Note: the Unicode character 3F5 is 'greek unate epsilon symbol', we translate it as ϵ, the character 3B5 'greek small letter epsilon' is translated as ε.

```
\AtBeginDocument{
  \UnicodeCharacter{x332}{\jgunderline}
  \UnicodeCharacter{x3F5}{\epsilon}
  \UnicodeCharacter{x3B5}{\varepsilon}
  \let\downslopeellipsis\ddots
  \mathchardef\Rightarrow="3229
  \let\@texttop\ra@useatxy
  \let\XURL\relax
  \selectlanguage{english}
  \let\@item\jg@item
}
```

3.4 The fotex.cfg file

The fotex.cfg file contains the definitions of some elements. Two of them are used by the Raweb. The action associated to these elements is shown above.

```
\XMLelement{fo:RATHEME}
  {}
  {\xmlgrab}
  {\foratheme{#1}}

\XMLelement{fo:INRIA}
  {\XMLattribute{year}{\ra@@year}{2001}}
  {}
  {\xdef\ra@year{\ra@@year}\foinria}
```

Chapter 4

Interpreting XSL/Format in TeX

In general, an XML file describing a document like the Raweb contains instructions like: insert the table of contents here, or emphasize this word, but gives no details. In the next chapter we shall explain how these details can be added (using something like XSL/Transformations); here we explain how TeX can interpret these formatting commands. We assume that our document conforms to XSL (Extensible Style Sheet), which is a W3C recommendation of 15 October 2001. The important part of the recommendation is chapter 6 (Formatting objects) that describes all XML elements; they are in some namespace, conventionally abbreviated to '`fo`'.

We start with an example; it contains one line of the table of contents, associated to some section, here it is named "National Actions" and has a unique identifier, number 120. The style sheet that creates this piece of code added the section number 8.2, but the page number is still unknown: TeX must compute it. Some other information is added, for instance the color of the link, the font of the text, various dimensions, the type of the leaders.

Normally, such details do not appear in a TeX file, but are defined by a class. Since everything is explicit, any class could be used, and we shall see later that the article class is used; however your document can have a front matter, main matter, etc., as in the book class; they are associated to page sequences and page masters, which are rather complicated concepts.

```
<fo:block font-weight="bold" text-indent="0.25in">8.2.  
  <fo:inline space-start="20pt">
    <fo:inline>National Actions</fo:inline>
  </fo:inline>
  <fo:leader rule-style="dotted" rule-thickness="0pt"/>
  <fo:inline color="red">
     <fo:basic-link internal-destination="uid120">
        <fo:page-number-citation ref-id="uid120"/>
     </fo:basic-link>
  </fo:inline>
</fo:block>
```

4.1 Generalities

We describe here the content of two files: fotex.xmt and fotex.sty. The fotex.xmt file contains definitions of elements, while the fotex.sty file contains some commands. Both files start like this

```
1 % Copyright 2003 Sebastian Rahtz/Oxford University
2 %      <sebastian.rahtz@oucs.ox.ac.uk>
```

```
%
% Permission is hereby granted, free of charge, to any person obtaining
% a copy of this software and any associated documentation files (the
% ``Software''), to deal in the Software without restriction, including
% without limitation the rights to use, copy, modify, merge, publish,
% distribute, sublicense, and/or sell copies of the Software, and to
% permit persons to whom the Software is furnished to do so, subject to
% the following conditions:
% The above copyright notice and this permission notice shall be included
% in all copies or substantial portions of the Software.
```

Lots of people are working on these files:

```
% Includes fixes from Tomas Bures <ghort@pauline.vellum.cz>
%                     Yura Zotov <yznews@hotbox.ru>
%                     Anton V. Boyarshinov <boyarsh@ru.echo.fr>
%                     Dirk Roorda <dirk.roorda@planet.nl>
```

In the first version of this document, we discussed version 1.17, here it is version 1.25 (distributed by teTeX3.0). Newer versions might exist. In what follows, version V1 refers to the first version of the document, hence to version 1.17 of the fotex package, and V2 refers to 1.25.

```
\ProvidesPackage{fotex}[2003/03/10: version 1.25.
   support for XSL formatting, S Rahtz]
```

This file was modified by José Grimm in some places, we shall see why later on, this line gives the local revision number (this corresponds to the version dated 2006/11/23).

```
\immediate\write20{fotex.sty version 1.25. S Rahtz
 (JG patches: $ Revision: 1.13 $)}
```

We shall indicate the differences between these two versions whenever appropriate. In some cases, local assignments were replaced by global ones; as a general rule, if `<a>` and `<b>` are children of `<c>`, then typesetting of `<b>` is independent of typesetting of `<a>`, and a groups is created for every element; hence, if side effects are needed while typesetting `<a>` they should be local, unless required by `<b>`, case where global assignments are needed; if moreover the scope is restricted to `<c>`, a local copy is needed. In some other cases, `\relax` tokens were added at the end of assignments. Remember that, if TEX sees `\foo=0`, it expands the following token, in order to see if additional digits are found. Since `\relax` is a non-expandable token that produces no text, it can be put after the zero. In most of the cases, the space inserted at the end of the line is enough.

The first difference is the following.

```
\gdef\XML@attrib@x#1#2{
    \gdef\XML@tempb##1#1##2##3##4\relax\relax{
        \global\let\inheritexplicit\relax
        {\set@display@protect
        \expandafter\ifx\csname#2\endcsname\inherit
                \global\let\inheritexplicit\noexpand
        \fi}%
        \ifx\inheritexplicit\relax
                \def##2{#2}%
        \fi
    ##1##4\relax\relax}}
```

As explained in Chapter 2, putting a `\def` in a `\def` in a `\def` is not usual. In our case, the inner definition is conditional, so that it is unclear what happens. The outer command takes two arguments, say A and B, and defines another command, say C. The command C takes as argument a long list, and extracts expression A, followed by two expressions D and E; the remaining part

of the list is evaluated again. Quantity E is ignored, and D is defined to be B. Assume that we evaluate an element, defined by `\XMLelement`, and the first argument says that attribute X should be put in command Y with default value Z. Then D is Y, A is related to X, and B is the value of the attribute or a default value. The original code sets D. The modified code sets it conditionally, if the first `\ifx` is false. In fact, the test is done in a group, and if the test is true, a variable is globally changed, and the second `\ifx` looks at this change. The test is true in case B is '`inherit`', but instead of comparing token lists, the code converts B in a command via `\csname` and compares command names. This can produce a lot of commands and might overflow the hash table. Commands produced by `\csname` are never undefined, but outside the group, the old value is restored.

Second addition:

```
\gdef\explicitattribute#1{
{ \expandafter\def\csname#1-test\endcsname{\global\def\isexplicit{1}}
  \global\let\isexplicit\relax
  \def\XML@doattribute##1##2##3{\csname##2-test\endcsname}
  \the\XML@attribute@toks}}
```

Write `\do` instead of `\XML@doattribute` and `\L` instead of `\XML@attribute@toks`. The quantity `\L` is a token list containing all items of the form of a `\do` followed by a namespace A, a local name B and a value C. This code takes an argument, say 'foo', and sets `\isexplicit` to true (i.e., `\non-relax`) if one B is 'foo'. The idea is simple: the `\do` command is redefined in such a way as to modify `\isexplicit` if B is 'foo'. Implementation is horrible: the package defines `\foo-test` to change the flag, and evaluates the command named B-test for every B.

In Chapter 2, we gave the example of an element with two attributes, 'foo:bar' and 'color', with values 'gee' and 'red'. The first modification creates commands `\gee` and `\red`, and compares them to `\inherit`. The second modification creates commands `\bar-test` and `\color-test`, and evaluates them; the assumption is that these commands are not defined (made `\relax` by `\csname`), hence provoke no undesired result.

In one of our examples the code fails because the attribute is something like '`&#2016;`', corresponding to the math construct `\left\Vert`. The second command always gives a negative result because applied in the case of an empty attribute list. One way to patch the first command is to redefine the ampersand; but the code shown here is safer.

```
\def\inheritname{inherit}
\def\jgXML@attrib@x#1#2{
    \gdef\XML@tempb##1#1##2##3##4\relax\relax{
       \def\tmp{#2}%
        \ifx\tmp\inheritname\else\def##2{#2}%
        \fi
    ##1##4\relax\relax}}
```

Patch of the second command. We put '`foo`' in `\tmp`, B in `\atmp`. In case of equality, we globally set `\isexplicit` to `\empty` (this should be different from relax).

```
\def\explicitattribute#1{{%
   \def\tmp{#1}
   \global\let\isexplicit\relax
   \def\XML@doattribute##1##2##3{%
      \def\atmp{##2}\ifx\tmp\atmp\global\let\isexplicit\empty\fi}
    \the\XML@attribute@toks}}
```

The fotex-add.sty file loads the following packages: graphics, multicol, rotating, curves, soul, array, amsmath, longtable, url, ulem, color, times, mlnames, unicode, marvosym, ipa, ifsym, ucharacters, nameref, hyperref, and raweb-uni. This last package defines (or redefines) some Unicode characters. Some packages, like ifsym, were not in the original style file, they were added because

providing access to some fonts. Setting the boolean `hyperref` to false via `\hyperreffalse` inhibits loading of the hyperref package.

The fotex-add.sty loads fotex-supp.tex if such a file exists. This file can load some additional packages; it can also inhibit loading of the package hyperref (in fact, it is the only place where you can use `\hyperreffalse`). This mechanism is a bit complicated, because all packages must be loaded before the start of the document, and you cannot unload a package. Both files fotex.xmt and fotex.sty had to be adapted for the Raweb.

```
50 \RequirePackage{fotex-add}
```

The fotex.xmt file starts with these two lines, it is followed by declarations of attributes, strings and elements. The namespace number of `fo` is 5 (see below), that of `fotex` is 6.

```
51 \DeclareNamespace{fotex}{http://www.tug.org/fotex}
52 \DeclareNamespace{fo}{http://www.w3.org/1999/XSL/Format}
```

4.2 The root

Handling the `<fo:root>` element is trivial, but some magic is implied. Let's describe it. The main TEX file should define the `\xmlfile` command to be the name of the XML file to process, and input the xmltex.tex file. The last action of this file is to input xmltex.cfg, `\jobname`.cfg and the XML file. The file xmltex.cfg typically defines a catalog similar to the one shown in section 2.7. It defines the XSL/Format namespace (it assigns the number 5 to it) and says that the definitions are to be found in the fotex.xmt file. It may define other things. The `\jobname`.cfg file can also define some commands.

The `\xmlfile` file is read using XML syntax. It typically starts with a `<?xml?>` declaration (that defines the encoding, say UTF-8). In the case of the Raweb, it is followed by a `<fo:root>` element. This one contains a `<fo:layout-master-set>`, followed by some `<fo:static-content>` elements and some `<fo:page-sequence>` elements. The root element has three attributes, of the form xmlns:XX = 'YY'. For each attribute, the code line 388 (page 21) is executed. One of the attributes has the name xmlns:fo and its value is the XSL/Format URI. This URI is the same as the declaration above (on line 51 above, in this chapter), so that the 'fo' in `<fo:root>` is the same namespace in the XML file and the xmt file. After all attributes have been read, the code on line 413 is executed. In particular, the 'fo' in `<fo:root>` is evaluated now (line 416, command `\XML@ns`, the value is 5). The command `\XML@checkknown` has as side effect to call `\inputonce` with, as argument, the file fotex.xmt, as explained in the catalogue. This has as effect to load the fotex.xmt file. After that, we know how to handle the root element. The first action is to evaluate the attributes. Almost all attributes are global. The norm[1] says: there is one attribute media-usage, in what follows, we shall assume that its value is 'paginate'. The action is as follows: we define the documentclass to something non-trivial but not too complicated, and load the fotex package.

```
53 \XMLelement{fo:root}
54   {\XMLNSAX{fo}{fotex:spacing-style}{\FoTeXSpacingStyle}{normal}}
55   {\documentclass{article}
56    \usepackage{fotex}
```

We are now ready to evaluate everything. We start with `\begin{document}` plus some action (empty pagestyle, default language). When we see `</fo:root>` we evaluate `\end{document}`: this will stop everything.

```
57    \begin{document}
58    \pagestyle{empty}
59    \FOSetHyphenation
```

[1] URL is http://www.w3.org/TR/xsl/slice6.html, see also slices 1 to 7.

```
    \FoTeXSetSpacingStyle
    }
  {\end{document}}
```

Bootstrap code: We use here `\XMLNSA` as a short name for `\XMLnamespaceattribute` (and the same for `\XMLNSAX`). Remember that this takes four arguments. In the line that follows, the effect is the following: for every element in the 'fo' namespace, the value of the language trait is stored in `\FOlanguage`; the default value is '`\inherit`', this means that it is the same as the value of the father. The initial value is 'none'. In the case of the hyphenate trait, the initial value is 'false', the default value is '`\inherit`'. Note: the mechanism works only if attributes are declared before elements; the fotex.xmt file starts with the list of all attributes, in alphabetic order; we prefer give the definition after first use.

```
\XMLNSAX{fo}{language}{\FOlanguage}{\inherit}
\XMLNSAX{fo}{hyphenate}{\FOhyphenate}{\inherit}
\gdef\FOlanguage{none}
\gdef\FOhyphenate{false}
\def\LastLanguage{(undefined)}
\selectlanguage{english}
```

This is an addition of V2. If fotex:spacing-style is 'french' then `\frenchspacing` will be executed.

```
\def\FoTeXSetSpacingStyle{%
  \ifx\FoTeXSpacingStyle\att@french \frenchspacing \fi
}
\XMLstringX\att@french<>french</>
```

In the case where hyphenation is desired, this piece of code is assumed to switch to the right language, it sets `\hyphenpenalty` to some value (typically 50), otherwise to infinity.

```
\def\FOSetHyphenation{%
  \ifx\FOhyphenate\att@true
     \LoadLanguage{\FOlanguage}%
     \hyphenpenalty=\exhyphenpenalty
  \else
     \hyphenpenalty=10000
  \fi}
```

We put in `\newL` a canonical version of the language, say 'FR', then evaluate `\L@FR`, unless the language is the same. We remember in `\LastLanguage` the language. The file mlnames.sty defines a lot of languages[2].

```
\def\LoadLanguage#1{%
  \begingroup\utfeight@protect@chars\xdef\newL{#1}\endgroup
  \ifx\newL\LastLanguage
  \else
    \csname L@\newL\endcsname
  \fi
  \edef\LastLanguage{\newL}}
```

4.3 Mathematics

The elements defined in this paragraph are not defined by XSL/Format. In fact, they are not used by the Raweb.

[2]The English language is selected by: GB, US, gb, us, en, and none. But not EN. New in V2: en_GB and en_US.

Typesetting of <fotex:inlinemath>*math*</fotex:inlinemath>. We evaluate \(*math*\).

```
\XMLelement{fotex:inlinemath}
   {} {\(}  {\)}
```

Typesetting of <fotex:equation>*math*</fotex:equation>. The evaluation of the *math* is in an equation environment.

```
\XMLelement{fotex:equation}
   {}
   {\begin{equation}}
   {\end{equation}}
```

Typesetting of <fotex:displaymath> *math* </fotex:displaymath>. The evaluation of the *math* is a displaymath environment.

```
\XMLelement{fotex:displaymath}
   {}
   {\begin{displaymath}}
   {\end{displaymath}}
```

Typesetting of <fotex:eqnarray>*math*</fotex:eqnarray>. The evaluation of the *math* is a gather* environment. This is an amsmath environment, that requires an explicit \end tag, so that we need to grab it.

```
\XMLelement{fotex:eqnarray}
 {}
 {\xmlgrab}
 {\begin{gather*}#1\end{gather*}}
```

Handling of <fotex:subeqn>*math*</fotex:subeqn>. There is some action, that consists of evaluating the math, and a double backslash; this end-of-row marker must be executed outside the group defined by \xmlgrab, thus the \aftergroup. If there is a label (i.e. an id attribute in \FOid), we execute the \label command[3], otherwise, the equation will have no number.

```
\XMLelement{fotex:subeqn}
  {}
  {\xmlgrab}
  {%\ifx\FOid\@empty
     \gdef\w@t{#1\nonumber\\}
   %\else
   %  \gdef\w@t{#1\label{\temp}\\}
   %\fi
   \aftergroup\w@t}
```

4.4 Multiple columns

The passive tex package provides the nomulticol.sty file. It contains the following comments:

```
%% This is file `nomulticol.sty',
%% a tweak in package multicol.sty [2000/07/10 v1.5z
%%     multicolumn formatting (FMi)]
%% Tweaked by Dirk Roorda 2003/01/09
```

The purpose is to have the \begin{multicols} ... \end{multicols} functionality without putting the material inside a group. This is needed because in PassiveTeX a <fo:flow> is embedded in a multicols environment. But the <fo:block> with attribute span = 'all' must be able to interrupt this. However, saying \end{multicols} just before and \begin{multicols}{N} just after does

[3] I don't understand this \temp stuff. Moreover, as we shall see, we never use the \label command

not work, because it makes the attributes, set between the start of the flow and the beginning of the block, invisible. That's why a grouping-transparent multicol setup is needed. The package provides two macros `\nobeginmulticols` and `\noendmulticols` that do essentially the same but do not create a group.

Initial version of the fotex package did not implement the span feature, hence loaded the multicol package like this.

```
\IfFileExists{multicol.sty}
  {\RequirePackage{multicol}[1997/12/16]}
  {\newenvironment{multicols}[1]%
  {\typeout{Warning,  at line \the\inputlineno,
    multicol package not available}}{}%
}
```

The current version tries to load this new package if possible, the old one otherwise.

```
\IfFileExists{nomulticol.sty}
  {\confirmnomulticols}
  {\IfFileExists{multicol.sty}
    {\warnnomulticols}
    {\warnmulticols}
}
```

If the nomulticol package is not available, we define the five commands provided by the package.

```
\def\fakenomulticols{
 \def\nobeginmulticols##1{\begin{multicols}{##1}}
 \def\noendmulticols{\end{multicols}}
 \def\interbeginmulticols##1{}
 \let\interendmulticols\relax
 \let\refreshmulticols\relax
}
```

This is the action in case where the new package exists, or the old one exists, or none is found.

```
\def\confirmnomulticols{
  \RequirePackage{nomulticol}[2003/01/09]
  \typeout{INFO (nomulticol.sty:  fo:block span="all" works}
}
\def\warnnomulticols{
  \RequirePackage{multicol}[1997/12/16]
  \typeout{WARNING (multicol.sty:  fo:block span="all" does not work}
  \fakenomulticols
}
\def\warnmulticols{
  \typeout{WARNING (no multicol.sty:  multiple columns not available}
  \newenvironment{multicols}[1]{\typeout{Warning,  at line
       \the\inputlineno, multicol package not available}}{}
  \fakenomulticols
}
```

4.5 Page masters

The children of `<fo:root>` are: a `<fo:layout-master-set>`, an optional `<fo:declarations>`, and some `<fo:page-sequence>`. The `<fo:declarations>` has a sequence of `<fo:color-profile>` elements as children, that define color profiles. These are currently unimplemented.

```
\XMLelement{fo:color-profile}{}{}{}
\XMLelement{fo:declarations}{}{}{}
```

The children of `<fo:layout-master-set>` are either `<fo:simple-page-master>` (that define the layout of a page) or `<fo:page-sequence-master>` (that define which simple page masters are to be used). We define in this section how simple-page-masters are handled. Each such object has a name, which is in the master-name trait. For the Raweb, we define eight such masters, named: simple1, left1, right1, first1, simple2, left2, right2, first2. Here the digit at the end indicates the number of columns of text on the page. The names left and right stand for even or odd pages (odd pages are on the right, even pages on the left). We have a special case for simple ('blank' pages) and first pages. Each master has a reference-orientation. This will be ignored; it has also a writing-mode, that will be ignored. We assume that it is 'lr-tb', meaning: inline components and text within a line are written left-to-right; lines and blocks are placed top-to-bottom. With this convention, we have before=top, after=bottom, start=left and end=right. Schematically this can be represented like this:

	before				top	
start	body	end	→	left	body	right
	after				bottom	

The page is divided into five rectangular regions, four of them are called region-before, region-after, region-start and region-end (they correspond to the header, the footer, left margin, right margin). The content of these is 'static', said otherwise, it is the same for each page, except that it can contain the current page number. These regions have an extent (vertically, or horizontally). Whether the upper-left corner is part of the region-before or region-start depend on the precedence of the region-before. The four regions mentioned above surround the region-body. Both the page and the region-body have margins. How these margins and extents define the size of the body is unclear to me. For instance, the Raweb defines one page master with the following attributes: master-name = 'left1', page-width = '210mm', page-height = '297mm', margin-top = '75pt', margin-bottom = '100pt', margin-left = '80pt', margin-right = '80pt'. With these values the text width is near 15cm, and the text height is 21cm (in fact, after the text, we have some space, an empty footer, and the margin, i.e. 24pt, 12pt and 100pt, a total of nearly 5cm).

We declare here some attributes (margins are declared later).

```
\XMLNSA{fo}{page-width}{\FOpagewidth}{auto}
\XMLNSA{fo}{page-height}{\FOpageheight}{auto}
\XMLNSAX{fo}{master-name}{\FOmastername}{}
\XMLNSAX{fo}{extent}{\FOextent}{0.0pt}
```

Action is trivial: we just evaluate the content.

```
\XMLelement{fo:layout-master-set}
  {} {} {}
```

This will be used twice. The idea is that typesetting a page uses "traits", that are computed from attributes; all four margins define a single trait, that can be defined by a single attribute (margin) or a sequence of four attributes (for instance margin-right), or can be inherited. We shall see later that the equivalent of TEX glue is a complicated trait.

```
\def\jg@expandmargins{%
    \ifx\FOmargin\@empty\relax\else
      \let\FOmarginright\FOmargin
      \let\FOmarginleft\FOmargin
      \let\FOmargintop\FOmargin
      \let\FOmarginbottom\FOmargin
     \fi}
```

The code for `<fo:simple-page-master>` has a comment that says *tests removed see above*. In V1, there were four lines of code shown here. These lines of code are now removed, so that our remark has to be commented out too: *As mentioned above, the sum of the vertical margins are used, so that it is unclear why one variable is replaced by the maximum. In our cases, outer margins are larger than inner margins, so that the tests are false, and no variable is changed.*

```
\ifdim\InnerTopMargin>\FOmargintop\def\FOmargintop{\InnerTopMargin}\fi
\ifdim\InnerBottomMargin>\FOmarginbottom\def\FOmarginbottom{\InnerBottomMargin}\fi
\ifdim\InnerRightMargin>\FOmarginright\def\FOmarginright{\InnerRightMargin}\fi
\ifdim\InnerLeftMargin>\FOmarginleft\def\FOmarginleft{\InnerLeftMargin}\fi
```

This defines `\Atomic:left1` (assuming that the master name is 'left1'), it contains the size of the page and the max values of the margins of the page and body. This is the old version.

```
\def\jg@define@atomic{
    \expandafter\xdef\csname Atomic:\FOMaster\endcsname{
     \MasterTopMargin\FOmargintop
     \MasterBottomMargin\FOmarginbottom
     \MasterRightMargin\FOmarginright
     \MasterLeftMargin\FOmarginleft
     \paperwidth\FOpagewidth
     \paperheight\FOpageheight}}
```

Assume that the master name is 'left1', this defines `\Atomic:left1`; when executed, this defines some commands to contain the sum of outer and inner values; the outer value is the value of the attribute (for instance margin-top) of the current element (the simple-page-master), the inner value is the value of the same attribute of the region-body child of the master. Quantities used in this code are defined later; the body of the macro contains ten lines of the form AXYR, where A is an optional `\advance`, R is an optional `\relax`, X is a dimension register, and Y a macro (definitions will be given later). The effect of the code is to store Y in X, or increment X by Y. Since the command is defined by `\xdef`, all possible tokens are expanded. Only these Y tokens can be expanded.

```
\def\jg@define@atomic{
 \expandafter\xdef\csname Atomic:\FOMaster\endcsname{
  \MasterTopMargin\FOmargintop
  \advance\MasterTopMargin\InnerTopMargin\relax
  \MasterBottomMargin\FOmarginbottom
  \advance\MasterBottomMargin\InnerBottomMargin\relax
  \MasterRightMargin\FOmarginright
  \advance\MasterRightMargin\InnerRightMargin\relax
  \MasterLeftMargin\FOmarginleft
  \advance\MasterLeftMargin\InnerLeftMargin\relax
  \paperwidth\FOpagewidth
  \paperheight\FOpageheight}}
```

In the example of the Raweb, `\Atomic:left1` contains the following.

```
\MasterTopMargin 75pt\advance \MasterTopMargin 24pt\relax
\MasterBottomMargin 100pt\advance \MasterBottomMargin 24pt\relax
\MasterRightMargin 80pt\advance \MasterRightMargin 0pt\relax
\MasterLeftMargin 80pt\advance \MasterLeftMargin 0pt\relax
\paperwidth 210mm
\paperheight 297mm.
```

The content of a `<fo:simple-page-master>` is a definition of each of the five regions mentioned above. In this version, we ignore the start and end regions: we assume that they are blank and have zero extent. Only region-body is mandatory, so that we provide a default for region-before

and region-after. After this element has been completely evaluated, six commands are defined: assume that the master name is 'left1'; we define `\left1:before`, `\left1:after`, to be names of regions, `\left1:before-extent`, `\left1:after-extent`, their extents; there is also `\left1:B` (instead of 'B', the name of the body region should be used, it is 'xsl-region-body', see below), and `\Atomic:left1`.

```
\XMLelement{fo:simple-page-master}
  {}
  {\let\FOMaster\FOmastername
    \jg@expandmargins
    \ifx\FOpagewidth\att@auto\edef\FOpagewidth{\paperwidth}\fi
    \ifx\FOpageheight\att@auto\edef\FOpageheight{\paperheight}\fi
    \expandafter\xdef\csname\FOMaster:after\endcsname{DummyRegion}
    \expandafter\xdef\csname\FOMaster:before\endcsname{DummyRegion}
    \expandafter\xdef\csname\FOMaster:before-extent\endcsname{\FOextent}
    \expandafter\xdef\csname\FOMaster:after-extent\endcsname{\FOextent}
    }
  {
  %% tests removed see above
    \begingroup
     \utfeight@protect@chars
     \jg@define@atomic
   \endgroup
   }
```

These lines are from fotex.sty. The quantities `\hoffset` and `\voffset` are the opposite of the TEX offsets. The default paper width is strange.

```
\paperwidth211mm
\paperheight297mm
\hoffset-1in
\voffset-1in
```

The `<fo:region-before>` element is empty, it has some traits. We only consider region-name and extent. In the example given above, attributes are region-name = 'xsl-region-before-left' and extent = '12pt'. Each region has a default name, it is not used here. The code defines two global commands. For instance, this could define `\left1:before` to the name shown above (in the case where the master name is 'simple1' or 'simple2', the default name is used).

```
\XMLelement{fo:region-before}
  {}
  {\ifx\FOregionname\@empty \def\FOregionname{xsl-region-before}\fi
   \begingroup
     \utfeight@protect@chars
     \expandafter\xdef\csname\FOMaster:before\endcsname{\FOregionname}
     \expandafter\xdef\csname\FOMaster:before-extent\endcsname{\FOextent}
   \endgroup
  }
  {}
```

Page footer. This is handled as above.

```
\XMLelement{fo:region-after}
  {}
  {\ifx\FOregionname\@empty \def\FOregionname{xsl-region-after}\fi
   \begingroup
     \utfeight@protect@chars
```

```
      \expandafter\xdef\csname\FOMaster:after\endcsname{\FOregionname}
      \expandafter\xdef\csname\FOMaster:after-extent\endcsname{\FOextent}
    \endgroup
  }
  {}
```

These are not implemented.

```
\XMLelement{fo:region-start}{}{}
\XMLelement{fo:region-end}{}{}
```

The `<fo:region-body>` is empty; it has traits as the region-before. In the example of 'left1' they are the following: margin-bottom = '24pt', margin-top = '24pt' (the default value for the two other margins is zero; no extent can be given here). The default value of the region-name is used, because all pages have the same layout.

There are possibly other attributes, but most of them are not implemented. As explained above, we store somewhere the margins so that the page-master can use them. We also store in `\left1:B` (where 'left1' is the name of the master, and B the name of the region), the four following quantities: top margin, bottom margin, column-count and column-gap: this is because the page can contain more than one column of text, and we have to specify this number and the distance between the columns.

```
\XMLelement{fo:region-body}
  {}
  {
  \jg@expandmargins
  \xdef\InnerBottomMargin{\FOmarginbottom}
  \xdef\InnerTopMargin{\FOmargintop}
  \xdef\InnerLeftMargin{\FOmarginleft}
  \xdef\InnerRightMargin{\FOmarginright}
  \ifx\FOregionname\@empty \def\FOregionname{xsl-region-body} \fi
  \begingroup
     \utfeight@protect@chars
      \expandafter\xdef\csname\FOMaster:\FOregionname\endcsname
        {\FOcolumngap|\FOcolumncount|\FOmarginbottom|\FOmargintop|}
  \endgroup
  }
 {}
```

This will be used later. It grabs the four values saved by the procedure above.

```
\def\Pass#1\\{\expandafter\@Pass#1}
\def\@Pass#1|#2|#3|#4|{%
 \columnsep=#1
 \def\NColumns{#2}%
 \def\Marginbottom{#3}%
 \def\Margintop{#4}%
 }
```

Some declarations.

```
\XMLNSAX{fo}{region-name}{\FOregionname}{}
\XMLNSAX{fo}{master-reference}{\FOmasterreference}{}
\XMLNSAX{fo}{column-gap}{\FOcolumngap}{12.0pt}
\XMLNSAX{fo}{column-count}{\FOcolumncount}{1}
\def\NColumns{1}
\gdef\PrevNColumns{1}
```

4.6 Page sequences

A <fo:page-sequence-master> is an element that has a single trait master-name; its content defines the page sequence with that name, according to its single child, that can be one of three possibilities. The Raweb defines the following: twoside1nofirst, twoside1, oneside1, twoside2, oneside2, but uses only the second one. The meaning is the following: all even pages are the same, as well as all odd pages; the first page is special. If you look very closely, there are three pages numbered 2. At least one of them is empty; but this is a kludge. In fact, the initial page (the title page) has no headings, it is of type First; is it followed by an empty page, generated by <cleardoublepage/>. This page is followed by a second page sequence that contains the table of contents; this page sequence is followed by the text; this is a page sequence that starts on a right page, numbered one. The first two page sequences should define force-page-count to be end-on-even, rather than using this kludge. The behavior will change in 2005.[4]

In all examples that follow, we shall assume that the master name is 'twoside1'; this quantity is saved in a variable that can be used by the children. The master can define a single page, in this case the child defines this page; it can define a sequence of pages that look all the same, in this case, the child gives a model for these pages; finally, it can define a sequence of pages, whose aspect depend on parameters, case where the child contains a list of definitions; for each of this definitions, the master is the grand-father, this explains why the name is stored in a variable named \Granpa.

We shall see later that the V2 implementation allows the use of more than one simple page, numbered one, two, three, etc. The counter \SimplePMRefs holds the index of the next simple page master.

```
\XMLelement{fo:page-sequence-master}
  {}
  {\global\SimplePMRefs0\relax
   \let\Granpa\FOmastername
  }
  {\global\SimplePMRefs0\relax}
```

We have an implementation problem: the FO specifications are too complex to match those of TEX, so that we use only 4 types of pages, named Even, Odd, First, Blank. The original fotex.sty did define but not use Blank. We changed this. In version V1, children of a page sequence master, say 'twoside1', were assumed to define the four commands \First:twoside1, \Blank:twoside1, \Odd:twoside1, \Even:twoside1, to be the name of a page master. In the case of a single, or repeatable page master reference, these four quantities were defined to be the master-reference attribute. In V2, we define only \Odd:twoside1 in the case of a repeatable page master reference, and we define \Lead:3:twoside1 if this is the third single page master reference in the list.

We define here <fo:single-page-master-reference>; this defines a single page. The non-trivial part here is that any Unicode character is allowed in the attribute value, and some protection is needed.

```
\XMLelement{fo:single-page-master-reference}
   {}
   {
    \global\advance\SimplePMRefs1\relax
    \begingroup
      \utfeight@protect@chars
      \expandafter\xdef\csname Lead:\the\SimplePMRefs:\Granpa\endcsname
             {\FOmasterreference}
    \endgroup
```

[4] The kludge is still active in 2006; let's hope it will be removed in 2007.

```
263     }
264    {}
```

We define here `<fo:repeatable-page-master-reference>`; this defines a sequence (of some length) of pages. The length is ignored. The action is as described above.

```
265 \XMLelement{fo:repeatable-page-master-reference}
266    {}
267    {
268     \begingroup
269       \utfeight@protect@chars
270       \expandafter\xdef\csname Odd:\Granpa\endcsname{\FOmasterreference}
271     \endgroup
272    }
273    {}
```

The content of `<fo:repeatable-page-master-alternatives>` is a sequence of conditional page master references. The recommendations specify how to chose them, but we use another algorithm.

```
274 \XMLelement{fo:repeatable-page-master-alternatives}
275    {} {} {}
```

The `<fo:conditional-page-master-reference>` may define one of `\First:twoside1`, etc., to the value of the master-reference-trait trait. This is the name of a page-master, for instance 'left1'. The conditions are defined by the following traits: page-position which can be 'first', 'last'[5], 'rest', or 'any' (here 'rest' means any page that is neither the first nor the last in a sequence); blank-or-not-blank can be 'blank', 'not-blank', or 'any' (a blank page is a forced page that contains no text); odd-or-even can be 'odd', 'even' or 'any'. The parity of the page number is considered (if there are two consecutive pages numbered one, they are both odd; thus, this is not the same as left-or-right). What we do here is to define one and only one command. For instance, we could define `\Even:twoside1` to be '`right1`'.

```
276 \XMLelement{fo:conditional-page-master-reference}
277    {}
278    {
279   \begingroup
280   \utfeight@protect@chars
281   \ifx\FOoddoreven\att@even
282       \expandafter\xdef\csname Even:\Granpa\endcsname{\FOmasterreference}
283   \else\ifx\FOoddoreven\att@odd
284       \expandafter\xdef\csname Odd:\Granpa\endcsname{\FOmasterreference}
285   \else \ifx\FOpageposition\att@first
286       \expandafter\xdef\csname First:\Granpa\endcsname{\FOmasterreference}
287   \else \ifx\FOblankornotblank\att@blank
288       \expandafter\xdef\csname Blank:\Granpa\endcsname{\FOmasterreference}
289   \else
290       \expandafter\xdef\csname Odd:\Granpa\endcsname{\FOmasterreference}
291   \fi\fi\fi\fi
292   \endgroup
293   }
294  {}
```

[5] Is it possible to know, in TeX, if a given page is the last in a sequence? Maybe, we could modify the `\output` routine, so that the `\clearpage` on line 372 will output some pages, the last one being special. The situation is complicated by the fact that lines 377 and 380 may output empty pages in the sequence. The `\clearpage` on line 389, together with the `\BlankPpage` on lines 393 and 396 are also candidates for producing the last page in the sequence.

Declarations of attributes used here.

```
\XMLNSAX{fo}{page-position}{\FOpageposition}{any}
\XMLNSAX{fo}{odd-or-even}{\FOoddoreven}{any}
\XMLNSAX{fo}{blank-or-not-blank}{\FOblankornotblank}{any}
\XMLstringX\att@even<>even</>
\XMLstringX\att@odd<>odd</>
\newcount\SimplePMRefs
```

A `<fo:page-sequence>` is linked via its master-reference trait to a page master or a page sequence master. Its children are some `<fo:static-content>` elements (they define the content of the regions defined by the page master, except for the region-body) and a `<fo:flow>` that defines the content of the pages. Some important attributes are: initial-page-number that indicates the number of the first page (it can be 'auto-even' or 'auto-odd', this means that the first page number should be even or odd; it may imply the use of a blank page). The value of force-page-count can be 'end-on-odd' or 'end-on-even' (this may insert a final blank page). The language can be defined for a page sequence, a block or a character. The action is to evaluate the language trait via `\FOSetHyphenation` and to put the others in variables. Some traits are unused: country, letter-value, grouping-separator, grouping-size, format[6]. In the case of the Raweb, the attributes of the main page sequence are format = '1', text-align = 'justify', hyphenate = 'true', language = 'en', initial-page-number = '1', master-name = 'twoside1'.

```
\XMLelement{fo:page-sequence}
  {}
  {\let\CurrentPageMaster\FOmasterreference
   \let\pendingID\FOid
   \let\PageNumber\FOinitialpagenumber
   \let\ForcePage\FOforcepagecount
   \FOSetHyphenation
   \LoadLanguage{\FOlanguage}
   }
  {}
```

Attributes.

```
\XMLNSAX{fo}{initial-page-number}{\FOinitialpagenumber}{auto}
\XMLNSAX{fo}{force-page-count}{\FOforcepagecount}{auto}
\XMLstringX\att@autoodd<>auto-odd</>
\XMLstringX\att@autoeven<>auto-even</>
\XMLstringX\att@endonodd<>end-on-odd</>
\XMLstringX\att@endoneven<>end-on-even</>
```

You can say `<fo:static-content>`, with an attribute flow-name whose value is something like xsl-region-before-right. This is the region-name of some page master (used by the current page sequence). Logically, given the name, it should define the header of odd pages. The content is a sequence of blocks, that will be typeset later (on one or more pages). The action is to call a command defined below. A priori, the result should be used only by the flow that is a sibling of this element, but this is hard to do in TEX.

```
\XMLelement{fo:static-content}
 {}
 {\xmlgrab}
 {\protectCS\FOflowname
  \FOSetStatic{#1}{\FOflowname}}
```

[6] This last one explains how page numbers are typeset; see later.

We show here a function `\FOSetStatic` that takes two arguments, the body of the static content, and its flow-name attribute. This code is wrong. Instead of `\noexpand#1`, there should be something that inserts the first argument, without expansion (this could be achieved by putting the argument in a token list, and using `\the`). The actual code uses `\gdef` instead of `\xdef` and `\expandafter` for each token that must be expanded.

```
\def\FOSetStatic#1#2{%
   \expandafter\xdef\csname Static:\FOflowname\endcsname
    {{{
     \def\noexpand\XML@parent{}
     \global\noexpand\FOinOutputtrue
       \def\noexpand\FOwhitespacecollapse{true}%
       \def\noexpand\FOwrapoption{wrap}%
       \def\noexpand\FOtextalign{start}%
       \def\noexpand\FOfontfamily{\FOfontfamily}%
       \def\noexpand\FOfontsize{\FOfontsize}%
       \def\noexpand\FOfontstretch{\FOfontstretch}%
       \def\noexpand\FOfontvariant{\FOfontvarian}%
       \def\noexpand\FOfontweight{\FOfontweight}%
       %\ignore{\FOtextindent}
       \def\noexpand\FOfontstyle{\FOfontstyle}
       \noexpand#1
    \global\noexpand\FOinOutputfalse
   }}}%
   \jg@hack@foot}
```

The action associated to `<fo:static-content>` is to grab some parameters. We have a sequence of commands, each of them reads and expands a quantity, and passes it to the next one. Assume that `\foo` expands to `bar`. Then `bar\\` is added to the input stream. This is re-read, and converted to `{bar}`. This could be done directly, and left in stream until used by the final function. Said otherwise, the line shown here has the same effect and is more efficient.

```
\def\@@SetStatic{\expandafter\@@@SetStatic\expandafter{\FOfontweight}}
```

That's the real code.

```
328 \def\FOSetStatic{\expandafter\@SetStatic\FOtextindent\\}
329 \def\@SetStatic#1\\{\expandafter\@@SetStatic\FOfontsize\\{#1}}
330 \def\@@SetStatic#1\\#2{\expandafter\@@@SetStatic\FOfontweight\\{#1}{#2}}
331 \def\@@@SetStatic#1\\#2#3{\expandafter
332     \@@@@SetStatic\FOfontvariant\\{#1}{#2}{#3}}
333 \def\@@@@SetStatic#1\\#2#3#4{\expandafter
334    \@@@@@SetStatic\FOfontstyle\\{#1}{#2}{#3}{#4}}
335 \def\@@@@@SetStatic#1\\#2#3#4#5{\expandafter
336    \@@@@@@SetStatic\FOfontstretch\\{#1}{#2}{#3}{#4}{#5}}
337 \def\@@@@@@SetStatic#1\\#2#3#4#5#6{\expandafter
338     \@@@@@@@SetStatic\FOfontfamily\\{#1}{#2}{#3}{#4}{#5}{#6}}
```

This code globally defines `\Static:xsl-region-before-right` (or a command like this) whose effect is to execute the content of the `<fo:static-content>` in a TeX group, where some parameters are defined. These parameters are: font family, font stretch, font style, font variant, font weight, font size, textindent (unused, but explicitly set to zero in the main block of the static content), (arguments `#1` to `#7`). Argument `#8` is the content, argument `#9` is the name. The switch `\ifFOinOutput` is set to true while evaluating the content. The `\XML@parent` command is locally set to empty. See explanations later.

```
\def\@@@@@@@SetStatic#1\\#2#3#4#5#6#7#8#9{%
   \expandafter\gdef\csname Static:#9\endcsname{%
     {%
      {\def\XML@parent{}\global\FOinOutputtrue
       \def\FOwhitespacecollapse{true}%
       \def\FOwrapoption{wrap}%
       \def\FOtextalign{start}%
       \def\FOfontfamily{#1}%
       \def\FOfontsize{#6}%
       \def\FOfontstretch{#2}%
       \def\FOfontvariant{#4}%
       \def\FOfontweight{#5}%
       \def\FOfontstyle{#3}#8\global\FOinOutputfalse}}%
   \jg@hack@foot{#9}
}
```

Assume that `\Static:xsl-footnote-separator` was just defined. Said otherwise, we have defined a static region for the footnote separator. We store in `\footnoterulepre` the name of this command, and execute it. This should produce the footnote rule. The dimension of the resulting box is remembered in `\skip\footins`, a quantity that defaults to `\bigskipamount`. We set `\footnotesep` to zero. This quantity is defined by the class file (to 6.6pt for a ten point article). Finally, we define `\footnoterule` to use this funny command in a vbox of height zero. The `\vfill` is strange.

```
\def\jg@hack@foot#1{
  \ifx\FOflowname\att@xsl@footnote@separator\relax
    \xdef\footnoterulepre{Static:#1}%
    \global\footnotesep\z@
    \setbox\@tempboxa\vbox{\csname\footnoterulepre\endcsname}%
    \@tempdima=\z@
    \advance\@tempdima\ht\@tempboxa
    \advance\@tempdima\dp\@tempboxa
    \global\skip\footins\@tempdima\relax
    \gdef\footnoterule{\vfill\vbox to\z@{
       \vss\csname\footnoterulepre\endcsname}}%
  \fi
}
```

We define here the action associated to `DummyRegion`, it is empty.

```
\expandafter\def\csname Static:DummyRegion\endcsname{}
\XMLNSAX{fo}{flow-name}{\FOflowname}{}
```

4.7 Flows

Changes in V2: the command `\setaccordingtomaster` was added; it contains a very long (one hundred lines) and complicated piece of code, moved from fotex.xmt to fotex.sty. We have split this code into smaller pieces, this is the only way to understand what happens. Another change: a great number of assignments are made global, replacing `\def` by `\gdef`, `\edef` by `\xdef`, or adding a `\global` prefix. The command `\BlankPage` not only creates a blank page, but also ships it out. Finally, handling of multiple columns per page has changed.

The content of the `<fo:flow>` formatting object is a sequence of blocks that defines a sequence of pages. The action associated is a bit complicated; for this reason, we shall define some pseudo functions instead of the original big code. Let's recall that the `<fo:page-sequence>` element

remembers in \pendingID the current id and in \PageNumber the value of the initial page number. What we do here is to evaluate these instructions. We start with \clearpage, this makes sure that all pages are shipped out; this is not needed in the case where there are more than one column per page, see below. In the case where the desired page number is 'auto' we are done; if it is 'auto-odd' or 'auto-even', we emit an empty page, if needed, so as to make sure that the page number has the given parity. Finally, the page number can be an integer, in this case, we set the page counter. From now one, the page number is correct, so that we can insert the label of the parent. It is however too early to typeset something, for instance, \textwidth is still random. Changes in V2: \BlankPage used instead of \hbox{}\newpage.

```
\def\jg@flowI{%
  \ifnum\PrevNColumns>1\relax
  \else
   \clearpage
  \fi
  \ifx\PageNumber\att@auto
  \else
     \ifx\PageNumber\att@autoeven
        \ifodd\c@page\BlankPage\fi
     \else
        \ifx\PageNumber\att@autoodd
          \ifodd\c@page\else\BlankPage\fi
        \else
          \setcounter{page}{\PageNumber}%
        \fi
     \fi
  \fi
  \let\FOid\pendingID \FOlabel\global\let\pendingID\@empty % hacked jg
}
```

After evaluation of the children of the flow, we terminate with a \clearpage. We look at the force-page-count trait. If this imposes that the last page be even or odd, we may emit a blank page (note that the page counter holds one more than the last page number).

```
\def\jg@flowV{%
   \clearpage
   \ifx\ForcePage\att@auto
   \else
     \ifx\ForcePage\att@endoneven
       \ifodd\c@page\else\BlankPage\fi
     \else
        \ifx\ForcePage\att@endonodd
          \ifodd\c@page\BlankPage\fi
        \fi
     \fi
   \fi
}
```

We define now four quantities \PEven, \POdd, \PBlank and \PFirst according to the master-reference; this can be the name of a page-sequence-master or a page-master. In the second case, we put this name into all four commands[7]. Otherwise, the reference is something like 'twoside1' and the command \Odd:twoside1 is a pagemaster. We put this value in \POdd. We do the same for the other commands. However, if \Peven is undefined, we use the value of \POdd instead.

[7] code shown here as '...', it is a bit more complicated in V2

The case of `\PFirst` is a bit more complicated: in fact, if it is undefined, we look at the parity of the current page number. The code shown here is much simpler than the initial one, but has the same effects. Note: the code shown in the first version of the document was wrong; we have replaced it by a call to `\ifnotdefined`, that is a conditional macro that should evaluate to true if the argument comes from an undefined `\csname`. We shall give the real code below.

```
\@ifundefined{Atomic:\CurrentPageMaster}
 {
  \edef\PFirst{\csname First:\CurrentPageMaster\endcsname}
  \edef\PBlank{\csname Blank:\CurrentPageMaster\endcsname}
  \edef\PEven{\csname Even:\CurrentPageMaster\endcsname}
  \edef\POdd{\csname Odd:\CurrentPageMaster\endcsname}
  \ifnotdefined\PBlank\let\PBlank\POdd\fi
  \ifnotdefined\PEven\let\PEven\POdd\fi
  \ifnotdefined\PFirst
    \ifodd\c@page\let\PFirst\POdd \else\let\PFirst\PEven\fi\fi
}
{...}
```

In order to make the code easy to understand, we introduce the command `\Fdef`, and we can say `\Fdef{\POdd}{Odd}`. This defines `\POdd` as above. The next function computes `\PFirst`; we cannot say: if the page master does not define First, than use `\POdd` or `\Peven` because these commands are not yet defined. If the first page is even, the value of `\PFirst` depends on whether the page master defines Even; if undefined, we use the value for odd pages. We use `\Cdef` for such a conditional definition. There is a further complication. The value of the first page is defined by `\FOinitialpagenumber`. In the code shown above, we assumed that the page counter has already been set according to this value. In the code below, this is not necessarily true. Thus, we introduce `\Pdef` that sets its first argument depending on the parity of the second.

```
\def\Fdef#1#2{\xdef#1{\csname #2:\CurrentPageMaster\endcsname}
\def\Cdef#1#2#3{%
 \@ifundefined{#2:\CurrentPageMaster}
   {\Fdef{#1}{#3}}{\Fdef{#1}{#2}}%
}
\def\Pdef#1#2{%
   \ifodd#2 \Fdef{#1}{Odd}
   \else \Cdef{#1}{Even}{Odd}\fi
}

\def\jg@compute@Pfirst{%
 \@ifundefined{First:\CurrentPageMaster}
   {
    \ifx\FOinitialpagenumber\att@auto
      \Pdef\PFirst\c@page
    \else \ifx\FOinitialpagenumber\att@autoeven
      \Cdef{\PFirst}{Even}{Odd}
    \else \ifx\FOinitialpagenumber\att@autoodd
      \Fdef{\Pfirst}{Odd}
    \else
      \Pdef\Pfirst\FOinitialpagenumber
    \fi\fi\fi\fi
   }
   {\Fdef{\Pfirst}{First} }
}
```

This is the complete code, in the case of a page sequence master like 'twoside1'. A non trivial question is when to use `\PFirst`. The answer is when the boolean `@specialpage` is true. This is initially set to true, then to false after first use. A special case is when `\Lead:17:twoside1` is not defined, but could be used. In this case, the page is special if the number is one, non-special otherwise.

```
\def\jg@flowII@conditional{
 \ifnum\SimplePMRefs>1\relax\global\@specialpagefalse\fi
 \jg@compute@Pfirst
 \Cdef{\PBlank}{Blank}{Odd}
 \Cdef{\PEven}{Even}{Odd}
 \Fdef{\POdd}{Odd}
}
```

This is the code in case the reference is a page-master sequence. In the case `\Lead:17:twoside1` is defined, this is the name of the page to be used, whatever the value of the page number.

```
\def\jg@flowII{
 \@ifundefined{Lead:\the\SimplePMRefs:\CurrentPageMaster}
 { \jg@flowII@conditional }
 {
   \xdef\PFirst{\csname Lead:\the\SimplePMRefs:\CurrentPageMaster\endcsname}
   \global\let\POdd\PFirst
   \global\let\PEven\PFirst
   \global\let\PBlank\PFirst
 }
}
```

This is now the whole function. There are two cases to consider. The master-reference can be the name a page-master, for instance 'left1', case where `\Atomic:left1` is defined and we set `\First`, as well as all other quantities to '`left1`'. It can be a page-sequence-master, case handled above. Action associated to `\jg@flowIII` is described later.

```
\def\setaccordingtomaster{%
  \global\@specialpagetrue
  \@ifundefined{Atomic:\CurrentPageMaster}
   { \jg@flowII }
   {
    \global\let\PFirst\CurrentPageMaster
    \global\let\PBlank\CurrentPageMaster
    \global\let\POdd\CurrentPageMaster
    \global\let\PEven\CurrentPageMaster
   }
  \jg@flowIII
}
```

We use the `\PEven` command to define four commands, that contain the name of the header, footer, and their extent. The same is done for the other type of pages.

```
\def\jg@set@headings{
   \xdef\EvenHeadExtent{\csname\PEven:before-extent\endcsname}
   \xdef\EvenHead{Static:\csname\PEven:before\endcsname}
   \xdef\EvenTailExtent{\csname\PEven:after-extent\endcsname}
   \xdef\EvenTail{Static:\csname\PEven:after\endcsname}
   \xdef\FirstHeadExtent{\csname\PFirst:before-extent\endcsname}
   \xdef\FirstHead{Static:\csname\PFirst:before\endcsname}
   \xdef\FirstTailExtent{\csname\PFirst:after-extent\endcsname}
```

```
    \xdef\FirstTail{Static:\csname\PFirst:after\endcsname}
    \xdef\OddHeadExtent{\csname\POdd:before-extent\endcsname}
    \xdef\OddHead{Static:\csname\POdd:before\endcsname}
    \xdef\OddTailExtent{\csname\POdd:after-extent\endcsname}
    \xdef\OddTail{Static:\csname\POdd:after\endcsname}
    \xdef\BlankHeadExtent{\csname\PBlank:before-extent\endcsname}
    \xdef\BlankHead{Static:\csname\PBlank:before\endcsname}
    \xdef\BlankTailExtent{\csname\PBlank:after-extent\endcsname}
    \xdef\BlankTail{Static:\csname\PBlank:after\endcsname}
}
```

We define here a function that fetches some of these parameters. The value stored in `\themargin` will be defined later. It depends on the parity of the page number. This was not in the original fotex.sty.

```
\def\jg@use@blankpage{%
     \def\@thehead{\csname\BlankHead\endcsname}%
     \def\@thefoot{\csname\BlankTail\endcsname}%
      \ifodd\count\z@ \let\@themargin\oddsidemargin
      \else \let\@themargin\evensidemargin\fi
     \def\headheight{\BlankHeadExtent}%
     \def\tailheight{\BlankTailExtent}}%
```

We consider here three commands that handle the other cases. It is assumed that the first page is always odd.

```
\def\jg@use@specialpage{%
     \def\@thehead{\csname\FirstHead\endcsname}%
     \def\@thefoot{\csname\FirstTail\endcsname}%
     \let\@themargin\oddsidemargin
     \def\headheight{\FirstHeadExtent}%
     \def\tailheight{\FirstTailExtent}}%
```

Case of even pages.

```
\def\jg@use@evenpage{%
     \def\@thehead{\csname\EvenHead\endcsname}%
     \def\@thefoot{\csname\EvenTail\endcsname}%
     \let\@themargin\evensidemargin
     \def\headheight{\EvenHeadExtent}%
     \def\tailheight{\EvenTailExtent}}%
```

Case of odd pages.

```
\def\jg@use@oddpage{%
     \def\@thehead{\csname\OddHead\endcsname}%
     \def\@thefoot{\csname\OddTail\endcsname}%
     \let\@themargin\oddsidemargin
     \def\headheight{\OddHeadExtent}%
     \def\tailheight{\OddTailExtent}}%
```

This is how we select one of these four commands.

```
\newif\ifBlankPage
\def\jg@usepagestyle{%
 \ifBlankPage
    \jg@use@blankpage
 \else \if@specialpage
   \jg@use@specialpage
```

```
 \else \ifodd\count\z@
   \jg@use@oddpage\else \jg@use@evenpage
 \fi\fi\fi
 \global\@specialpagefalse
 \global\BlankPagefalse
}
```

This defines a blank page. The original code was wrong. This was corrected in V2. Originally there was a `\mark{}`, it has disappeared. Is this OK?

```
\def\nocontentbox{\vbox to \z@{}}
\def\BlankPage{%
 \global\BlankPagetrue
 \nocontentbox
 \newpage
}
```

The original output routine contained: `\hb@xt@ \textwidth {\@thehead}`. In the code that follows, we removed the line that decreases the text width by `\FOheadindent`, because this value is always zero. We also replace `\vfil` by `\vss`, in the case of a border.

```
\def\jg@headings#1#2{%
  \@tempdima\textwidth
  %\advance\@tempdima by -\FOheadindent
  \setbox\@tempboxa \vbox to #1{%
     \color@hbox
     \normalcolor
     \hb@xt@\textwidth{\hfill\llap{\hb@xt@\@tempdima{#2}}}%
     \color@endbox
     \vss%  \vfil
  }%
  \dp\@tempboxa \z@
  \box\@tempboxa}%
```

The boolean quantity `\ifForcePageSetup` is sometimes true. We shall see later that it is true inside a flow, said otherwise, almost always. The command is called at the end of the output routine; we increment the `\SimplePMRefs` counter, and recompute the new page master. This is done for every page; in V1, it was done once per `<fo:flow>`; this make the code a little bit slower.

```
\def\jg@check@setup{%
 \ifForcePageSetup
   \global\advance\SimplePMRefs1\relax
   \setaccordingtomaster
   \ifnum\NColumns>1\relax
     \refreshmulticols
   \fi\fi
}
```

This is the modified output routine. Changes in V2 use `\offinterlineskip` instead of the code commented out below. The result is not exactly the same. Added lines marked V2.

```
\def\@outputpage{%
\begingroup           % the \endgroup is put in by \aftergroup
  \let \protect \noexpand
%  \@resetactivechars
  \@parboxrestore
  \shipout \vbox{%
```

```
    \set@typeset@protect
    \aftergroup \endgroup
    \aftergroup \set@typeset@protect
  \jg@usepagestyle
  \reset@font
  \normalsize
  \normalsfcodes
  \let\label\@gobble
  \let\index\@gobble
  \let\glossary\@gobble
  \offinterlineskip
  %\baselineskip\z@skip \lineskip\z@skip \lineskiplimit\z@  % V2 remove
   \@begindvi
    \vskip \topmargin
    \vskip -\InnerTopMargin  % V2 add
    \moveright\@themargin \vbox {%
      \jg@headings{\headheight}{\@thehead}%
      \vskip \headsep
      \vskip\InnerTopMargin % V2 add
      \box\@outputbox
      \baselineskip \footskip
      \vskip \bottommargin
      \vskip-\tailheight % V2 add
      \jg@headings{\tailheight}{\@thefoot}%
      }%
    }%
  \global \@colht \textheight
  \stepcounter{page}
  \jg@check@setup  % V2 add
  \let\firstmark\botmark
}
```

Bootstrap code. Seems useless. As a consequence, no variable is added for blank pages.

```
\gdef\OddTail {}
\gdef\OddHead {}
\gdef\EvenTail {}
\gdef\EvenHead {}
\gdef\FirstTail {}
\gdef\FirstHead {}
\gdef\OddTailExtent{\z@}
\gdef\OddHeadExtent{\z@}
\gdef\EvenTailExtent{\z@}
\gdef\EvenHeadExtent{\z@}
\gdef\FirstTailExtent{\z@}
\gdef\FirstHeadExtent{\z@}
```

We simplified the code by removing all references to a quantity SpecialOffset that was always zero. We have already explained what the `\Pass` command does: it fetches some parameters from the region-body (note the fixed name here); they are `\columnsep`, `\NColumns`, `\Marginbottom` and `\Margintop`. We also evaluate `\Atomic:XXX`. The result is to define `\MasterXXXMargins`, as well as `\paperwidth` and `\paperheight`. We hope that both evaluations yield the same result. The only difference between them is that the left margin can change (we hope that the sum of the left

and right margins are the same). All these variables will be used by `\FOSetPage`, and forgotten after that.

```
\def\jg@flowIII{%
  \expandafter\Pass\csname\POdd:xsl-region-body\endcsname\\
  \csname Atomic:\POdd\endcsname
  \global\oddsidemargin\MasterLeftMargin
  \csname Atomic:\PEven\endcsname
  \global\evensidemargin\MasterLeftMargin
  \jg@set@headings
  \FOSetPage}
```

The purpose of this command is to compute the text height and text width, and to store it wherever needed (`\hsize`, `\vsize`, `\@colht`, etc). We also remember three quantities: `\bottommargin`, `\headsep` and `\topmargin`. There are two kinds of changes in V2. Some lines are commented out because some quantities are now computed differently. On the other hand, this command is called from `\jg@flowIII`, hence from `\@outputpage`, hence inside the `\output` command; this explains while assignments must be global. Moreover, since this is called for every page, the current page is no more special, and the assignment to `\if@specialpage` has to be moved elsewhere.

```
\def\FOSetPage{%
 \global\bottommargin\Marginbottom
% \global\headsep\Margintop
 \global\headsep\z@
 \global\topmargin\MasterTopMargin
 \global\textheight\paperheight
 \global\textwidth\paperwidth
% \global\advance\textheight by -\FirstHeadExtent
% \global\advance\textheight by -\FirstTailExtent
 \global\advance\textheight by -\MasterTopMargin
% \global\advance\textheight by -\Margintop
 \global\advance\textheight by -\MasterBottomMargin
% \global\advance\textheight by -\Marginbottom
 \global\advance\textwidth by -\MasterLeftMargin
 \global\advance\textwidth by -\MasterRightMargin
 \FOpdfsetpagesize{\paperwidth}{\paperheight}
 \global\global\@colht\textheight
 \global\@colroom\textheight
 \global\vsize\textheight
% \global\linewidth\textwidth
 \global\columnwidth\textwidth
 \global\hsize\columnwidth
 \global\linewidth\hsize
 \gdef\headheight{12pt}%
 \FOResetPageParts
% \global\@specialpagetrue
}
```

The next command is used for typesetting lists. In the case where `\This@LineWidth` is defined, i.e., non-`\relax`, we put its value into `\linewidth`.

```
\def\FOResetPageParts{
  \expandafter\ifx\csname This@LineWidth\endcsname\relax\else
      \global\linewidth\This@LineWidth\relax
 \fi
}
```

This might be useful.

```
\def\FOpdfsetpagesize#1#2{%
 \@ifundefined{pdfoutput}{}{%
   \global\pdfpagewidth\paperwidth
   \global\pdfpageheight\paperheight}}
```

Added in V2. Added at the start of a `\fo:flow` element.

```
\def\jg@start@multicolumns{%
   \xdef\PrevNColumns{\NColumns}%
   \ForcePageSetuptrue
   \ifnum\NColumns>1\relax \nobeginmulticols{\NColumns}%
   \fi}
```

Added at the end of a `\fo:flow` element.

```
\def\jg@end@multicolumns{%
  \ForcePageSetupfalse
  \ifnum\NColumns>1\relax \noendmulticols
  \else \clearpage
  \fi
}
```

This is the flow element. With all these simplifications and auxiliary commands, the code becomes easy to understand.

```
\XMLelement{fo:flow}
  {}
  {\global\SimplePMRefs1\relax
   \FOSetHyphenation
   \jg@flowI
   \setaccordingtomaster
   \jg@start@multicolumns
   }
  {
   \jg@end@multicolumns
   \jg@flowV
   }
```

4.8 Borders

A border has three properties: a color, a style and a width. There are four borders. They are called ‘before’, ‘after’, ‘start’ and ‘end’. These quantities are called relative, because they depend on the writing-mode trait. We assume that this is ‘lr-td’ (Inline components and text within a line are written left-to-right. Lines and blocks are placed top-to-bottom). In such a case, ‘before’ is ‘top’, ‘after’ is ‘bottom’, ‘start’ is ‘left’ and ‘end’ is ‘right’, see schema section 4.5. Quantities like ‘top’ and ‘bottom’ are called absolute. We shall use absolute quantities only to set relative quantities.

Here we declare the relative attributes.

```
\XMLNSA{fo}{border-before-color}{\FOborderbeforecolor}{\FOcolor}
\XMLNSA{fo}{border-after-color}{\FOborderaftercolor}{\FOcolor}
\XMLNSA{fo}{border-start-color}{\FOborderstartcolor}{\FOcolor}
\XMLNSA{fo}{border-end-color}{\FOborderendcolor}{\FOcolor}

\XMLNSAX{fo}{border-before-style}{\FOborderbeforestyle}{none}
```

```
\XMLNSAX{fo}{border-after-style}{\FOborderafterstyle}{none}
\XMLNSAX{fo}{border-start-style}{\FOborderstartstyle}{none}
\XMLNSAX{fo}{border-end-style}{\FOborderendstyle}{none}

\XMLNSA{fo}{border-before-width}{\FOborderbeforewidth}{medium}
\XMLNSA{fo}{border-after-width}{\FOborderafterwidth}{medium}
\XMLNSAX{fo}{border-start-width}{\FOborderstartwidth}{medium}
\XMLNSAX{fo}{border-end-width}{\FOborderendwidth}{medium}
```

Here we declare the absolute attributes. The default value is a special marker.

```
\XMLNSA{fo}{border-top-color}{\FObordertopcolor}{\LINK}
\XMLNSA{fo}{border-bottom-color}{\FOborderbottomcolor}{\LINK}
\XMLNSA{fo}{border-left-color}{\FOborderleftcolor}{\LINK}
\XMLNSA{fo}{border-right-color}{\FOborderrightcolor}{\LINK}

\XMLNSAX{fo}{border-top-style}{\FObordertopstyle}{\LINK}
\XMLNSAX{fo}{border-bottom-style}{\FOborderbottomstyle}{\LINK}
\XMLNSAX{fo}{border-left-style}{\FOborderleftstyle}{\LINK}
\XMLNSAX{fo}{border-right-style}{\FOborderrightstyle}{\LINK}

\XMLNSAX{fo}{border-top-width}{\FObordertopwidth}{\LINK}
\XMLNSAX{fo}{border-bottom-width}{\FOborderbottomwidth}{\LINK}
\XMLNSAX{fo}{border-left-width}{\FOborderleftwidth}{\LINK}
\XMLNSAX{fo}{border-right-width}{\FOborderrightwidth}{\LINK}

\XMLNSAX{fo}{border-left}{\FOborderleft}{\LINK}
\XMLNSAX{fo}{border-right}{\FOborderright}{\LINK}
\XMLNSAX{fo}{border-top}{\FOborder top}{\LINK}
\XMLNSAX{fo}{border-bottom}{\FOborderbottom}{\LINK}
```

The attribute border is a shorthand for all four borders. Its value is a list of three items: width, style and color. This means that border = 'thin solid red' is a valid specification. All four borders will have the same value; the command `\interpretwidth` replaces 'thin' by a numeric value.

```
\def\expandBorder#1 #2 #3\\{%
    \def\FOborderstartcolor{#3}%
    \def\FOborderendcolor{#3}%
    \def\FOborderbeforecolor{#3}%
    \def\FOborderaftercolor{#3}%
    \def\FOborderstartwidth{#1}%
    \def\FOborderendwidth{#1}%
    \def\FOborderbeforewidth{#1}%
    \def\FOborderafterwidth{#1}%
    \def\FOborderstartstyle{#2}%
    \def\FOborderendstyle{#2}%
    \def\FOborderbeforestyle{#2}%
    \def\FOborderafterstyle{#2}
    \interpretwidth
}
```

The width of a border can be a dimension, or one of 'thin', 'medium' or 'thick'. This command sets the width to 0.4pt, 0.8pt or 1.2pt accordingly.

```
\def\interpretwidth{%
   \ifx\FOborderwidth\att@thin\def\FOborderwidth{0.4pt}\fi
```

```
    \ifx\FOborderwidth\att@medium\def\FOborderwidth{0.8pt}\fi
    \ifx\FOborderwidth\att@thick\def\FOborderwidth{1.2pt}\fi
    \ifx\FOborderbeforewidth\att@thin\def\FOborderbeforewidth{0.4pt}\fi
    \ifx\FOborderbeforewidth\att@medium\def\FOborderbeforewidth{0.8pt}\fi
    \ifx\FOborderbeforewidth\att@thick\def\FOborderbeforewidth{1.2pt}\fi
    \ifx\FOborderafterwidth\att@thin\def\FOborderafterwidth{0.4pt}\fi
    \ifx\FOborderafterwidth\att@medium\def\FOborderafterwidth{0.8pt}\fi
    \ifx\FOborderafterwidth\att@thick\def\FOborderafterwidth{1.2pt}\fi
    \ifx\FOborderstartwidth\att@thin\def\FOborderstartwidth{0.4pt}\fi
    \ifx\FOborderstartwidth\att@medium\def\FOborderstartwidth{0.8pt}\fi
    \ifx\FOborderstartwidth\att@thick\def\FOborderstartwidth{1.2pt}\fi
    \ifx\FOborderendwidth\att@thin\def\FOborderendwidth{0.4pt}\fi
    \ifx\FOborderendwidth\att@medium\def\FOborderendwidth{0.8pt}\fi
    \ifx\FOborderendwidth\att@thick\def\FOborderendwidth{1.2pt}\fi
}
```

We declare here the border attribute. There are three other attributes that specify only one quantity (color, width, style) for all four borders.

```
\XMLNSAX{fo}{border}{\FOborder}{}
\XMLNSA{fo}{border-color}{\FObordercolor}{black}
\XMLNSAX{fo}{border-width}{\FOborderwidth}{}
\XMLNSAX{fo}{border-style}{\FOborderstyle}{}
```

We declare here the padding variables[8].

```
\XMLNSAX{fo}{padding}{\FOpadding}{\z@}

\XMLNSAX{fo}{padding-top}{\FOpaddingtop}{\z@}
\XMLNSAX{fo}{padding-bottom}{\FOpaddingbottom}{\z@}
\XMLNSAX{fo}{padding-left}{\FOpaddingleft}{\z@}
\XMLNSAX{fo}{padding-right}{\FOpaddingright}{\z@}

\XMLNSAX{fo}{padding-before}{\FOpaddingbefore}{\z@}
\XMLNSAX{fo}{padding-after}{\FOpaddingafter}{\z@}
\XMLNSAX{fo}{padding-start}{\FOpaddingstart}{\z@}
\XMLNSAX{fo}{padding-end}{\FOpaddingend}{\z@}
```

Here we declare the attributes for the margins, and we define default values. These values are absolute. The corresponding relative properties are space-before, space-after, space-start, and space-end. The quantities space-before and space-after are defined for block level formatting objects (in `\SpaceAttributes`), while space-start, and space-end are for inline objects[9]. There are also two quantities start-indent, end-indent which are related to margins, but are a bit complicated. They are defined later.

```
\XMLNSAX{fo}{margin}{\FOmargin}{}
\XMLNSAX{fo}{margin-left}  {\FOmarginleft}  {0pt}
\XMLNSAX{fo}{margin-right}  {\FOmarginright}  {0pt}
\XMLNSAX{fo}{margin-top}  {\FOmargintop}  {0pt}
\XMLNSAX{fo}{margin-bottom}  {\FOmarginbottom}  {0pt}

\gdef\FOmarginbottom{\z@}
\gdef\FOmarginleft{\z@}
```

[8] It seems that fotex does not use absolute values.

[9] They are unused by fotex.

```
\gdef\FOmarginright{\z@}
\gdef\FOmargintop{}
```

Absolute values have precedence over relative values, for the style, width and color of a border, hence the 12 first lines of the code. The case of the border is a bit special, and explained above. In the case of style, width, color, and margin, you can either set a single value, or four values. This code copies the single value in the four slots whenever adequate. In the case of color, there is a little problem: the code here makes no difference between black and no value. Finally, we hack the border style. The value can be one of 'none', 'hidden', 'dotted', 'dashed', 'solid', 'double', 'groove', 'ridge', 'inset', or 'outset'. Currently, only 'solid' is implemented. In all other cases, we set the width to zero and ignore it. In the case where one of the four borders is solid, we set a boolean value to true. The line marked 'JG' was not in the original...

```
\def\FOexpandattributes{%
\ifx\FObordertopstyle\LINK\else\let\FOborderbeforestyle\FObordertopstyle\fi
\ifx\FOborderbottomstyle\LINK\else\let\FOborderafterstyle\FOborderbottomstyle\fi
\ifx\FOborderrightstyle\LINK\else\let\FOborderendstyle\FOborderrightstyle\fi
\ifx\FOborderleftstyle\LINK\else\let\FOborderstartstyle\FOborderleftstyle\fi
\ifx\FObordertopwidth\LINK\else\let\FOborderbeforewidth\FObordertopwidth\fi
\ifx\FOborderbottomwidth\LINK\else\let\FOborderafterwidth\FOborderbottomwidth\fi
\ifx\FOborderrightwidth\LINK\else\let\FOborderendwidth\FOborderrightwidth\fi
\ifx\FOborderleftwidth\LINK\else\let\FOborderstartwidth\FOborderleftwidth\fi
\ifx\FObordertopcolor\LINK\else\let\FOborderbeforecolor\FObordertopcolor\fi
\ifx\FOborderbottomcolor\LINK\else\let\FOborderaftercolor\FOborderbottomcolor\fi
\ifx\FOborderrightcolor\LINK\else\let\FOborderendcolor\FOborderrightcolor\fi
\ifx\FOborderleftcolor\LINK\else\let\FOborderstartcolor\FOborderleftcolor\fi
  \ifx\FObordercolor\att@black
  \else
    \let\FOborderstartcolor\FObordercolor
    \let\FOborderendcolor\FObordercolor
    \let\FOborderbeforecolor\FObordercolor
    \let\FOborderaftercolor\FObordercolor
  \fi
  \ifx\FOborderwidth\@empty
  \else
    \let\FOborderstartwidth\FOborderwidth
    \let\FOborderendwidth\FOborderwidth
    \let\FOborderbeforewidth\FOborderwidth
    \let\FOborderafterwidth\FOborderwidth
  \fi
  \ifx\FOborderstyle\@empty
  \else
    \let\FOborderstartstyle\FOborderstyle
    \let\FOborderendstyle\FOborderstyle
    \let\FOborderbeforestyle\FOborderstyle
    \let\FOborderafterstyle\FOborderstyle
  \fi
  \ifx\FOborder\@empty
  \else
    \expandafter\expandBorder\FOborder\\{}%
  \fi
  \ifdim\FOpadding>\z@
    \let\FOpaddingstart\FOpadding
```

```
782     \let\FOpaddingend\FOpadding
783     \let\FOpaddingbefore\FOpadding
784     \let\FOpaddingafter\FOpadding
785   \fi
786   \ifx\FOmargin\@empty
787   \else
788     \let\tmpmargin\FOmargin
789     \let\FOmarginleft\tmpmargin
790     \let\FOmarginright\tmpmargin
791     \let\FOmargintop\tmpmargin
792     \let\FOmarginbottom\tmpmargin
793   \fi
794   \ifx\FOborderendstyle\att@solid
795    \FOBlockGrabtrue
796   \else
797    \def\FOborderendwidth{\z@}%
798   \fi
799   \ifx\FOborderstartstyle\att@solid
800    \FOBlockGrabtrue
801   \else
802    \def\FOborderstartwidth{\z@}%
803   \fi
804   \ifx\FOborderafterstyle\att@solid
805    \FOBlockGrabtrue %%  <---  JG
806   \else
807    \def\FOborderafterwidth{\z@}%
808   \fi
809   \ifx\FOborderbeforestyle\att@solid
810    \FOBlockGrabtrue
811   \else
812    \def\FOborderbeforewidth{\z@}%
813   \fi
814   \interpretwidth
815 }
```

4.9 Spacing for blocks

In XSL/Format a block is the equivalent of a TEX box. It can define some space before it, and after it. If we have two blocks, one with some space x after it and another one, with some space y before it, there are some precedence rules that explain what to do. These are not implemented in fotex. However assume that we have a sequence of spaces; each such sequence is defined by a triple (x_1, x_2, x_3), optimum, maximum, and minimum value. If I understand correctly, the following happens. First, the largest sequence of spaces is considered. Then, conditionality is considered; this is used at the start or end of a page, at the start or end of a line (in LATEX, it is the difference between `\hspace` and `\hspace*`); in such a case, all initial conditional spaces are removed. If any of these spaces is forcing, the result is the sum of the forcing spaces, otherwise, the merge of them. When merging spaces, only those of highest priority are used. Consider two of them (x_1, x_2, x_3), and (y_1, y_2, y_3). If $x_1 \neq y_1$ (different optimum values), then the one with lowest optimum value is discarded. Otherwise, the result is $(x_1 = y_1, \min(x_2, y_2), \max(x_3, y_3))$.

These spaces are local to a block, hence are not globally defined.

```
\def\SpaceAttributes{
 \XMLattributeX{space-after.optimum}{\FOspaceafteroptimum}{\z@}
 \XMLattributeX{space-after.maximum}{\FOspaceaftermaximum}{\z@}
 \XMLattributeX{space-after.minimum}{\FOspaceafterminimum}{\z@}
 \XMLattributeX{space-before.optimum}{\FOspacebeforeoptimum}{\z@}
 \XMLattributeX{space-before.maximum}{\FOspacebeforemaximum}{\z@}
 \XMLattributeX{space-before.minimum}{\FOspacebeforeminimum}{\z@}
 \XMLattributeX{space-after}{\FOspaceafter}{}
 \XMLattributeX{space-before}{\FOspacebefore}{}}
```

This is how we use these space-after.xxx attributes. Original code was inlined. Let A, B, and C, be the optimum, minimum and maximum values. The specifications say that, if `\FOspaceafter` is given, this should be the value of non-provided A, B and C. It also states that, if margin-top is defined, then setting space-before.minimum will have no effect. The code that follows does not implement these subtleties. Assume that the values are 2, 3 and 4pt. Then we could use some glue of value `3pt plus 1pt minus 1pt`. The shrink part is $A-B$, the stretch part is $C-A$. This code uses $A+C$ instead (strange). In fact, the stretch value can grow arbitrarily in TEX; said otherwise, we cannot implement exactly the XSL/Format mechanism. The argument of this command is a skip register, that contains the desired glue, or a TEX command that uses the glue.

```
\def\jg@usespaceafter#1{
  \ifx\@empty\FOspaceafter
      \@tempdima\FOspaceafteroptimum
      \advance\@tempdima by -\FOspaceafterminimum
      \@tempdimb\FOspaceafteroptimum
      \advance\@tempdimb by \FOspaceaftermaximum
      #1\FOspaceafteroptimum plus \@tempdimb minus \@tempdima
    \else
      #1\FOspaceafter
    \fi}
```

This is how we use these space-before.xxx attributes. Original code was inlined. The code is as above.

```
\def\jg@usespacebefore#1{
    \ifx\@empty\FOspacebefore
      \@tempdima\FOspacebeforeoptimum
      \advance\@tempdima by -\FOspacebeforeminimum
      \@tempdimb\FOspacebeforeoptimum
      \advance\@tempdimb by \FOspacebeforemaximum
      #1\FOspacebeforeoptimum plus \@tempdimb minus \@tempdima
    \else
      #1\FOspacebefore
    \fi}
```

Attributes for keep-together. There are three sub-cases: within line, page, column. The value can be 'always', 'auto', or a number. It corresponds in TEX to some penalty: 0 for 'auto', 10000 for 'always'. Otherwise, the number should produce something between these two values (currently ignored). Keep-within-line means a penalty in horizontal mode, otherwise in vertical mode. Note that it is not possible to associate a penalty to a column-break (in TEX switching from one column to the other is the same as switching from one page to the other; the difference is how `\output` handles these cases). The within-line case is not implemented.

```
\XMLNSAX{fo}{keep-together}{\FOkeeptogether}{\inherit}
\XMLNSAX{fo}{keep-together.within-column} {\FOkeeptogetherColumn}{\inherit}
```

```
\XMLNSAX{fo}{keep-together.within-page} {\FOkeeptogetherPage}{\inherit}
\XMLstringX\att@always<>always</>
```

Attributes for keep-with-next. As above. There is also a keep-with-previous, but this is not implemented. Too bad.

```
\XMLNSAX{fo}{keep-with-next}{\FOkeepwithnext}{auto}
\XMLNSAX{fo}{keep-with-next.within-column}{\FOkeepwithnextColumn}{auto}
\XMLNSAX{fo}{keep-with-next.within-page} {\FOkeepwithnextPage}{auto}
```

This was inlined. We call `\samepage` if no page break should occur.

```
\def\jg@keep@together{%
  \ifx\FOkeeptogether\att@always\samepage\fi
  \ifx\FOkeeptogetherColumn\att@always\samepage\fi
  \ifx\FOkeeptogetherPage\att@always\samepage\fi}
```

This is a bit more complicated: we set a switch that says that we have to keep this item with the next one.

```
\def\jg@keepnext{%
   \@tempswafalse
   \ifx\FOkeepwithnext\att@always\@tempswatrue\fi
   \ifx\FOkeepwithnextColumn\att@always\@tempswatrue\fi
   \ifx\FOkeepwithnextPage\att@always\@tempswatrue\fi}
```

In order to understand the following code, you must know that there are four kinds of blocks. If `\ifFOinOutput` is true, we are typesetting a static area (page headers and footers). If `\FOinTable` is positive, we are in a table, and a special case is when we typeset the label of an item in a list. Originally, the code did some action if `\FOTableNesting` was positive; however, it is currently impossible to nest tables, and the counter is never modified.

This inserts some vertical space. We compute in `\@tempskipa` the quantity to add and in `\@tempswa` a boolean value that says whether or not a penalty should be inserted. We insert the penalty and the skip. This resets to zero the value of `\FOspacebefore`. Moreover another quantity is computed but not used, it is not indicated here.

```
\def\FOvspacebefore{%
  \ifFOinOutput
  \else
    \jg@usespacebefore{\@tempskipa}
    \jg@keepnext
    \if@tempswa\addpenalty\@secpenalty\fi
    \addvspace\@tempskipa
  \fi
  \def\FOspacebefore{\z@}}
```

This adds some vertical space after a block. Nothing is done in table headings. Changes in V2: infinite penalty `\@M` before `\vspace` was replaced by a finite one, namely 9996, after `\vspace`.

```
\newskip\FOafterskip
\def\FOvspaceafter{%
  \ifFOinOutput
  \else
   \jg@usespaceafter{\FOafterskip}
   \jg@keepnext
   \addvspace\FOafterskip
   \if@tempswa\addpenalty{9996}\fi
  \fi}
```

Note : the package redefines `\addpenalty`, but the old code is the same as the new one, not shown here.

The code above was modified in the following way. Consider the case of a section title, A, followed by a subsection title B, followed by some text C. Both titles forbid a page break, hence `\if@tempswa` is true in both commands above, this means that we add twice a very high penalty (was infinite in V1). However, when we add a space at the start of B and C, this inserts some penalty (found in`\@secpenalty`). This is a negative one, hence encourages a page break before B and C, said otherwise, after A and B. The modification consists in remembering in a global variable the value of the second `@tempswa` (true if keep-with-next.within-page is 'always' for a given block), and to modify the action at the start of a block: we discard the value of `@tempswa` (that depends on keep-with-next), and instead, if the global variable is true insert a high positive penalty, and otherwise, a negative one. This gives much better page breaks; however, there is little stretchability between blocks, so that the height of the pages varies considerably.

4.10 Quadding

The value of the text-align can be one of the following: 'start', 'end', 'center', 'justify', 'insid'e, 'outside', 'left', 'right', or a string. The value of text-align-last can be any of these, plus 'relative'. The value 'relative' means that forced lines behave like other ones, except if the text is justified, case where forced lines are left aligned (in TEX, this corresponds to a default `\parfillskip`). By forced line, we mean either the last in a paragraph, or one induced by evaluation of the character U+000A (this is not implemented in fotex; note however that U+2028 calls `\newline`). This is a CSS property, adapted to XSL/Format; for this reason 'left' is the same as 'start' and 'right' as 'end'. If alignment is 'start', it means that there is no space between the first character and the margin, if it is 'end', then there is no space between the last character and the margin.

We declare the attributes.

```
\XMLNSAX{fo}{text-align}{\FOtextalign}{\inherit}
\XMLNSAX{fo}{text-align-last}{\FOtextalignlast}{\inherit}
\XMLstringX\att@relative<>relative</>
\gdef\FOtextalign{start}
\gdef\FOtextalignlast{relative}
```

The indentation (left and right) is given by two quantities, stored in `\FOstartindent` and `\FOendindent`. In the case where we have a list and an item in a list, the start of the body is defined by `body-start()` and the end of the label by `label-end()`. In fact, there are two traits, one that indicates the distance between the starts of the label and the body, and one that indicates the distance between the end of the label and the start of the body.

```
\XMLNSA{fo}{start-indent}{\FOstartindent}{\inherit}
\XMLNSA{fo}{end-indent}{\FOendindent}{\inherit}
\XMLstring\att@labelend<>label-end()</>
\XMLstring\att@bodystart<>body-start()</>
```

We use here two commands that return zero in the case where the attribute value is one of the functions mentioned above, and a third one that set these quantities to zero.

```
\gdef\EndIndent{\ifx\FOendindent\att@labelend\z@\else\FOendindent\fi}
\gdef\StartIndent{\ifx\FOstartindent\att@bodystart\z@\else\FOstartindent\fi}
\def\jg@hackindent{
   \ifx\FOstartindent\att@bodystart \let\FOstartindent\z@ \fi
   \ifx\FOendindent\att@labelend \let\FOendindent\z@ \fi}
```

We use three commands `\QuaddingStart`, `\Quadding` and `\QuaddingEnd`. The first function is defined as follows. This does not seem to correspond to the explanations given above.

```
\def\QuaddingStart{%
 \ifx\FOtextalignlast\att@relative
   \csname startQ@\FOtextalign\endcsname
 \else
   \csname startQ@\FOtextalignlast\endcsname
 \fi}
```

Function two for quadding.

```
\def\QuaddingEnd{%
 \ifx\FOtextalignlast\att@relative
   \csname endQ@\FOtextalign\endcsname
 \else
   \csname endQ@\FOtextalignlast\endcsname
 \fi}
```

Function three for quadding.

```
\def\Quadding{%
 \ifx\FOtextalignlast\att@relative
   \csname Q@\FOtextalign\endcsname
 \else
   \csname Q@\FOtextalignlast\endcsname
 \fi}
```

Remaining code in this paragraph comes from the file mlnames.sty. These commands describe the action in the case where the text should be centered. Note that line separator character U+2028 is bound to `\newline`, and needs to be redefined. Why is `\Q@centered` defined? The code of `\Q@center` is interesting. Assume that start- and end-indent are a and b respectively. In order to center the text in a region where a has been removed on the left and b on the right, we can put a plus 1fil in `\leftskip`, b plus 1fil in `\rightskip`. In fact we subtract $a + b$ from both these quantities (this does not change the alignment). Question: why do we change `\@rightskip`? In V2, fil was replaced by fill.

```
\def\startQ@center{\hskip\z@ plus 1filll}
\def\endQ@center{\hskip\z@ plus 1filll}
\def\Q@center{%
  \let\newline\@centercr
  \rightskip-\StartIndent plus 1fill%
  \@rightskip\rightskip
  \leftskip-\EndIndent plus 1fill%
  \parfillskip\z@skip
}
\let\Q@centered\Q@center
```

This is in the case right-justified, case where alignment is 'right' or 'end'. Changes in V2: `\rightskip` and `\@rightskip` added.

```
\def\startQ@end{\hfill}
\def\endQ@end{}
\def\Q@end{
  \let\newline\@centercr
  \leftskip\StartIndent plus 1fill % fill
  \rightskip\EndIndent
  \@rightskip\rightskip
  \parfillskip\z@skip
}
\let\startQ@right\startQ@end
```

```
\let\endQ@right\endQ@end
\let\Q@right\Q@end
```

This is in the case left-justified, case where alignment is 'start' or 'left'.

```
\def\startQ@start{}
\def\endQ@start{\hfill}
\def\Q@start{
  \let\newline\@centercr
  \rightskip\EndIndent plus 1fil
  \@rightskip\rightskip
  \leftskip\StartIndent
  \parfillskip\z@skip
}
\let\startQ@left\startQ@start
\let\endQ@left\endQ@start
\let\Q@left\Q@start
```

This is in the case left and right justified. Why do we need a definition for 'justified'?

```
\def\startQ@justify{}
\def\endQ@justify{}

\def\startQ@justified{%
  \leftskip\StartIndent
  \rightskip\EndIndent
  \@rightskip\rightskip
}
\def\Q@justified{%
  \parfillskip\@flushglue
  \leftskip\StartIndent
  \rightskip\EndIndent
  \@rightskip\rightskip
}
\def\endQ@justified{}
\let\Q@justify\Q@justified
```

Case empty. Strange.

```
\let\startQ@\startQ@justified
\let\endQ@\endQ@justified
\let\Q@\Q@justified
```

This is what the documentation says if the value of text-align is 'inside': If the page binding edge is on the start-edge, the alignment will be 'start'. If the binding is the end-edge, the alignment will be 'end'. If neither, use 'start' alignment. For 'outside', it is the opposite. If I understand correctly, this may depend on the parity of the page, hence cannot be implemented in TeX.

```
\def\startQ@pageoutside{\hfill}
\def\endQ@pageoutside{}

\def\startQ@pageinside{}
\def\endQ@pageinside{\hfill}
```

4.11 Arrays

Implementing arrays is a bit complicated. It uses a lot of variables. There were also different tentatives, so that some variables and tests have no usage anymore.

We start with a piece of code that remembers the column widths. This declares a counter.

```
\newcount\arraylength
```

After `\Array{foo}[bar]{gee}`, the command `\foobar` contains `gee`. This is assumed to be the width of column 'bar' of array 'foo'.

```
\def\Array#1[#2]#3{%
  \expandafter\xdef\csname #1#2\endcsname{#3}}
```

Initialization of the `foo` array. Column 0 of the array is set to empty. In the original code, this constructed `\foo` so that `\foo[bar]` calls `\foobar`, but the command was never used.

```
\def\DeclareArray#1{%
   \Array{#1}[0]{}}
```

This finds the length of an array by considering the first `\csname` that produces `\relax` as result (i.e. first undefined slot).

```
\def\getArraylength#1{%
   \arraylength0
   \loop\expandafter\ifx\csname #1\the\arraylength\endcsname\relax%
   \else\advance\arraylength by1\repeat}%
```

We find the end of the array, then insert something there.

```
\def\addToArray#1#2{\getArraylength{#1}%
        \Array{#1}[\the\arraylength]{#2}}%
```

This removes from memory everything associated to this table. Since it is not possible to remove the command from the hash table, we set it to `\relax` (not undefined!).

```
\def\clearArray#1{\getArraylength{#1}%
        \loop\ifnum\arraylength >0%
        \global\expandafter\let\csname #1\the\arraylength\endcsname\relax%
        \advance\arraylength by-1\repeat}%
```

These are the variables used below.

```
\newcount\AbsoluteTableCount   % unique Id for a table
\newcount\CellCount            % index of a cell in a row
\newcount\FOinTable            % >0 if in a table
\newcount\NCols                % non-zero if col specs given
\newdimen\CurrentCellWidth     % width of current cell
\newdimen\TableWidth           % width of the table
\newif\ifFOFirstCell           % unused...
\def\TableHeader{}             % current table header
\newtoks\BoxedFootnotes        % this contains the notes of the table
\NCols0
\FOinTable0
% \newcount\RowCount           % unused
% \newtoks\ColSpecs            % unused
```

We do not want TEX to insert vertical space between rows of our table. There is a command `\offinterlineskip` designed for this purpose (it is like the code shown here, with `\maxdimen` instead of $2^{14} - 1$). The command saves some parameters.

```
\def\saveinterlineskip{%
   \edef\savedbaselineskip{\the\baselineskip}%
```

```
    \edef\savedlineskip{\the\lineskip}%
    \edef\savedlineskiplimit{\the\lineskiplimit}%
    \baselineskip=-1000pt\relax
    \lineskiplimit=16383pt\relax
    \lineskip=0pt
}
```

This restores the settings saved by the previous command.

```
\def\restoreinterlineskip{%
  \baselineskip\savedbaselineskip\relax
  \lineskip\savedlineskip\relax
  \lineskiplimit\savedlineskiplimit\relax
}
```

In version two, some quantities are stored in global variables, associated to the array whose number is in `\AC` (short for `\AbsoluteTableCount`). This is the function that stores the parameters.

```
\def\jg@save@parameters{
    \addToArray{fotabletextalign\the\AC:}{\FOtextalign}%
    \addToArray{fotableborderbeforestyle\the\AC:}{\FOborderbeforestyle}%
    \addToArray{fotableborderafterstyle\the\AC:}{\FOborderafterstyle}%
    \addToArray{fotableborderstartstyle\the\AC:}{\FOborderstartstyle}%
    \addToArray{fotableborderendstyle\the\AC:}{\FOborderendstyle}%
    \addToArray{fotableborderbeforewidth\the\AC:}{\FOborderbeforewidth}%
    \addToArray{fotableborderafterwidth\the\AC:}{\FOborderafterwidth}%
    \addToArray{fotableborderstartwidth\the\AC:}{\FOborderstartwidth}%
    \addToArray{fotableborderendwidth\the\AC:}{\FOborderendwidth}%
    \addToArray{fotableborderbeforecolor\the\AC:}{\FOborderbeforecolor}%
    \addToArray{fotableborderaftercolor\the\AC:}{\FOborderaftercolor}%
    \addToArray{fotableborderstartcolor\the\AC:}{\FOborderstartcolor}%
    \addToArray{fotableborderendcolor\the\AC:}{\FOborderendcolor}%
}
```

This is the command that declares the parameters.

```
\def\jg@declare@parameters{%
  \DeclareArray{fotabletextalign\the\AC:}%
  \DeclareArray{fotableborderbeforestyle\the\AC:}%
  \DeclareArray{fotableborderafterstyle\the\AC:}%
  \DeclareArray{fotableborderstartstyle\the\AC:}%
  \DeclareArray{fotableborderendstyle\the\AC:}%
  \DeclareArray{fotableborderbeforewidth\the\AC:}%
  \DeclareArray{fotableborderafterwidth\the\AC:}%
  \DeclareArray{fotableborderstartwidth\the\AC:}%
  \DeclareArray{fotableborderendwidth\the\AC:}%
  \DeclareArray{fotableborderbeforecolor\the\AC:}%
  \DeclareArray{fotableborderaftercolor\the\AC:}%
  \DeclareArray{fotableborderstartcolor\the\AC:}%
  \DeclareArray{fotableborderendcolor\the\AC:}%
}
```

This is the command that clears the array.

```
\def\jg@clear@parameters{%
    \clearArray{fotabletextalign\the\AC:}%
    \clearArray{fotableborderbeforestyle\the\AC:}%
    \clearArray{fotableborderafterstyle\the\AC:}%
```

```
    \clearArray{fotableborderstartstyle\the\AC:}%
    \clearArray{fotableborderendstyle\the\AC:}%
    \clearArray{fotableborderbeforewidth\the\AC:}%
    \clearArray{fotableborderafterwidth\the\AC:}%
    \clearArray{fotableborderstartwidth\the\AC:}%
    \clearArray{fotableborderendwidth\the\AC:}%
    \clearArray{fotableborderbeforecolor\the\AC:}%
    \clearArray{fotableborderaftercolor\the\AC:}%
    \clearArray{fotableborderstartcolor\the\AC:}%
    \clearArray{fotableborderendcolor\the\AC:}%
}
```

This is the command that gets values from attributes or from quantities stored in the table.

```
\def\jg@compute@parameters{%
  \inheritfromcolumn{text-align}{textalign}%
  \inheritfromcolumn{border-before-style}{borderbeforestyle}%
  \inheritfromcolumn{border-after-style}{borderafterstyle}%
  \inheritfromcolumn{border-start-style}{borderstartstyle}%
  \inheritfromcolumn{border-end-style}{borderendstyle}%
  \inheritfromcolumn{border-before-width}{borderbeforewidth}%
  \inheritfromcolumn{border-after-width}{borderafterwidth}%
  \inheritfromcolumn{border-start-width}{borderstartwidth}%
  \inheritfromcolumn{border-end-width}{borderendwidth}%
  \inheritfromcolumn{border-before-color}{borderbeforecolor}%
  \inheritfromcolumn{border-after-color}{borderaftercolor}%
  \inheritfromcolumn{border-start-color}{borderstartcolor}%
  \inheritfromcolumn{border-end-color}{borderendcolor}%
}
```

The following command takes two arguments, say 'foo' and 'bar'. It looks at the attribute list of the current element, to see if 'foo' is in the list. If so, the command `\isexplicit` is set to a non-`\relax` value. Otherwise, we define `\FObar` to the value found in the table (this is the command that holds the value of the attribute 'foo' in case it is explicitly given).

```
\def\inheritfromcolumn#1#2{%
 \explicitattribute{#1}%
 \ifx\isexplicit\relax
  \expandafter\edef\csname FO#2\endcsname{%
    \csname fotable#2\the\AbsoluteTableCount:\the\CellCount\endcsname}%
 \fi
}
```

This sets the width of the table to be the argument, minus the `\tabcolsep`, the left margin and the right margin.

```
\def\jg@settablewidth#1{%
    \TableWidth#1%
    \advance\TableWidth by -\tabcolsep
    \advance\TableWidth by -\FOmarginleft
    \advance\TableWidth by -\FOmarginright}
```

Same code, without the `\tabcolsep`.

```
\def\jg@settablewidth@alt#1{%
    \TableWidth#1%
    \advance\TableWidth by -\FOmarginleft
    \advance\TableWidth by -\FOmarginright}
```

This initializes some other quantities.

```
\def\jg@tablesetup{%
   \NCols0
   \gdef\TableHeader{}%
   \NoTableSetup}
```

New in V2. If footnotes appear in a table, they are stored in a token list and inserted later on. This inserts the list, kills it, and resets the command that typesets footnotes.

```
\def\NoTableFinish{
  \ifnum\FOinTable=0
    \the\BoxedFootnotes
    \global\BoxedFootnotes={}%
    \global\let\FOfoottext\FOplainfoottext
  \fi
}
```

This is the command used at the start of a table. In this case and the previous one, if a table is in another one, the action is executed only for the outer table.

```
\def\j@start@tablenotes{
  \ifnum\FOinTable=0
    \global\BoxedFootnotes{}%
    \global\let\FOfoottext\FOboxedfoottext
  \fi
}
```

This piece of code is executed at the start of a table or tabular. In the case of a tabular in a table, it will be executed twice. We commented out the code that increments the table counter: this is because it is currently impossible to put tables in tables; and once a table is typeset, all information about it is discarded. However, some names remain in the hash table.

```
\def\NoTableSetup{%
  \ifx\FOwidth\att@auto\else     %% this test added by JG
    \jg@settablewidth{\FOwidth}%
 \fi
  % \global\advance\AbsoluteTableCount by 1 %
  \DeclareArray{fotable\the\AbsoluteTableCount:}%
  \jg@declare@parameters
  \global\CellCount0
  \jg@start@tablenotes
 }
```

We declare here an attribute for the placement of tables. This is in the fotex namespace. We declare also the reference orientation. This can be used in an inline container, for turning things.

```
\XMLNSAX{fo}{fotex:placement}{\FOkplacement}{}
\XMLNSAX{fo}{reference-orientation}{\FOreferenceorientation}{0}
\XMLname{fo:inline-container}{\FOInlineContainer}
\gdef\FOkplacement{}
```

A table is defined by the `<fo:table-and-caption>` element, has `<fo:table-caption>` (optional) and `<fo:table>` (required) as children.

In the case of `<fo:table-and-caption>`, we use a floating environment. This can be a table or a sideways table. We must close the same environment at the end.

```
\XMLelement{fo:table-and-caption}
  {}
  {
```

```
    \jg@settablewidth{\linewidth}
    \jg@tablesetup
     \ifx\XML@parent\FOInlineContainer
       \ifnum\FOreferenceorientation=0
       \else \begin{sidewaystable}\fi
     \else
      \ifnum\FOreferenceorientation=0
        \ifx\FOkplacement\@empty
         \begin{table}[!htbp]\FOlabel
          \else \edef\ktable{\noexpand\begin{table}[\FOkplacement]} \ktable \fi
       \else \begin{sidewaystable}\fi
     \fi
     \FOlabel
   }
   {\ifx\XML@parent\FOInlineContainer
      \ifnum\FOreferenceorientation=0  \else \end{sidewaystable}  \fi
    \else
      \ifnum\FOreferenceorientation=0 \end{table} \else \end{sidewaystable} \fi
    \fi
    \NoTableFinish
   }
```

Typesetting the caption is trivial. In particular, we do not call `\caption`. The table has a caption-side trait that explains where to put the caption. This will be ignored.

```
\XMLelement{fo:table-caption}
  {}
  {}
  {\par}
```

A `<fo:table>` can be used inside or outside of a `<fo:table-and-caption>`. The content: some `<fo:table-column>` elements, an optional `<fo:table-header>`, an optional `<fo:table-footer>` and some `<fo:table-body>` elements. Translation is trivial.

This is what we do for a table (equivalent of a LaTeX `tabular`). It has no header. Changes in V2: the whole table is put in a vbox, that starts with a top border and finishes with a bottom border.

```
\XMLelement{fo:table}
  {}
  {
   \FOexpandattributes
   \jg@settablewidth@alt{\linewidth}
   \jg@tablesetup
   \vbox\bgroup\FOBorderTop
  }
  {\FOBorderBottom\egroup
   \NoTableFinish}
```

The header of a table is not typeset now, but later. The footer is currently ignored. Too bad. The global definition here is one that forbids putting tables in tables.

```
\XMLelement{fo:table-header}
  {}
  {\xmlgrab}
  {\gdef\TableHeader{#1}}
```

This command is used to specify the width and other properties of one or more columns. The doc says: The number-columns-repeated property specifies the repetition of a `<fo:table-column>` specification n times; with the same effect as if the `<fo:table-column>` formatting object had been repeated n times in the result tree. The column-number property, for all but the first, is the column-number of the previous one plus the value of the number-columns-spanned property.

Changes in V2: attribute text-align added, but why? Some attributes are saved by the routines explained above. The list of these is larger than the list of attributes defined by the standard.

```
\XMLelement{fo:table-column}
 {
   \XMLattributeX{text-align}{\FOtextalign}{\inherit}
 }
 {
  \@tempcnta0
  \loop\ifnum\FOnumbercolumnsrepeated>\@tempcnta
   \advance\@tempcnta by 1
   {\NoTableColumn}%
  \repeat
  }
 {}
```

There is something wrong in this procedure: the column number is unused (said otherwise, if specifications are not in the order 1, 2, 3, etc, they will be stored in random order). Note: the column-number should not be zero. If at least one table column has been given, then `\Ncols` is not zero. The test to proportional-column-width is strange: what if the argument is not one? If the value is a percentage, it refers to the width of the table. The computed value is stored in `\@tempdima`, and then in the array data structure.

```
\def\NoTableColumn{%
   \ifx\@empty\FOcolumnnumber
    \global\advance\NCols by 1
   \else
    \global\NCols\FOcolumnnumber
   \fi
   \ifx\prop@width\FOcolumnwidth\def\FOcolumnwidth{1in}\fi
   \ifx\@empty\FOcolumnwidth\def\FOcolumnwidth{1in}\fi
   \TablePercentToDimen{\FOcolumnwidth}%
   \addToArray{fotable\the\AbsoluteTableCount:}{\the\@tempdima}%
   \jg@save@parameters
}
\XMLstringX\prop@width<>proportional-column-width(1)</>
```

The table body, header and footer contain either rows, or cells. Footers are currently ignored. Inside a table `\FOinTable` is 1. Since V2, the interlineskip parameters are saved and set to zero. Each row of the table is a box (a hbox in a vbox), and we do not want additional vertical space between the rows.

```
\XMLelement{fo:table-body}
  {  }
  { \FOFirstCelltrue
    \FOinTable1
    \saveinterlineskip
    \expandafter\NoTableStart{\TableHeader}%
  }
  { \NoTableEnd }
```

Action is trivial.

```
\def\NoTableStart#1{#1}
\def\NoTableEnd{%
   \clearArray{fotable\the\AbsoluteTableCount:}%
   \jg@clear@parameters
}
```

We use a command for typesetting a row.

```
\XMLelement{fo:table-row}
  {}
  {\xmlgrab}
  {\NoTableRow{#1}}
```

We use two passes for our table rows. For the first pass the height of the row may be unknown; hence we use a boolean that says that it is the first or the second pass.

```
\newdimen\NoTableCellHeight
\newif\ifNoTableCheckHeight
```

This is for the first pass. We use a `\strut`, a capital letter and a letter with a descender. Note: it is two small if the cell has a border or padding. Changes in V2. If a height attribute is given, it will be used instead this string for computation of the strut.

```
\def\jg@default@cell@height{%
 \setbox0=\vbox{
  \ifx\FOheight\att@auto%
    \strut They
  \else
    \rule{\z@}{\FOheight}%
  \fi
  }%
 \NoTableCellHeight=\ht0
 \advance\NoTableCellHeight by \dp0
 \NoTableCheckHeightfalse}
```

In the first pass, we put the row in a box, and we compute the total height plus depth of the box. Changes in V2: a hbox in a vbox is used instead of a simple hbox. Footnotes are ignored (in the second pass they will move). Then we look at page parameters to see if there is enough place on the current page; we may call `\clearpage`.

```
\def\jg@tablerow@firstpass#1{%
 \setbox0=\vbox{\hbox{\let\FOfoottext\FOnofoottext#1}}%
 \@tempdima=\ht0
 \advance\@tempdima by \dp0
 \FOspaceleft=\pagegoal
 \advance\FOspaceleft by -\pagetotal
 \ifdim\FOspaceleft<\@tempdima \clearpage \fi}
```

Second pass: we use the actual height as target height. In V1, we could use the box if not too big; but in V2, we have to process the table again, because otherwise we could lose footnotes.

```
\def\jg@tablerow@secondpass#1{
 \ifdim\@tempdima>\NoTableCellHeight
   \NoTableCellHeight=\@tempdima
 \fi
 \global\CellCount0
 \NoTableCheckHeighttrue
 \vbox to \NoTableCellHeight {\hbox{#1}}}
```

The code looks like this. In V1, we inserted the opposite of `\lineskip`; this assumes that TEX places a glue of value `\lineskip`, because the height of the box is greater than the baselineskip limit. It can fail if the height is small. In V2, we inhibit insertion of this glue, hence the code is not needed anymore.

```
\def\NoTableRow#1{%
 \jg@default@cell@height
 \global\CellCount0 \jg@tablerow@firstpass{#1}%
 \jg@tablerow@secondpass{#1}%
 %\vskip-\lineskip
}
```

In the case of a table cell, we call some functions. A table body can consist of rows, or cells. In the case of cells, we have an attribute that says if the cell starts or ends a row. We set `\FOinTable` to 2, meaning "in cell".

```
\XMLelement{fo:table-cell}
  {
   \XMLattributeX{ends-row}{\FOendsrow}{false}
   \XMLattributeX{starts-row}{\FOstartsrow}{false}
  }
  {\xmlgrab}
  {\FOlabel
  \FOexpandattributes
  \FOinTable2
  \NoTableCell{#1}}
```

This reduces the argument by the width of the padding, margin and border width on both sizes. Thus, we have in `#1`, a dimension register, the width of the region in which we can typeset the cell.

```
\def\jg@removemarg#1{
   \advance#1 by -\FOpaddingstart
   \advance#1 by -\FOpaddingend
   \ifx\FOborderstartstyle\att@solid\advance#1 by -\FOborderstartwidth\fi
   \ifx\FOborderendstyle\att@solid\advance#1 by -\FOborderendwidth\fi
   \advance#1 by -\FOmarginright
   \advance#1 by -\FOmarginleft}
```

We increment the cell count, or reset it, in case there is an attribute that says this cell is the first in a row.

```
\def\jg@test@startrow{%
 \ifx\FOstartsrow\att@true
%    \vskip-\lineskip  % not needed in V2
    \global\CellCount1
 \else
    \global\advance\CellCount by 1
 \fi}
```

If this cell is the last in a row, set the counter to zero.

```
\def\jg@test@endrow{%
 \ifx\FOendsrow\att@true
    %\vskip-\lineskip   % not needed in V2
    \global\CellCount0
 \fi}
```

Write `\CCWidth` instead of `\CurrentCellWidth` for simplicity. This is the width of the cell. If `\NCols` is zero, no specifications are given for the table. In this case, we use the natural width of the cell. Otherwise we use the stored value. We must reset the interline skip to its normal value.

```
\def\jg@getCCwidth#1{
 \ifnum\NCols<1
  \CCWidth\z@
  \setbox0=\hbox{\restoreinterlineskip#1\strut}%
  \CCWidth=\wd0
 \else
  \CCWidth=\csname fotable\the\AbsoluteTableCount:\the\CellCount\endcsname
  \jg@compute@parameters
  \interpretwidth
 \fi}
```

If the cell spans more than one column, we assume that specifications are given for all columns. We compute the sum of these quantities.

```
\def\jg@getCCwidth@aux{%
 \ifnum\FOnumbercolumnsspanned>1
    \@tempcnta1
    \loop\ifnum\@tempcnta<\FOnumbercolumnsspanned
        \advance\@tempcnta by 1
        \global\advance\CellCount by 1
        \advance\CCWidth\csname fotable\the
           \AbsoluteTableCount:\the\CellCount\endcsname
    \repeat
 \fi}
```

This is now the code of a cell. The parent is a table row or a table. If we are in a row, all cells are typeset (see next section) and the resulting boxes are put in a row-box, these row-boxes are put one above the other. These cells align nicely, because a cell in row i column j has a common height H_i and a width W_j (these value depends only on i and j and not the cell, the height is in `\NoTableCellHeight` and used only for the second pass, the width is in `\CCwidth`, real name of command shown above). In the case where the cell is directly in a table, the cell is put in a hbox, and `\leavevmode` is called. The attributes telling this is the first or last cell in a row have as only purpose to change the cell count. As a consequence, all cells are on a single line[10].

```
\def\NoTableCell#1{%
 \jg@test@startrow
 \jg@getCCwidth{#1}%
 \jg@removemarg{\CCWidth}
 \jg@getCCwidth@aux
 \ifx\XML@parent\FOTableRow
   \FOTableCellBlock#1\FOEndTableCellBlock
 \else
   \leavevmode\hbox{\FOTableCellBlock#1\FOEndTableCellBlock}%
 \fi
 \jg@test@endrow}
```

Some declarations.

```
\XMLNSA{fo}{column-width}{\FOcolumnwidth}{}
\XMLNSAX{fo}{number-columns-repeated}{\FOnumbercolumnsrepeated}{1}
```

[10] The `\vskip` commands, commented out in V2 as "not needed" on lines 1253 and 1260, have as side effect to end the paragraph. This gives unexpected side effects. In the Raweb case, cells are not directly in a table.

```
\XMLNSAX{fo}{number-columns-spanned}{\FOnumbercolumnsspanned}{1}
\XMLNSAX{fo}{column-number}{\FOcolumnnumber}{}
```

4.12 Boxed blocks

We typeset a cell by opening in a lrbox (this is really a `\hbox`) in which we open a `\vbox`. Since V2, a color group is added only if the `\ifFOinOutput` switch is false.[11] We hack spacing. We removed a test to an undefined variable, that could center vertically the box.

```
\def\FOTableCellBlock{%
 \begin{lrbox}{\CellBox}%
 \vbox\bgroup
 \hsize\the\CurrentCellWidth
 \restoreinterlineskip
 \ifFOinOutput\else \color@begingroup\fi
 \FOSetFont{tablecellblock}%
 \jg@activew}
```

This is a bit longish, but easy to understand. We finish our `\vbox` and our lrbox. After that, we construct some boxes. Let's denote by PA, PB, PS, and PE the padding before, after (vertical), start, end (horizontal), by BA, BB, BS, and BE the border width (whenever the border is solid), and by ML, MR, MT, and MB the margins (left, right, top, bottom). Let a be the sum of MT, PB and BB, and let b be the sum of PS, PE and the width of the cell box. The first box we construct is A, the cell box. The second box we construct is B, an hbox containing PS, A and PE. The third box is C, a vbox containing PB, filler, B, filler, and PA. The filler is a `\vfil`, which is inserted in certain circumstances : if NoTableCheckHeight is true, we insert the filler if some attributes have the right value. In this case C is a `\vtop` to the height of the table computed earlier (this is the height of the row, we are in the second pass). The width of this box is the quantity b defined above. Now we construct a box D, an hbox with BS, bg, C, and BE, where bg may be empty if no background is desired. Otherwise it consists of a rule of width b, followed by a kern of width minus b (the height of the rule is the height of C). Then comes a box E, this is a vbox containing BB, D and BA. Note that BS, BE, BB, and BA are borders: they are implemented via hrules or vrules. Then comes a box F, it is a vbox containing MT, E and MB, and a box G, this is an hbox that contains ML, F and MR. This box is shifted down by the quantity a. Finally, we put everything in an hbox.

Changes in V2: Horizontal margins use `\hskip`, but vertical margins use `\kern` instead of `\vskip`. In the case of BoxedBlock, `\kerns` are used for both horizontal and vertical margins.

```
\def\FOEndTableCellBlock{%
 \ifx\FOverticalalign\att@top\vfill\fi
 \ifFOinOutput\else \color@endgroup\fi
 \egroup
 \end{lrbox}%
 \@tempdima\FOmargintop
 \advance\@tempdima\FOpaddingbefore
 \ifx\FOborderbeforestyle\att@solid\advance\@tempdima\FOborderbeforewidth\fi
\@tempdimb\wd\CellBox
\advance\@tempdimb by \FOpaddingstart
\advance\@tempdimb by \FOpaddingend
\hbox{%
    \lower\@tempdima
```

[11] What if the page footer contains a table with a blue cell?. Note that the `\clearpage` on line 1220 forbids putting tables inside page headers...

```
      \hbox{%
       \hskip\FOmarginleft
       \vbox{%
         \kern\FOmargintop
         \vbox{%
         \ifx\FOborderbeforestyle\att@solid
            {\color{\FOborderbeforecolor}\hrule\@height\FOborderbeforewidth}%
         \fi
         \hbox{%
           \ifx\FOborderstartstyle\att@solid
            {\color{\FOborderstartcolor}\vrule\@width\FOborderstartwidth}%
           \fi
           \ifx\FObackgroundcolor\att@transparent
           \else
            {\color{\FObackgroundcolor}\vrule\@width\@tempdimb\kern-\@tempdimb}%
           \fi
            \ifNoTableCheckHeight
              \vtop to \NoTableCellHeight{%
                \kern\FOpaddingbefore
                \ifx\FOdisplayalign\att@auto
                  \else\ifx\FOdisplayalign\att@before
                    \else\ifx\FOdisplayalign\att@after\vfil
                      \else\ifx\FOdisplayalign\att@centered\vfil\fi
                    \fi
                  \fi
                \fi
                \hbox{\kern\FOpaddingstart\box\CellBox\kern\FOpaddingend}%
                \ifx\FOdisplayalign\att@auto\vfil
                  \else\ifx\FOdisplayalign\att@before\vfil
                    \else\ifx\FOdisplayalign\att@after
                      \else\ifx\FOdisplayalign\att@centered\vfil\fi
                    \fi
                  \fi
                \fi
                \kern\FOpaddingafter
              }%
            \else
             \vbox{%
               \kern\FOpaddingbefore
                 \hbox{\kern\FOpaddingstart\box\CellBox\kern\FOpaddingend}%
               \kern\FOpaddingafter
               }%
           \fi
          \ifx\FOborderendstyle\att@solid
           {\color{\FOborderendcolor}\vrule\@width\FOborderendwidth}%
          \fi
         }%
         \ifx\FOborderafterstyle\att@solid
            {\color{\FOborderaftercolor}\hrule\@height\FOborderafterwidth}\fi
         }%
         \kern\FOmarginbottom
      }%
```

```
      \hskip\FOmarginright
     }%
    }%
}
```

This is done when we start a boxed object. This is like the start of a cell, but a bit simpler. Note the `\Quadding` and the `\strut`.[12]

```
\def\FOBoxedBlock#1{%
 \@tempdimb#1%
 \jg@removemarg{\@tempdimb}
 \begin{lrbox}{\BlockBox}%
 \vbox\bgroup
 \hsize\the\@tempdimb
 \FOSetFont{tableblock}%
 \color@begingroup
 \jg@activew
 \parindent\FOtextindent
 \Quadding
 \start@strut }
```

This is like the end of a cell box. It is a bit simpler. Dimensions a and b are renamed to b and c. We construct a box, as follows: A is the box that contains the material seen so far, B contains PS, A and PE, C contains PB, B and PA, D contains BS, bg, C, and BE, E contains BD, D and BA, F contains MT, E and MB, finally G contains Ml, F and MR. The difference with a cell: no vertical skip, and Ml is margin-left plus text-indent[13].

```
\def\FOEndBoxedBlock{%
 \start@strut
 \color@endgroup
 \egroup
 \end{lrbox}%
 \@tempdimb\FOmargintop
 \advance\@tempdimb\FOpaddingbefore
 \ifx\FOborderbeforestyle\att@solid\advance\@tempdimb\FOborderbeforewidth\fi
 \@tempdimc\wd\BlockBox
\advance\@tempdimc by \FOpaddingstart
\advance\@tempdimc by \FOpaddingend
\FOtempdim\FOmarginleft
\advance\FOtempdim by \FOtextindent
\hbox{%
    \lower\@tempdimb
     \hbox{%
      \kern\FOtempdim
      \vbox{%
        \kern\FOmargintop
        \vbox{%
        \ifx\FOborderbeforestyle\att@solid
           {\color{\FOborderbeforecolor}\hrule\@height\FOborderbeforewidth}%
        \fi
        \hbox{%
          \ifx\FOborderstartstyle\att@solid
            {\color{\FOborderstartcolor}\vrule\@width\FOborderstartwidth}\fi
```

[12] We hacked a bit this code.
[13] Why do we add the indentation?

```
          \ifx\FObackgroundcolor\att@transparent
          \else
           {\color{\FObackgroundcolor}\vrule\@width\@tempdimc\kern-\@tempdimc}%
          \fi
            \vbox{%
              \kern\FOpaddingbefore
               \hbox{\kern\FOpaddingstart\box\BlockBox\kern\FOpaddingend}%
              \kern\FOpaddingafter
              }%
          \ifx\FOborderendstyle\att@solid
           {\color{\FOborderendcolor}\vrule\@width\FOborderendwidth}\fi
        }%
        \ifx\FOborderafterstyle\att@solid
           {\color{\FOborderaftercolor}\hrule\@height\FOborderafterwidth}\fi
        }%
        \kern\FOmarginbottom
     }%
     \kern\FOmarginright
    }%
   }%
}
```

Attributes.

```
\XMLNSAX{fo}{display-align}{\FOdisplayalign}{\inherit}
\XMLstringX\att@top<>top</>
\gdef\FOdisplayalign{auto}
\XMLstringX\att@after<>after</>
\XMLstringX\att@before<>before</>
```

4.13 Lists

In XSL/Format four formatting objects construct lists: the main element is `<fo:list-block>`, the children are one or more `<fo:list-item>`, each containing a pair of `<fo:list-item-label>` and `<fo:list-item-body>`. There are two attributes, specific to lists. We shall abbreviate the names to PLS and PDBS.

```
\XMLNSAX{fo}{provisional-label-separation}
   {\FOprovisionallabelseparation}{\inherit}
\XMLNSAX{fo}{provisional-distance-between-starts}
   {\FOprovisionaldistancebetweenstarts}{\inherit}
```

This command is used twice: at the start of a list, and at the start of a block in a list. It defines some parameters (item indent, left margin, right margin, and label width) that are used by the `\item` command. Changes in V2: PDBS can be a percentage.

```
\def\jg@use@listparams{
  \itemindent=\FOstartindent
  \PercentToDimen{\FOprovisionaldistancebetweenstarts}%
  \leftmargin=\@tempdima\relax
  \rightmargin=\FOmarginright
  \labelwidth=\@tempdima\relax
  \advance\labelwidth by -\FOprovisionallabelseparation}
```

The translation of `<fo:list-block>` is essentially a call to the `list` environment (it takes two arguments, explained below). We globally change (increase by one) the `\FOinList` counter. The text-align attribute will be used for alignment of the labels, for instance, if it is 'start', we call the command `\Liststart`, that defines `\makelabel` adequately (it will left justify the label). We use `\csname` for this purpose; the `\expandafter` before it is useless.

Changes in V2: Call to `\FOSetFont` commented out. Setting of `\partopsep` added. Conditional call to `\leavevmode` and final `\par` commented out. As a consequence a list could be inline. This means that some checks have to be made elsewhere.

There is a second change in V2, it concerns `\linewidth`. The question is: how should this be defined? what command uses it? In general, the value is the same as `\hsize`. It is used and modified by `\list` (as shown below), and used by tables (in a non-trivial manner by `\@stopline`, but this is not used by fotex). However, we have already seen that the fotex implementation of tables uses this value, and we have seen another use of the variable. Now comes the trick. Let X be the command `\This@LineWidth`, and Y its value, let H be the value of `\hsize`. The command `\setaccordingtomaster` is called for each page, and for each flow; it calls `\FOResetPageParts` that redefines `\hsize`. It also redefines `\linewidth` to Y, if X is defined, and to H otherwise. The question is whether or not the value should be modified (whenever a new flow is created, it is clear that we should set it to H, when a new page is entered, the value should not change). Given this scheme, the variable X, whenever it has a value, should contain the right quantity. The code here defines two scopes: the `<fo:list-block>` element and the `list` environment, let's call them A and B. The first definition is in scope A, and the purpose is to restore the value of the line width (it will be globally modified). The second definition is in scope B, it is used only in the case where a page break occurs while typesetting the list.

```
\XMLelement{fo:list-block}
  {\SpaceAttributes}
  {
   \jg@hackindent
   %\FOSetFont{normal}%
   \advance\FOinList by 1\relax
   %\ifnum\FOinList>1\relax\leavevmode\fi
   \edef\This@LineWidth{\the\linewidth}%
   \begin{list}{}{%
     \jg@use@listparams
     \advance\leftmargin by \FOmarginleft
     \expandafter\csname List\FOtextalign\endcsname
     \labelsep\FOprovisionallabelseparation
     \itemsep\z@ \parsep\z@ \partopsep\z@ \topsep\z@  \parskip\z@
     \jg@usespacebefore{\topsep} }
     %\edef\This@LineWidth{\the\linewidth}% comment JG
   }
  {\end{list}
   \global\linewidth\This@LineWidth\relax
   \advance\FOinList by -1\relax
   %\par
  }
```

These are the definitions used in the code above. The default value (reset by `\list` in any case) corresponds to 'end'.

```
\def\Listjustified{  \gdef\makelabel##1{##1}}
\def\Liststart{      \gdef\makelabel##1{##1\hfil}}
\def\Listend{        \gdef\makelabel##1{\hfil##1}}
```

```
\def\Listcentered{   \gdef\makelabel##1{\hfil##1\hfil}}
\def\Listcenter{     \gdef\makelabel##1{\hfil##1\hfil}}
```

The command `\list` has been changed in V2, as explained above. It takes two arguments. The first defines `\@itemlabel`, the value to be used when `\item` has no optional argument (unused here), and the second contains a list of commands to be executed before the call to `\@trivlist`. The standard `\list` command increments `\@listdepth` and tests it for overflow; this is useless in our case, especially because the effects of command that depends on the list level (for instance `\@listii`) are neutralized via the second argument. In effect we define `\itemsep`, `\parsep`, `\partopsep`, `\listparindent`, and `\parskip` to be zero, `\topsep` to be the value of the space-before attribute, `\itemindent` to be the value of start-indent. After that, the left and right margin will take the value of the corresponding attribute, but the left margin is increased by the value of PDBS, `\labelwidth` will contain PDBS minus PLS and `\labelsep` will contain PLS. After `\@trivlist`, we copy two zero quantities in `\parskip` and `\parindent`. We subtract left and right margin from `\linewidth`. Note: this is done globally in V2, and the caller restores the old value (this is a bit strange). Finally, we call `\parshape`, so that the first line (holding the item) is typeset differently.

```
\def\list#1#2{%
  \ifnum \@listdepth >5\relax \@toodeep
  \else \global\advance\@listdepth\@ne
  \fi
  \rightmargin\z@
  \listparindent\z@
  \itemindent\z@
  \csname @list\romannumeral\the\@listdepth\endcsname
  \def\@itemlabel{#1}%
  \let\makelabel\@mklab
  \@nmbrlistfalse
  #2\relax
  \@trivlist
  \parskip\parsep
  \parindent\listparindent
  \global\advance\linewidth -\rightmargin        % V2
  \global\advance\linewidth -\leftmargin         % V2
  \advance\@totalleftmargin \leftmargin
  \parshape \@ne \@totalleftmargin \linewidth
  \ignorespaces}
```

The `\endlist` command calls `\endtrivlist` (shown below) and decrements the list depth counter. For completeness, we show here the code of `\@trivlist`. The command on the first line is "a switch set true by a sectioning command when it is creating an in-text heading with `\everypar`" according to Lamport. In such a case, the heading is kept in a variable, and output only when a new paragraph is started, hence the `\leavevmode`. Then comes a piece of code shown below, and settings of three skip parameters (for justification). We copy `\parskip` somewhere, set a boolean to true and start checking for missing items.

```
\def\@trivlist{%
  \if@noskipsec \leavevmode \fi
  \jg@trivlist@A
  \leftskip \z@skip
  \rightskip \@rightskip
  \parfillskip \@flushglue
  \jg@trivlist@B
  \global \@newlisttrue
  \@outerparskip \parskip}
```

Four boolean variables control how lists are typeset. We have already seen `@newlist`. This is true as long as the list is new. Said otherwise, it is set to true at the start of the list, and set to false at the start of the first paragraph (`\everypar` in an item). This redefines `\par`, to be `\@@par` (the primitive command) if the list is not new, and otherwise to do nothing (however, after a thousand calls an error will be signaled). As a consequence, if there is some text before the first item, even if there are `\par` commands, the current mode will be horizontal.

```
\def\jg@trivlist@B{%
 \par@deathcycles \z@
 \@setpar{%
   \if@newlist
     \advance\par@deathcycles \@ne
     \ifnum \par@deathcycles >\@m \@noitemerr {\@@par}\fi
   \else {\@@par} \fi}}%
```

The second switch, `@inlabel`, "is false except between the time an `\item` is encountered and the time that TEX actually enters horizontal mode", the quote is from Lamport. At the end of a list, or inside `\newpage`, if the switch is true, `\leavevmode` is executed, and the switch reset to false; this has as effect to flush pending labels. Otherwise, the switch is set to true in `\jg@itemA` shown below, and to false by `\jg@itemB`. There is third switch, `@noparitem`, it is set by `\list` to true if `@inlabel` is true. This is the case when the user says, for instance `\item[foo] \begin{itemize}\item[bar]`. In this case, we have two items in a row. Finally `@noparlist` is set at the start of a list as a copy of `@inlabel`.

Two dimensions are defined. The first is `\@topsepadd`, containing `\topsep` plus `\partopsep` (omitted in horizontal mode); this is copied in `\@topsep`, unless we are in a label (case where `\@topsep` is left unchanged). If `\noparlist` is false, the command `\@endparenv` is called at the end of the list, and inserts the `\@topsepadd` vertical glue. Denote this by T', denote by P the value of `\parskip`, and let T be the sum of these two quantities; we put T in `\@topsep`; it will be inserted by `\@item`.

```
\def\jg@trivlist@A{%
  \@topsepadd \topsep
  \ifvmode \advance\@topsepadd \partopsep \else \unskip \par \fi
  \if@inlabel \@noparitemtrue \@noparlisttrue
  \else
    \if@newlist \@noitemerr \fi
    \@noparlistfalse
    \@topsep \@topsepadd
  \fi
  \advance\@topsep \parskip}
```

Second part of `\endtrivlist`. This is used when `\noparlist` is false, said otherwise, if the list was not inside a label. Let L, P and O be the values of the last skip, the parskip, and `\@outerparskip` (the parskip that was in effect before the list). If the last item in the vertical list is glue, then L contains this value, otherwise it will be zero. If L is negative or zero, nothing is done; otherwise, the opposite of L is added, followed by $L + P - O$. This gives three items of glue; there are similar cases where a penalty is inserted after the second glue.

```
\def\jg@endtrivlist@B{%
    \ifdim\lastskip >\z@
      \@tempskipa\lastskip \vskip -\lastskip
      \advance\@tempskipa\parskip \advance\@tempskipa -\@outerparskip
      \vskip\@tempskipa
    \fi}
```

The code that follows handles the special case of a list A, containing an item B, that starts with a list C; in this case the `@noparitem` switch is true. When we see the first item D in the list C, we call this piece of code (the source has a comment: "this clause hardly every taken, so made a macro"). The action is divided into three parts: first, we set the switch to false, second, we shift the labels left by the value of `\leftmargin` (note that this is how much the current list is indented with respect to its environment), finally, we adjust vertical space. We add two items of glue. With the the same notations as before, we insert are $-L$ and $L+O-P$. When the label is finally printed, TEX will use the current `\parskip`, hence P. This gives a total of $L+O$: the old glue plus the old `\parskip`. Note that the code above inserts $L+P-O$.

```
\def\@donoparitem{%
  \@noparitemfalse
  \global\setbox\@labels\hbox{\hskip -\leftmargin
                              \unhbox\@labels
                               \hskip \leftmargin}%
  \if@minipage\else
    \@tempskipa\lastskip
    \vskip -\lastskip
    \advance\@tempskipa\@outerparskip
    \advance\@tempskipa -\parskip
    \vskip\@tempskipa
  \fi}
```

First part of `\endtrivlist`. If we are still in the label, we enter horizontal mode, and say that we are no more in the label. An error is signaled if the list is new (case where no item was found).

```
\def\jg@endtrivlist@A{%
  \if@inlabel
    \leavevmode
    \global \@inlabelfalse
  \fi
  \if@newlist
    \@noitemerr
    \global \@newlistfalse
  \fi}
```

This is now `\endtrivlist`, as defined by V2. The test `\ifInInsertion` was added. An error test (if the end of the list occurs in math mode) was removed for no clear reason.

```
\def\endtrivlist{%
  \jg@endtrivlist@A
  \ifhmode\unskip \par\fi %% else error omitted
  \if@noparlist \else\ifInInsertion\else
    \jg@endtrivlist@B\fi
    \@endparenv
  \fi}
```

Quoted from the LATEX source: "When you leave a list environment, returning either to an enclosing list or normal text mode, LATEX begins a new paragraph if and only if you leave a blank line after the `\end` command. This is accomplished by the `\@endparenv` command. Blank lines are ignored every other reasonable place". Later on, there is "Instead of redefining `\par` and `\everypar`, `\@endparenv` was changed to set the `@endpe` switch, letting `\end` redefine `\par` and `\everypar`". The fotex V2 file redefines the command by adding a test to `\ifInInsertion` (true in a footnote).

```
\def\@endparenv{%
  \ifInInsertion\else
```

```
1559      \addpenalty\@endparpenalty
1560      \addvspace\@topsepadd
1561      \@endpetrue
1562    \fi}
```

The `\fotex` file redefines `\item` like this. Note that `\leavevmode` could be used to flush pending labels.

```
1563 \let\olditem\item
1564 \def\item{\if@inlabel\leavevmode\fi\olditem}
```

We now redefine the `\item` command. This is a bit complicated. For this reason, we split the command into smaller pieces.

```
1565 \def\@item[#1]{%
1566   \@itemA
1567   \@itemB
1568   \@itemC
1569   \@itemD{#1}%
1570   \@itemE
1571   \ignorespaces}
```

The first idea is that the label is typeset in a box, and this box will be used later. The `\sbox` command produces essentially a hbox. The LaTeX source file says that the `\label` command should produce some glue, for instance, the code on lines 1473 to 1476 is OK, the code on line 1472 is not. The first line was added because it corrects a bug. See explanations later.

```
1572 \def\@itemD#1{%
1573   %\savebox{\ItemBox}{#1}% Added Grimm
1574   \sbox\@tempboxa{\makelabel{#1}}}%
```

We can have more than one consecutive labels. For this reason, we put in the box `\@labels` all these labels. We consider three dimensions, say A, B, and C, that contain the space after the label, the nominal width of the label, and the total space allowed for the label and its surrounding space. We emit four items of glue, first, three items that correspond to $C - A - B$, then the label box, then A. In the case where the width of the label is smaller than B, we use `\hbox to XX`, with a `\unhbox`: here it is important that the box contains glue that can stretch without producing underful boxes.

```
1575 \def\@itemE{%
1576   \global\setbox\@labels\hbox{%
1577     \unhbox\@labels
1578     \hskip \itemindent
1579     \hskip -\labelwidth
1580     \hskip -\labelsep
1581     \ifdim \wd\@tempboxa >\labelwidth
1582       \box\@tempboxa
1583     \else
1584       \hbox to\labelwidth {\unhbox\@tempboxa}%
1585     \fi
1586     \hskip \labelsep}}%
```

In the case of an enumeration, there is a counter, whose name is in `\@listctr`, that has to be incremented. We know that we are in an enumeration, because `\if@nmbrlist` is true (in the FO case, it is false). In the case `\item` has an optional argument, `\if@noitemarg` is false, and in this case the user gives a number, so that the counter is not modified.

```
1587 \def\@itemC{%
1588   \if@noitemarg
```

```
1589      \@noitemargfalse
1590      \if@nmbrlist \refstepcounter\@listctr \fi
1591    \fi}
```

The following piece of code is executed if `\if@newlist` is true (hence for the first item in a list). The effect of `\@nbitem` is to add some vertical space (the value is $O - P$, according to the description above). The LaTeX source has a comment that says: "Kludge if list follows `\section`". This was commented out by fotex.sty. Otherwise, we use two `\addvspace` instead of a single one, namely T and minus P; remember that T' is $T - P$.

```
1592 \def\@itemAA{%
1593 %       \if@nobreak \@nbitem \else  %%% REMOVED
1594          \addpenalty\@beginparpenalty
1595          \addvspace\@topsep
1596          \addvspace{-\parskip}%
1597 %       \fi
1598 }
```

This is now how `\item` manages vertical space. The code on line 1603 is strange: remember that `\indent` inserts current paragraph indentation, while `\par` terminates the current paragraph (all we need is to insert the `\everypar` token list, in order to flush previous items). On the next line, we have a double `\unskip`; the reason is that commands like `\addvspace` add a double skip in order to make it more difficult to remove it. However, this is generally not done in horizontal mode, strange... Changes in V2: `\ifInInsertion` added (this test is true inside a foot note).

```
1599 \def\@itemA{
1600   \if@noparitem
1601     \@donoparitem
1602   \else
1603     \if@inlabel \indent \par \fi
1604     \ifhmode \unskip\unskip \par \fi
1605     \ifInInsertion \else
1606        \if@newlist \@itemAA
1607        \else \addpenalty\@itempenalty \addvspace\itemsep \fi
1608     \fi
1609     \global\@inlabeltrue
1610   \fi
1611 }
```

Now, the `\everypar` token list is redefined. The difference with the original LaTeX code is that some `\global` declarations have been added. In the case `\if@inlabel` is true, we set it to false, and output the label box. We kill `\itemindent`, in the case where the user says `\noindent` (in other cases, a new paragraph is created by an explicit or implicit `\indent`, that inserts a box of width `\parindent`). There is another change in V2: the call to `\FOlabel`; we commented it out, because the `\FOlabel` is also called after the item is completely typeset (and before the box is flushed, but can a page break appear just after the link? this is not so clear).

```
1612 \def\@itemB{%
1613  \global\everypar{%   %% Global added
1614     \@minipagefalse
1615     \global\@newlistfalse
1616     \if@inlabel
1617       \global\@inlabelfalse
1618       {\setbox\z@\lastbox
1619        \ifvoid\z@ \kern-\itemindent \fi}%
1620       \box\@labels\FOlabel %% added in V2
```

```
1621        \penalty\z@
1622      \fi
1623      \if@nobreak
1624        \global\@nobreakfalse   % Global added
1625        \clubpenalty \@M
1626      \else
1627        \clubpenalty \@clubpenalty
1628        \global\everypar{}%   % Global added
1629      \fi}}
```

A list should contain `<fo:list-item>` elements. We insert some vertical space at the start of the item. The `\par` was missing in the first version. Without it, the following happens: the command `\jg@usespacebefore` computes some quantity in `\@tempdima`, and calls `\vskip`. This, being a vertical mode command, terminates the current paragraph, and may start a new page. In this case, the command (with the arguments) is evaluated after the page is shipped out; in this case, `\@tempdima` is modified, it contains now Xpt, the text width), and the argument of `\vskip` is now 4pt plus 4pt minus Xpt. In one case, TEX found an optimum break point with 30% of glue set (meaning: the actual glue between the two items is 4pt minus one third of the text width, this is minus 125 pt, say ten lines). In the case there is a label, we handle it after the `\vskip`.

```
1630 \XMLelement{fo:list-item}
1631   {\SpaceAttributes}
1632   {
1633    \FOSetHyphenation
1634    \par
1635    \jg@usespacebefore{\vskip}% noskip if spacebefore is zero V2
1636 %      \ifdim\FOspacebefore=0pt\relax\else\vskip\FOspacebefore\fi
1637    \FOlabel}
1638   {}
```

An item is declared via `<fo:list-item-label>`. Question: why do we put the argument into `\ItemBox`? the box is never used! There is a similar code on line 1573. This is to correct a bug, see explanations below.

```
1639 \newsavebox\ItemBox
1640 \XMLelement{fo:list-item-label}
1641   {}
1642   {\xmlgrab}
1643   {\jg@hackindent
1644    \savebox{\ItemBox}{#1}\item[#1]\FOlabel}
```

This is called in the case where the current block is the label of a list. This computes some quantities (left margin, right margin, itemsep, etc). It is not clear whether or not these quantities need to be computed. Note that the last line of code defines `\makelabel`.

```
1645 \def\FOListBlock{%
1646    \FOSetFont{normal}%
1647    \get@external@font\xdef\FOlistlabelfont{\external@font}%
1648    \jg@usespacebefore{\itemsep}
1649    \jg@use@listparams
1650    \expandafter\csname List\FOtextalign\endcsname}
```

Bug explanation. The content of `<fo:list-item-label>` is evaluated twice, fake and real, see line 1644 (occurence A, fake), line 1573 (occurence B, fake), and line 1574 (occurrence C, real). We once thought it better to move the fake code from A to B (as near as possible to C); the trouble is the preceding like above: it defines globally `\makelabel`, according to the text-align attribute of the `<fo:block>` of the item. Since `\makelabel` is used to typeset this block, redefining it inside

C is useless (hence, without A, the first item in a list was justified according to the list, and each other was justified according to the previous one). While debugging, we found that the text-align attribute of the list item was 'justify', and `\Listjustify` was not defined. We added a definition, making it equivalent to 'center'.

This piece of code sets the switch `FOListInnerPar` to true if we are inside a list and this is not the first block. The `\relax` was missing in V1, as a consequence the test was evaluated before the assignment is complete; consequences of this are explained below. Also equals to one was replaced by less than two.

```
\def\jg@setlistinnerpar{%
        \ifx\XML@parent\FOListItemBody
          \global\advance\FOListBlocks by 1\relax%
          \ifnum\FOListBlocks<2\relax\else\FOListInnerPartrue\fi
        \else
          \ifFOListBody \FOListInnerPartrue \fi
        \fi}
```

We explain here a second bug, that occurs when dvi is produced instead of Pdf. If you look at lines 1620 and 1644, you will see two occurrences of `\FOlabel`; only one of these two will be used because `\FOlabel` kills `\FOid` (the command is defined in section 4.22). Consider the scenario where we have only a command at line 1644, and we generate a Pdf file; it will contain some glue and a penalty, then a `\write` with `\newlabel`, and a `\pdfdest` command (each followed by an infinite penalty), then `\parskip` and `\baselineskip` (let's hope that a page break does not occur here, but at the penalty mentioned above), then a paragraph, starting with the label. Now, it happens that adding a hyper anchor in a dvi file calls `\leavevmode` (via `\pdfmark` and `\pdf@rect`), and this flushes the label (the `\everypar` token list defined on line 1613 is executed, and the line 1620 is executed). This has as consequence that the dvi file contains: first some glue and a penalty, then the `\write` then `\parskip` and `\baselineskip`, then a paragraph starting with the label, followed by some `\special` commands.

Now, if we have a command at lines 1620 and 1644, the command on line 1620 is executed before the other one is complete, and the `\write` is executed twice. This means that the command should be used only on line 1620. Now the bug is that the first paragraph of the item is not considered as such, and a `\par` command is executed (line 1826). If the label is not flushed, this command does nothing, otherwise, it inserts a line break after the label; thus we have a different behavior in the Pdf and dvi file.

The body of the item is declared via `<fo:list-item-body>`. We say that we are in the body of the list, and start a counter with zero. This counter is (globally) incremented for each paragraph (i.e., `<fo:block>`) that is not the child of a list item label, nor in a static section, neither a table cell, but is the child of a list item body (see line 1653); the counter is local to the list item, hence must be saved in a local variable `\This@FOListBlocks`, and restored at the end of the element.

```
\XMLelement{fo:list-item-body}
  {}
  {\ifx\FOstartindent\att@bodystart \let\FOstartindent\z@  \fi
   \edef\This@FOListBlocks{\the\FOListBlocks}%
   \FOListBodytrue\global\FOListBlocks0}
  {\global\FOListBlocks\This@FOListBlocks\relax}
```

Default values for the attributes.

```
\gdef\FOstartindent{\z@}
\gdef\FOendindent{\z@}
\gdef\FOprovisionaldistancebetweenstarts{24.0pt}
\gdef\FOprovisionallabelseparation{6.0pt}
```

```
\newcount\FOListBlocks
\edef\This@FOListBlocks{\the\FOListBlocks}
```

4.14 Blocks

The `<fo:block>` is an important object.

We have seen cases where the switch `FOBlockGrab` has to be set to true. Here is another one: when the border style is solid or when the background is not transparent.

```
\def\jg@hack@background{%
     \ifx\FObackgroundcolor\att@transparent
        \ifx\FOborderstyle\att@solid \FOBlockGrabtrue \fi
     \else
         \FOBlockGrabtrue
     \fi}
```

This is the major addition in V2. If the current block should span the whole page, but the current page has more than one column, we stop and restart the multi-columns environment. Test of widows and orphans added. The documentation says: the orphans (widows) property specifies the minimum number of line-areas in the first (last) area generated by the formatting object; default value is 2. This code is correct if the value is one or two; better than that cannot be done in TeX.

```
\def\jg@block@span{%
  \ifx\FOspan\att@all
    \ifnum\NColumns>1\relax \interendmulticols \fi
  \fi
  \ifnum\FOwidows>1\relax\widowpenalty10000\relax\else\widowpenalty0\relax\fi
  \ifnum\FOorphans>1\relax\clubpenalty10000\relax\else\clubpenalty0\relax\fi}
```

Action at end of block.

```
\def\jg@endblock@span{%
  \ifx\FOspan\att@all
    \ifnum\NColumns>1\relax \interbeginmulticols{\NColumns}\fi
  \fi}
```

This is the action associated to a block.
We define two commands, `\w@t` and `\@whattodonext`, to be executed at the start of the element, and at the end. The first is followed by `\jg@keep@together`, the second is preceded by `\jg@keep@together`. This is a command that may evaluate a `\samepage` command. There are four cases to consider. Case 1: the parent is a list-item-label. In this case, we execute `\FOListBlock` and `\FOEndBlock`. Case 2: `FOInOutput` is true. The actions are `\FOOutputBlock` and `\FOEndOutputBlock`. Case 3: In a table. Actions are `\FOBoxedBlock` and `\FOEndBoxedBlock`. Case 4 (otherwise): actions are `\FONormalBlock` and `\FOendBlock`. Remember, when we typeset a static block (page headers), we are in case 2, and the parent is empty; can this element contain lists? this would give strange page headings.

```
\XMLelement{fo:block}
  {\SpaceAttributes}
  {%
  \jg@block@span
  \FOBlockGrabfalse
  \FOexpandattributes
  \FOSetHyphenation
  \ifx\XML@parent\FOListItemLabel
```

```
        \def\w@t{\FOListBlock}%
        \def\@whattodonext{\FOEndBlock}%
   \else
    \ifFOinOutput
      \jg@hack@background
      \def\w@t{\FOOutputBlock}%
      \def\@whattodonext{\FOEndOutputBlock}%
    \else
        \ifnum\FOinTable>1\relax
          \def\w@t{\FOBoxedBlock{\CurrentCellWidth}}%
          \def\@whattodonext{\FOEndBoxedBlock}
        \else
         \jg@setlistinnerpar
         \jg@hack@background
         \def\@whattodonext{\FOEndBlock}
         \def\w@t{\FONormalBlock}%
        \fi
     \fi
   \fi
   \w@t
   \jg@keep@together
   }
   {\jg@keep@together
   \@whattodonext
   \jg@endblock@span}
```

Declarations.

```
\XMLNSA{fo}{background-color}{\FObackgroundcolor}{transparent}
\XMLstringX\att@transparent<>transparent</>
\XMLstringX\att@solid<>solid</>
```

This piece of code is executed at the end of a block (except output, or boxed). We do nothing if this is a label of a list (see later). We emit a `\par` if this block is somewhere else in a list and `FOListInnerPar` is true. We call `\FOEndBlockTwo` in the case where the block is neither in a list nor in a table.

```
\def\FOEndBlock{%
  \ifx\XML@parent\FOListItemLabel
  \else
    \ifnum\FOinList>0
      \ifInInsertion\start@strut\fi
      \ifFOListInnerPar\unskip\par\fi
    \else
       \ifnum\FOTableNesting>0
       \else \FOEndBlockTwo  \fi
    \fi
  \fi}
```

This is the end of a normal block. We first terminate the paragraph. After that, we call `\FOEndBoxedBlock` if the block is boxed, or else add some vertical space and some padding. After that, we decide if a new page, or a new double page should be started here. We instantiate the switch `@tempswa` with a boolean value that says whether or not it is wise to start a new page here. If it is, we just insert some vertical space, otherwise, we insert a penalty, some vertical space, and call a command `\@afterheading` (a standard LaTeX command). Changes in V2: instead of adding

an infinite penalty, this calls `\addpenalty`, a LaTeX command that can do strange things (included signal a missing `\item` error). Normally it inserts a penalty, a large one here.

```
\def\FOEndBlockTwo{%
  \par
   \ifFOBlockGrab
       \FOEndBoxedBlock
   \else
        \ifdim\FOpaddingafter>\z@ \vskip\FOpaddingafter \fi
        \FOBorderBottom
   \fi
   \jg@handle@breakafter
   \jg@keepnext
   \if@tempswa\addpenalty{9993}\fi
   \FOvspaceafter
%   \if@tempswa\@afterheading\fi
}
\XMLstringX\att@oddpage<>odd-page</>
\XMLstringX\att@evenpage<>even-page</>
```

This considers the break-after attribute. The value can be one of 'auto', 'column', 'page', 'even-page' or 'odd-page'. The code was improved in V2, but still is strange. In particular the second `\cleardoublepage` seems ineffective. In my opinion, we should use a blank page or a double-blank page here.

```
\def\jg@handle@breakafter{%
   \ifx\FObreakafter\att@page
     \clearpage
   \else \ifx\FObreakafter\att@oddpage
      \ifodd\c@page\cleardoublepage\else\clearpage\fi
   \else \ifx\FObreakafter\att@evenpage
      \ifodd\c@page\clearpage\else\cleardoublepage\fi
   \fi\fi\fi
}
```

In principle, a break before a block should be interpreted in the same way as a break after. Note that here, we have a blank page.

```
\def\jg@handle@breakbefore{%
      \ifx\FObreakbefore\att@page
        \penalty -\@M
      \else
        \ifx\FObreakbefore\att@oddpage
        \penalty -\@M
        \ifodd\c@page\else\BlankPage\newpage\fi
        \fi
      \fi}
```

This is the start of an output block. It is not complicated. We call functions that specify how to align (justify) the text. If the switch `BlockGrab` is true, we put the text in a box with a given size. Note: We added `\@inlabelfalse` because the `\color` command says `\leavevmode` if this is true. This gives an empty line if a page break occurs because of an `\item`. We kill also `\FOid`. I'm not sure that this is useful, but the page head or foot should contain no anchor. Changes in V2: conditional part moved from before quadding to after quadding. Command `\Quadding` commented out.

```
\def\FOOutputBlock{%
 \FOSetFont{output}%
 \@inlabelfalse\let\FOId\@empty
 \QuaddingStart
 %\Quadding
 \ifFOBlockGrab \FOBoxedBlock{\textwidth}  \fi
}
```

The end code is easy. If we have opened a grab block, we must close it.

```
\def\FOEndOutputBlock{%
 \QuaddingEnd
 \ifFOBlockGrab \FOEndBoxedBlock \fi
 \par}
```

This is used at the start of a border block. If the border is solid and has non-zero width, we insert the `\hrule`.

```
\def\FOBorderTop{%
  \ifdim\FOborderbeforewidth>\z@
   \ifx\FOborderbeforestyle\att@solid
    {\color{\FObordercolor}\hrule height \FOborderbeforewidth}%
   \fi
 \fi}
```

This is used at the end of a border block. If the border is solid and has non-zero width, we insert the `\hrule`. Note: This may set the switch `FOBlockGrab` to true. This is very strange (isn't it too late?).

```
\def\FOBorderBottom{%
  \ifx\FOborderafterstyle\att@solid
   \ifx\FOborderafterwidth\att@thin\def\FOborderafterwidth{0.4pt}\fi
   \ifx\FOborderafterwidth\att@medium\def\FOborderafterwidth{0.8pt}\fi
   \ifx\FOborderafterwidth\att@thick\def\FOborderafterwidth{1.2pt}\fi
  \else
   \def\FOborderafterwidth{\z@}%
  \fi
  \ifx\FOborderbeforestyle\att@solid
   \ifx\FOborderbeforewidth\att@thin\def\FOborderbeforewidth{0.4pt}\fi
   \ifx\FOborderbeforewidth\att@medium\def\FOborderbeforewidth{0.8pt}\fi
   \ifx\FOborderbeforewidth\att@thick\def\FOborderbeforewidth{1.2pt}\fi
   \FOBlockGrabtrue
  \else
   \def\FOborderbeforewidth{\z@}%
  \fi
   \ifdim\FOborderafterwidth>\z@
    \ifx\FOborderafterstyle\att@solid
      {\color{\FObordercolor}\hrule height \FOborderafterwidth}%
    \fi
   \fi
}
```

This will be used later. It redefines the behavior of space and end-of-line. The value of white-space can be 'normal', 'pre', or 'nowrap'. Its effect is to specify some other properties; first white-space-treatment, that can be 'ignore', 'preserve', 'ignore-if-before-linefeed', 'ignore-if-after-linefeed', 'ignore-if-surrounding-linefeed'; then white-space-collapse that can be 'true' or 'false'; then linefeed-treatment can be 'ignore', 'preserve', 'treat-as-space', 'treat-as-zero-width-space'; finally,

wrap-option can be 'wrap' or 'no-wrap'. It is not clear to me if the code that follows has anything to do with the XSL/Format recommendations.

```
\def\jg@activew{%
   \ifx\FOwhitespace\att@pre\obeyspaces\obeylines\fi
   \ifx\FOwhitespacecollapse\att@false\obeyspaces\fi
   \ifx\FOwrapoption\att@nowrap\obeylines\fi}
```

We declare the attribute and the default value.

```
\XMLNSAX{fo}{wrap-option}{\FOwrapoption}{\inherit}
\XMLNSAX{fo}{white-space}{\FOwhitespace}{\inherit}
\XMLNSAX{fo}{white-space-collapse}{\FOwhitespacecollapse}{\inherit}
\XMLNSAX{fo}{orphans}{\FOorphans}{\inherit}
\XMLNSAX{fo}{widows}{\FOwidows}{\inherit}
\gdef\FOwrapoption{wrap}
\gdef\FOwhitespace{normal}
\gdef\FOwhitespacecollapse{true}
\XMLstringX\att@nowrap<>no-wrap</>
\gdef\FOwidows{2}
\gdef\FOorphans{2}
```

This is what we do in the case of a normal block. The code was a bit simplified: We removed a reference to `\@x` (always `\relax`), the code executed in case of tables in tables (this cannot happen), and the code that saves `\FOid` if a page is started here.

If we are in a list, three actions are taken: a) we evaluate the label, b) emit a `\par` and some vertical space (provided that this is allowed), and c) look at translation of spaces and new lines. In the general case, more actions are taken. If there is an attribute that says that we must change page we do it (this test was modified in V2). We emit a `\par`. If we must grab, we start a grab box. Otherwise we add some vertical space (note: a page break can occur here; in this case the label is on the wrong page; for this reason we moved the `\FOlabel` near the end). We define `\leftskip`, `\rightskip` and some other quantities. We execute action c) above.

Changes in V2: `\unskip` added before `\par`; `\ifInInsertion` test added; minus infinite penalty replaced by `\newpage`; test added: we change to normal font only if the parent is not `\FOFootnoteBody`.

```
\def\FONormalBlock{%
%  \ifnum\FOTableNesting>0
%  \else
    \ifnum\FOinList>0
      \FOlabel
      \ifFOListInnerPar\unskip\par\FOvspacebefore\fi
      \jg@activew
      \ifInInsertion\start@strut\fi
    \else
      \jg@handle@breakbefore
      \ifx\FObreakbefore\att@page
        \newpage
      \else
        \ifx\FObreakbefore\att@oddpage
          \newpage
          \ifodd\c@page\else\BlankPage\fi
        \else
           \ifx\FObreakbefore\att@evenpage
              \newpage
```

```
              \ifodd\c@page\BlankPage\fi
            \fi
          \fi
        \fi
        \par
%       \FOlabel JG
        \Quadding
        \ifFOBlockGrab
          \FOBoxedBlock{\linewidth}%
        \else
          \FOBorderTop
          \ifdim\FOpaddingbefore>\z@
           \vskip\FOpaddingbefore
          \fi
          \FOvspacebefore
          \parindent\FOtextindent
          \advance\leftskip by  \FOpaddingstart
          \advance\leftskip by  \FOmarginleft
          \advance\rightskip by \FOpaddingend
          \advance\rightskip by \FOmarginright
        \fi
        \jg@activew
      \fi
      \FOlabel %moved this [jg]
      \ifx\XML@parent\FOFootnoteBody\else\FOSetFont{normal}\fi
%   \fi
}
```

Attributes.

```
\XMLNSAX{fo}{break-before}{\FObreakbefore}{auto}
\XMLNSAX{fo}{break-after}{\FObreakafter}{auto}
```

4.15 Percentages

This command is a helper for percentage. If it is called with argument '`foo\relax%.\@{A}{B}`', and foo has no % character in it, then `#2` is a dot, the test is false, `B` is evaluated. If foo has the form `33.33%`, then the test is true, the number `33.33` is put in `\percentval` and `A` is evaluated.

```
{\catcode`\%=12
  \gdef\percenttest#1%#2#3\@{\ifx#2\relax
    \def\percentval{#1}\expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
```

We define here `\defpercentother`, a command that globally defines an active percent character to be `\percentother` that evaluates to a non-active percent character (in V1, the category code of the percent sign above was 13, and this trick was not needed). There is a little problem with this code: the old value of the percent character as an active character is never restored. In the case where the percent character was made active in order to typeset it, this piece of code is fine; otherwise, we might get unexpected results.

```
{\catcode`\%=12\relax
\gdef\percentother{%}
```

```
}
{\catcode`\%=13\relax
\gdef\defpercentother{\xdef%{\percentother}}
}
```

In version 1, percentages were computed using the first line shown here. In V2, we change the category code of the percent sign, so that the second line is used now.

```
%%\expandafter\percenttest#1\relax%.\@
%%\performpercent{#1}
```

The command `\performpercent` fully expands the argument as well as the percent sign. Result is put in a temporary command that is evaluated again.

```
\gdef\performpercent#1{
   \defpercentother
   \edef\dopercent{\noexpand\percenttest#1\relax%.\noexpand\@}
   \dopercent}
```

This takes an argument, a dimension or a percentage. In the case of a dimension, it is put in `\@tempdima`. Otherwise it is converted. Conversion is strange: Assume that the number is `33.33`. We put `33.33pt` in a skip register, divide this by 100, and use `\strip@pt`, a command that returns the value (without the unit, here 0.3333). This is then used as a multiplier for `\TableWidth`. The result is in `\SCALE`.

```
\gdef\TablePercentToDimen#1{\performpercent{#1}
  {\@tempdimb\percentval pt\relax\divide\@tempdimb by 100
   \edef\SCALE{\strip@pt\@tempdimb}\global\@tempdima
         =\SCALE\TableWidth}{\global\@tempdima#1}}
```

Same code, but we use `\hsize` instead of `\TableWidth`. The result is in `\SCALE`.

```
\gdef\PercentToDimen#1{\performpercent{#1}
  {\@tempdimb\percentval pt\relax\divide\@tempdimb by 100
   \edef\SCALE{\strip@pt\@tempdimb}\global\@tempdima
          =\SCALE\hsize}{\global\@tempdima#1}}
```

Same code, but we use `\Gin@nat@width`. We use the value of `\FOcontentwidth`. The result is in `\WSCALE`, is used for `\setkeys {Gin} {width=something}`.

```
\gdef\FOSetGWidth{\performpercent{\FOcontentwidth}
  {\@tempdima\percentval pt\relax\divide\@tempdima by 100
  \edef\WSCALE{\strip@pt\@tempdima}\setkeys{Gin}%
    {width=\WSCALE\Gin@nat@width}}{\setkeys{Gin}{width=\FOcontentwidth}}}
```

Same code, but we use `\Gin@nat@height` instead of `\TableWidth`. We use the value of `\FOcontentheight`. The result is in `\HSCALE`, used for `\setkeys{Gin} {height=something}`.

```
\gdef\FOSetGHeight{\performpercent{\FOcontentheight}
  {\@tempdima\percentval pt\relax\divide\@tempdima by 100
  \edef\HSCALE{\strip@pt\@tempdima}\setkeys{Gin}%
     {height=\HSCALE\Gin@nat@height}}{\setkeys{Gin}{height=\FOcontentheight}}}
```

Same idea. However, we divide `\f@size` by 100. The result is in `\FOfontsizefinal`.

```
\gdef\PlayWithFSize#1{\@default\f@size pt
  \performpercent{#1}
      {\dimen@0.01\@default
     \multiply\dimen@\percentval\relax}{\dimen@#1}%
     \edef\FOfontsizefinal{\the\dimen@}}
```

Same code, but we use `\baselineskip` instead of `\TableWidth`. Result in `\dimen@`.

```
\gdef\PlayWithShift{performpercent{\FOverticalalign}
      {\dimen@0.01\baselineskip\multiply\dimen@\percentval\relax}%
   {\dimen@\FOverticalalign}}
%
```

4.16 Fonts

We show here some commands that deal with fonts. This is bit longish... This defines some values for the sizes.

```
\expandafter\def\csname size-xx-small\endcsname{7pt}
\expandafter\def\csname size-x-small\endcsname{8pt}
\expandafter\def\csname size-small\endcsname{9pt}
\expandafter\def\csname size-medium\endcsname{10pt}
\expandafter\def\csname size-large\endcsname{14.4pt}
\expandafter\def\csname size-x-large\endcsname{18pt}
\expandafter\def\csname size-xx-large\endcsname{20pt}
```

This puts in `\FOfontsizefinal` a size (could be 10pt, 100% or medium).

```
\def\computeFOfontsize{%
  \expandafter\ifx\csname size-\FOfontsize\endcsname\relax
    \PlayWithFSize\FOfontsize
  \else
    \edef\FOfontsizefinal{\csname size-\FOfontsize\endcsname}%
  \fi}
```

The mlnames.tex file defines `\Family@foo`, for different values of `foo`, for instance monospace, sansserif, serif, or Arial, Helvetica, or Avant-Garde, New-Century-Schoolbook, or cmr, cmss, etc.

```
\def\FOSetFont#1{%
 \FOSetHyphenation
 \edef\LaTeXshape{\csname Width@\FOfontstretch\endcsname
      \csname Weight@\FOfontweight\endcsname}%
 \ifx\LaTeXshape\@empty\def\LaTeXshape{m}\fi
 \edef\fFamName{\FOfontfamily}%
 \edef\f@series{\LaTeXshape}%
 \edef\f@shape{\csname Posture@\FOfontstyle\endcsname}%
 \ifx\FOfontvariant\att@smallcaps \def\f@shape{sc}\fi
 \let\f@family\relax
 \@for\FOfoo:=\FOfontfamily\do{%
    \ifx\f@family\relax
    \expandafter\let\expandafter\f@family
         \csname Family@\FOfoo\endcsname
    \fi}%
    \ifx\f@family\relax
      \def\f@family{\csname Family@\Defaultx@fontfamily\endcsname}%
    \fi
 \FOSetFontSize
 \selectfont
 \ifx\FOcolor\@empty \else \color{\FOcolor}\fi
}
```

This command computes some parameters for font changing. If no line-height is given, we use the font size plus twenty percent.

```
\def\FOSetFontSize{%
 \computeFOfontsize
 \ifx\FOlineheight\att@normal
   \@tempdima\FOfontsizefinal
   \multiply\@tempdima by 12
   \divide\@tempdima by 10
   \set@fontsize\baselinestretch{\FOfontsizefinal}{\@tempdima}%
 \else
   \set@fontsize\baselinestretch{\FOfontsizefinal}{\FOlineheight}%
 \fi}
```

Same code as above. However `\set@fontsize` is not executed, but saved, as well as its argument in the command `\FORestoreFontSize`.

```
\def\FOSaveFontSize{%
 \computeFOfontsize
 \ifx\FOlineheight\att@normal
   \@tempdima\FOfontsizefinal
   \multiply\@tempdima by 12
   \divide\@tempdima by 10
   \xdef\FORestoreFontSize{\noexpand\set@fontsize
       \noexpand\baselinestretch{\FOfontsizefinal}{\the\@tempdima}}%
 \else
   \xdef\FORestoreFontSize{\noexpand\set@fontsize
       \noexpand\baselinestretch{\FOfontsizefinal}{\FOlineheight}}%
 \fi
}
```

Declarations and attributes.

```
\XMLNSAX{fo}{line-height}{\FOlineheight}{\inherit}
\XMLNSAX{fo}{font-weight}{\FOfontweight}{\inherit}
\XMLNSAX{fo}{font}{\FOfont}{\inherit}
\XMLNSAX{fo}{font-family}{\FOfontfamily}{\inherit}
\XMLNSA{fo}{font-size}{\FOfontsize}{\inherit}
\XMLNSAX{fo}{font-size-adjust}{\FOfontsizeadjust}{\inherit}
\XMLNSAX{fo}{font-stretch}{\FOfontstretch}{\inherit}
\XMLNSAX{fo}{font-style}{\FOfontstyle}{\inherit}
\XMLNSAX{fo}{font-variant}{\FOfontvariant}{\inherit}
\gdef\FOlineheight{normal}
\gdef\FOfontweight{normal}
\gdef\Defaultx@fontfamily{Times-Roman}
\gdef\FOfont{}
\gdef\FOfontfamily{Times-Roman}
\gdef\FOfontsizeadjust{none}
\gdef\FOfontsize{medium}
\gdef\FOfontstretch{normal}
\gdef\FOfontstyle{normal}
\gdef\FOfontvariant{normal}
\XMLstringX\att@smallcaps<>small-caps</>
```

All definitions in the remaining of this section come from mlnames.sty. We start with the Family.

```
\def\Family@monospace{pcr}      \def\Family@sansserif{phv}
\expandafter\def\csname Family@sans-serif\endcsname{phv}
\def\Family@serif{ptm}          \def\Family@cursive{uzc}
```

```
\def\Family@fantasy{uzc}       \def\Family@unknown{<unknown>}
\def\Family@Arial{phv}         \def\Family@Helvetica{phv}
\def\Family@Palatino{ppl}      \def\Family@Bookman{pbk}
\def\Family@BaskervilleMT{mbv} \def\Family@Courier{pcr}
\def\Family@Symbol{psy}        \def\Family@Wingdings{pzd}
\def\Family@WingDings{pzd}     \def\Family@LucidaSans{hls}
\def\Family@LucidaBright{hlh}  \def\Family@LucidaTypewriter{hlst}
\def\Family@Savoy{usb}         \def\Family@Luxi{ul9}
\def\Family@ACaslon{pca}       \def\Family@Caslon{uca}
\def\Family@Formata{pfa}       \def\Family@FranklinGothic{pfg}
\def\Family@OCRAbyBT{boa}      \def\Family@AGaramond{pad}
\expandafter\def\csname Family@Avant-Garde\endcsname{pag}
\expandafter\def\csname Family@Courier New\endcsname{pcr}
\expandafter\def\csname Family@New-Century-Schoolbook\endcsname{pnc}
\expandafter\def\csname Family@Times-Roman\endcsname{ptm}
\expandafter\def\csname Family@Trade-Gothic\endcsname{ptg}
\expandafter\def\csname Family@Times-New-Roman\endcsname{ptm}
\expandafter\def\csname Family@Times New Roman\endcsname{ptm}
\expandafter\def\csname Family@Times Roman\endcsname{ptm}
\expandafter\def\csname Family@Times-NR-MT\endcsname{mnt}
\expandafter\def\csname Family@Courier-New\endcsname{pcr}
\expandafter\def\csname Family@Zapf-Dingbats\endcsname{pzd}
\expandafter\def\csname Family@Gill-Sans\endcsname{pgs}
\expandafter\def\csname Family@iso-serif\endcsname{ptm}
\expandafter\def\csname Family@sans-serif\endcsname{phv}
\expandafter\def\csname Family@iso-sanserif\endcsname{phv}
\expandafter\def\csname Family@iso-monospace\endcsname{pcr}
\expandafter\def\csname Family@LetterGothic12PitchBT\endcsname{blg}
\expandafter\def\csname Family@NewsGothic\endcsname{bng}
\expandafter\def\csname Family@NewsGothicBT\endcsname{bng}
\expandafter\def\csname Family@Humanist521\endcsname{bgs}
\expandafter\def\csname Family@Humanist521BT\endcsname{bgs}
\expandafter\def\csname Family@Monospace821\endcsname{bhvt}
\expandafter\def\csname Family@Monospace821BT\endcsname{bhvt}
\expandafter\def\csname Family@OCRB10PitchBT\endcsname{bob}
\expandafter\def\csname Family@OCR-A\endcsname{boa}
\expandafter\def\csname Family@OCR-B-10PitchBT\endcsname{bob}
\expandafter\def\csname Family@Computer-Modern-Typewriter\endcsname{aett}
\expandafter\def\csname Family@Computer-Modern-Sans\endcsname{aess}
\expandafter\def\csname Family@Computer-Modern\endcsname{aer}
\expandafter\def\csname Family@Computer-Modern-Caps-And-Small-Caps\endcsname
{cmcsc}
\def\Family@cmr{cmr}              \def\Family@cmss{cmss}
\def\Family@cmtt{cmtt}            \def\Family@cmcsc{cmcsc}
\def\Family@ectt{ectt}            \def\Family@Utopia{put}
\def\Family@ZapfChancery{pzc}     \def\Family@Fibonacci{cmfib}
\def\Family@Funny{cmfr}           \def\Family@Dunhill{cmdh}
\def\Family@Concrete{ccr}         \def\Family@Charter{bch}
\def\Family@Fontpxr{pxr}          \def\Family@Fontaer{aer}
\def\Family@Fontaess{aess}        \def\Family@Fontaett{aett}
\def\Family@Fontlcmss{lcmss}      \def\Family@Fontlcmtt{lcmtt}
\def\Family@Fontcmvtt{cmvtt}      \def\Family@Fontcmbr{cmbr}
\def\Family@Fontcmtl{cmtl}        \def\Family@Fontpxss{pxss}
\def\Family@Fonttxss{txss}        \def\Family@Fonttxr{txr}
```

We define some postures.

```
\def\Posture@upright{n}
\def\Posture@normal{n}
\def\Posture@math{it}
\def\Posture@oblique{sl}
\def\Posture@backslantedoblique{ui}
\def\Posture@italic{it}
\def\Posture@backslanteditalic{ui}
```

We define some weights.

```
\def\Weight@ultralight{ul}
\def\Weight@extralight{el}
\def\Weight@light{l}
\def\Weight@semilight{sl}
\def\Weight@medium{}
\def\Weight@normal{}
\def\Weight@semibold{sb}
\def\Weight@bold{bx}
\def\Weight@extrabold{eb}
\def\Weight@ultrabold{ub}
\def\Weight@false{}
```

We define some widths.

```
\expandafter\def\csname Width@ultra-condensed\endcsname{uc}
\expandafter\def\csname Width@extra-condensed\endcsname{ec}
\expandafter\def\csname Width@condensed\endcsname{c}
\expandafter\def\csname Width@semi-condensed\endcsname{sc}
\expandafter\def\csname Width@normal\endcsname{}
\expandafter\def\csname Width@semi-expanded\endcsname{sx}
\expandafter\def\csname Width@expanded\endcsname{x}
\expandafter\def\csname Width@extra-expanded\endcsname{ex}
\expandafter\def\csname Width@ultra-expanded\endcsname{ux}
\def\Width@ultracondensed{uc}
\def\Width@extracondensed{ec}
\def\Width@condensed{c}
\def\Width@semicondensed{sc}
\def\Width@medium{}
\def\Width@semiexpanded{sx}
\def\Width@expanded{x}
\def\Width@extraexpanded{ex}
\def\Width@ultraexpanded{ux}
```

4.17 Links

The XSL/Format standard defines three attributes: target-presentation-context, target-processing-context, and target-stylesheet, but they are ignored. The code make use of a variable that is not defined. We removed the test. I don't understand these two `\let\xx\empty`, hence commented them out. Note: the code was modified in January 2007, using `\url` to typeset the argument; this removes overful hboxes for long URLs; however the `\urlstyle` has be to changed to 'same', so as to use the current font, and the url package has to be loaded with the 'obeyspaces' option, so as to keep spaces.

```
\XMLelement{fo:basic-link}
  { }
  {\xmlgrab}
  {
```

```
  \ifx\FOverticalalign\att@auto
     \let\FOverticalalign\FObaselineshift
  \fi
   \protectCS{\FOinternaldestination}%
   \protectCS{\FOexternaldestination}%
   \ifx\FOexternaldestination\@empty
    \hyperlink{\FOinternaldestination}{\FO@inlinesequence{#1}}%
    %\let\FOinternaldestination\@empty  %% JG
   \else
      \href{\FOexternaldestination}{\FO@inlinesequence{#1}}
    %\let\FOexternaldestination\@empty  %% JG
    \fi
  }
```

These are the attributes used above.

```
\XMLNSAX{fo}{external-destination}{\FOexternaldestination}{}
\XMLNSAX{fo}{internal-destination}{\FOinternaldestination}{}
```

4.18 Footnotes

Case of `<fo:footnote>`*mark text*`</fo:footnote>`. The specifications say that the first child is a `<fo:inline>` formatting object that gives the citation, and the second is a `<fo:footnote-body>`, containing the body of the note. In version V2, we handle the case where the footnote is in a table: since cells are evaluated twice, the command `\FOfoottext` has a meaning depending on the context. New in V2: `\FOplainfoottext` renamed to `\FOfoottext`.

```
\XMLelement{fo:footnote}
  {}
  {\xmlgrab}
  {\FOSaveFontSize\xmltextwochildren\FOplainfootmark\FOfoottext#1}
```

This is transparent.

```
\XMLelement{fo:footnote-body}
  {}  {}   {}
\XMLname{fo:footnote-body}{\FOFootnoteBody}
```

This saves some parameters. All these `\global` prefixes are useless. Note that we move the content of the `\@labels` box, we do not copy it (said otherwise, the box is empty while typesetting the footnote). All these parameters have something to do with the `\item` command, whose code is shown above, so that I wonder if the easier solution would not be a complete rewrite of the `\item` command, without all these ugly hacks.

```
\def\jg@save@footnote{%
  \xdef\Sav@FOListBlocks{\the\FOListBlocks}
  \global\let\sav@if@inlabel\if@inlabel
  \global\let\sav@if@nobreak\if@nobreak
  \global\let\sav@if@newlist\if@newlist
  \global\setbox\sav@labels\box\@labels
  \expandafter\global\expandafter\sav@everypar\expandafter{\the\everypar}
}
```

This restores the parameters saved above. Question: is it safe to restore `\everypar`? Is it wise to do it globally?

```
\def\jg@restore@footnote{%
  \global\FOListBlocks\Sav@FOListBlocks\relax
```

```
  \global\let\if@inlabel\sav@if@inlabel
  \global\let\if@nobreak\sav@if@nobreak
  \global\let\if@newlist\sav@if@newlist
  \global\setbox\@labels\box\sav@labels
  \expandafter\global\expandafter\everypar\expandafter{\the\sav@everypar}
}
```

This modifies some commands that are saved above. The `\relax` tokens are useless. Same question as above for `\everypar`. Is global modification needed?

```
\def\jg@set@footnote{%
  \FOListBlocks0\relax
  \global\everypar{}\relax
  \FOinList0\relax
  \FOListBodyfalse
  \InInsertiontrue
}
```

The command that typesets footnotes is a modification of the standard LaTeX command, shown here. The command `\@makefntext` is defined in a class file, it should typeset the argument, plus the mark found in `\@makefnmark`. There are two major changes in the fotex implementation: no label is defined, and the mark is not printed (the text should start with a copy of the mark).

```
\long\def\@footnotetext#1{\insert\footins{%
    \reset@font\footnotesize
    \interlinepenalty\interfootnotelinepenalty
    \splittopskip\footnotesep
    \splitmaxdepth \dp\strutbox \floatingpenalty \@MM
    \hsize\columnwidth \@parboxrestore
    \protected@edef\@currentlabel{%
       \csname p@footnote\endcsname\@thefnmark
    }%
    \color@begingroup
      \@makefntext{%
        \rule\z@\footnotesep\ignorespaces#1\@finalstrut\strutbox}%
    \color@endgroup}}%
```

This is now the code.

```
\long\def\FOplainfoottext#1{%
  \insert\footins{\relax
    \reset@font\footnotesize
    \FORestoreFontSize
    \size@update
    \interlinepenalty\interfootnotelinepenalty
    \splittopskip0pt\relax
    \splitmaxdepth \dp\strutbox \floatingpenalty \@MM
    \hsize\columnwidth\@parboxrestore
    \color@begingroup
    \jg@save@footnote
    \jg@set@footnote
    #1%
    \ifhmode\nobreak\fi
    \jg@restore@footnote \relax
    \color@endgroup}%
}
```

We define here two other commands. The first one is used in the first pass of a table: the text of the footnote is ignored. The other one is used for the second pass, where the text and the typesetting command is put in a token register. Note: we have seen in section 2.7 how to add tokens at the end of a token list `\T`. The idea is to put the value of the token list in a command `\t` defined via `\edef`, then to fill `\T` with `\t` and additional material (for instance `\W{#1}`). For some strange reason, the command `\W` is inserted in `\t`, hence must be preceded by `\noexpand`.

```
\def\FOnofoottext#1{}

\def\FOboxedfoottext#1{
  \edef\boxedfootnotetext{\the\BoxedFootnotes\noexpand\FOplainfoottext}%
  \global\BoxedFootnotes=\expandafter{\boxedfootnotetext{#1}}%
}
```

Useful function. It inserts a vertical rule, whose vertical dimensions are those of the `\strutbox`.

```
\def\start@strut{%
  \vrule height \ht\strutbox depth \dp\strutbox width \z@\relax
}
```

Bootstrap code. We declare a token list that contain the copy of the `\everypar`, a box that contains a copy of the unprocessed item labels. The command with a long name holds the name of a XSL region: the one between normal text and a footnote.

```
\let\FOfoottext\FOplainfoottext
\newtoks\sav@everypar
\newbox\sav@labels
\XMLstringX\att@xsl@footnote@separator<>xsl-footnote-separator</>
```

Changes in V2: the command `\@makecol` was redefined, without the line that sets `\@elt` to `\relax`. This is strange.

4.19 Inline material

Handling of `<fo:inline` *`att`*`>`*`body`*`</fo:inline>`. We call one of two alternatives, depending on whether the border is solid or not. As usual, we evaluate 'thin', 'thick' or 'medium'.

```
\XMLelement{fo:inline}
  {}
  {\xmlgrab}
  {
  \ifx\FOverticalalign\att@auto \let\FOverticalalign\FObaselineshift \fi
  \FOlabel
  \ifx\FOborderstyle\att@solid
      \interpretwidth
      \FOboxedsequence{#1}%
  \else
      \FO@inlinesequence{#1}%
  \fi}
```

Helper function for letter spacing added in V2. The recommendation says that the attribute letter-spacing is either 'normal', a ⟨length⟩ or a ⟨space⟩ (this is something like a TEX glue, with conditionality and precedence). This handles only the case of a length, using the `\sodef` command of the soul package. Note that the last three arguments of the command are glues (the first one is a font switch, unless empty), so that the case of ⟨space⟩ could be implemented. The last two parameters define inter-word spacing. Are the values given here the good ones? The test seems strange to me: what if the command is already defined with a different value of the attribute?

```
\def\jg@so#1{%
   \ifx\FOletterspacing\att@normal
       \def\pre@sequence{{#1}}%
   \else
       \def\pre@sequence{%
           \@ifundefined{thisso}
              {\sodef\thisso{}{\FOletterspacing}{.4em}{.5em}}%
              {}
           {\thisso{#1}}}%
   \fi
}
```

Second helper function. This applies a function depending on `\FOtextdecoration` to the `\FOlabel` and text computed above.

```
\jg@apply@deco{%
  \csname DECO@\FOtextdecoration\endcsname{\FOlabel\pre@sequence}
```

Normal (non-boxed) inline sequence. We take into account three attributes: letter spacing, decoration, and vertical alignment. The alignment can be 'baseline', this is the normal behavior; it can be 'super' or 'sub', case where we raise or lower using special command; otherwise alignment is a dimension (or a percentage), and we use `\raisebox` to raise the box. Other possibilities, not yet implemented, are 'middle' (equivalent of `\vcenter` in TeX), 'text-top', 'text-bottom', 'top', or 'bottom'.

We use `\csname` for the decoration. Question: do we really need the `\FOlabel` command after the `\csname`? Maybe we could put it *after* the last `\fi`.

```
\def\FO@inlinesequence#1{%
 \FOSetFont{normal}%
 \jg@so
 \ifx\FOverticalalign\att@baseline
   \jg@apply@deco
 \else \ifx\FOverticalalign\att@super
    \textsuperscript{\jg@apply@deco}}%
 \else \ifx\FOverticalalign\att@sub
      \textsubscript{\jg@apply@deco}}%
 \else
   \PlayWithShift
   \raisebox{\dimen@}{\jg@apply@deco}%
 \fi\fi\fi
}
```

The original code uses two booleans `\ifFOSuper` and `\ifFOSub`. They are never defined, hence we removed the code that uses them. However, the code should be the same as before, with a frame. So, is this a bug? Letter spacing should be added here also.

```
\def\FOboxedsequence#1{%
  \FOSetFont{normal}%
  \ifx\FOborderwidth\@empty\else\interpretwidth\fboxrule\FOborderwidth\fi
  \ifx\FOverticalalign\att@baseline
     \fbox{\csname DECO@\FOtextdecoration\endcsname{\FOlabel#1}}%
  \else
       \PlayWithShift \fbox{\raisebox{\dimen@}{\FOlabel#1}}%
  \fi}
```

This declares the attribute.

```
\XMLNSAX{fo}{text-decoration}{\FOtextdecoration}{none}
\XMLNSAX{fo}{baseline-shift}{\FObaselineshift}{baseline}
\XMLstringX\att@sub<>sub</>
\XMLstringX\att@super<>super</>
\XMLstringX\att@baseline<>baseline</>
```

The following decorations are known. Notice the first: if no decoration is given, the argument is typeset, without the braces. The recommendation says that, for each foo in 'blink', 'underline', 'overline', 'strike-out', there is also no-foo; this is not implemented.

```
\def\DECO@{\@firstofone}
\def\DECO@blink{\uwave}
\def\DECO@underline{\uline}
\expandafter\def\csname DECO@line-through\endcsname{\sout}
\def\DECO@overline{\oline}
\def\oline#1{$\overline{\mbox{#1}}$}
```

4.20 Floats and images

Support for .gif format. Is this needed?

```
\g@addto@macro\Gin@extensions{,.gif}
\@namedef{Gin@rule@.gif}#1{{png}{.png}{`giftopng #1}}
```

The fotex file defines Gin/scale in the same way as graphicx.sty. We do not repeat the code here.

```
\define@key{Gin}{scale}{ ... }
```

Comment by S. Rahtz: "Taken from Heiko Oberdiek's epstopdf.sty but the redefinitions need to be global". This allows on the fly conversions from one image type to another. For instance, this may call `giftopng`.

```
\global\let\orgGin@setfile\Gin@setfile
\global\def\Gin@setfile#1#2#3{%
  \if`\@car #3\relax\@nil
    \let\Gin@base\filename@base
    \immediate\write18{\@cdr #3\@empty\@nil}%
    \orgGin@setfile{#1}{#2}{\filename@base #2}%
  \else
    \orgGin@setfile{#1}{#2}{#3}%
  \fi}
```

This is trivial.

```
\XMLelement{fo:float}
  {\XMLattributeX{float}{\FOfloat}{float}}
  {\ifx\FOfloat\att@none \begin{figure}[!htp] \else  \begin{figure} \fi
   \FOlabel}
  {\end{figure}}
```

The following is a bit more complicated. One problem is that we have to merge content-height and height, the same for the width. Question: is this the desired result? In order to make the code easier to understand, we have added three auxiliary functions. The first function sets `\FOcontentheight` and `\FOcontentwidth` to the values of `\FOheight` and `\FOwidth`, unless both values are 'auto'.

```
\def\jg@merge@aux{%
    \ifx\FOwidth\att@auto
```

```
      \ifx\FOheight\att@auto        \relax
      \else
        \def\FOcontentheight{\FOheight}%
        \def\FOcontentwidth{auto}%
       \fi
    \else
      \def\FOcontentwidth{\FOwidth}%
      \ifx\FOheight\att@auto
        \def\FOcontentheight{auto}%
      \else
        \def\FOcontentheight{\FOheight}%
      \fi
    \fi
}
```

These commands perform a non trivial action if the value of the attribute is not 'auto'. It can be 'scale-to-fit' or a dimension, or percentage (using code shown above).

```
\def\jg@mergeW{%
    \ifx\FOcontentwidth\att@scaletofit
      \setkeys{Gin}{width=\hsize}%
    \else
      \ifx\FOcontentwidth\att@auto\else\FOSetGWidth\fi
    \fi
}
\def\jg@mergeH{%
    \ifx\FOcontentheight\att@scaletofit
       \setkeys{Gin}{height=\vsize}%
    \else
       \ifx\FOcontentheight\att@auto\else\FOSetGHeight\fi
    \fi
}
```

This piece of code considers text-align but we could also take into account display-align, that can be 'before', 'after' or 'center' (as well as 'auto'). Note that text-align can be 'start' or 'end' as well.

If scaling is 'uniform', the aspect ratio should be preserved; is the default value of the Gin variable true?

```
\XMLelement{fo:external-graphic}
   {
      \XMLattributeX{content-height}{\FOcontentheight}{auto}
      \XMLattributeX{content-width}{\FOcontentwidth}{auto}
   }
   {
    \jg@filetest
    \jg@merge@aux
    \jg@mergeW
    \jg@mergeH
    \def\aligntype{center}
    \ifthenelse{\equal{\FOtextalign}{right}}{\def\aligntype{flushright}}{}
    \ifthenelse{\equal{\FOtextalign}{left}}{\def\aligntype{flushleft}}{}
    \def\Picscaled{\begin{\aligntype}%
       \includegraphics{\FOsrcname}\end{\aligntype}}
    \ifx\FOscaling\att@uniform\else\setkeys{Gin}{keepaspectratio=false}\fi
```

```
2283      \ifx\FOborderstyle\att@solid\fbox{\Picscaled}\else\Picscaled\fi
2284      }
2285      {}
```

Attributes used here.

```
2286 \XMLNSAX{fo}{scaling}{\FOscaling}{uniform}
2287 \XMLNSAX{fo}{height}{\FOheight}{auto}
2288 \XMLstringX\att@scaletofit<>scale-to-fit</>
2289 \XMLstringX\att@uniform<>uniform</>
```

This makes the string canonical. This is used in some places; it has an alternate name elsewhere.

```
2290 \def\protectCS#1{%
2291  \begingroup
2292        \utfeight@protect@chars
2293        \xdef\FOtempCS{#1}%
2294  \endgroup
2295  \let#1\FOtempCS}%
```

This function does something if the argument starts with 'url'. Otherwise, it removes a prefix of the form '`file://`' or '`file:`'. The result is in `\FOsrcname`.

```
2296 \def\FOfiletest#1#2#3#4#5#6#7#8\@{%
2297   \def\@tempa{#1#2#3#4#5#6#7}%
2298   \def\@tempb{#1#2#3#4#5}%
2299   \def\@tempc{#1#2#3#4}%
2300   \ifx\@tempa\file@prefix
2301     \xdef\FOsrcname{#8}%
2302   \else
2303    \ifx\@tempb\file@shortprefix
2304       \xdef\FOsrcname{#6#7#8}%
2305    \else
2306     \ifx\@tempc\file@urlprefix
2307     %%   \expandafter\FOurlfiletest#5#6#7#8%
2308     %% \@empty\@empty\@empty\@empty\@empty\@empty\@empty\@empty\@empty\@empty
2309     \jg@filetestaux#5#6#7#8)\@
2310     \else
2311       \xdef\FOsrcname{#1#2#3#4#5#6#7#8}%
2312     \fi  \fi  \fi}
```

These lines were added instead of lines 2307 and 2208. Guess why.

```
2313 \def\jg@filetestaux#1)#2\@{%
2314   \FOurlfiletest#1\@empty\@empty\@empty\@empty\@empty\@empty\@empty\@empty)}
```

Version with all the arguments.

```
2315 \def\jg@filetest{%
2316    {\utfeight@protect@chars
2317     \expandafter\FOfiletest\FOsrc\@empty
2318        \@empty\@empty\@empty\@empty\@empty\@empty\@}}
```

This function puts foo in `\FOsrcname` if the argument is `url(file://foo)` or `url(file:foo)` or `url(foo)` (the prefix '`url(`' has already been removed).

```
2319 \def\FOurlfiletest#1#2#3#4#5#6#7#8){%
2320   \def\@tempa{#1#2#3#4#5#6#7}%
2321   \def\@tempb{#1#2#3#4#5}%
2322   \ifx\file@prefix\@tempa
2323     \xdef\FOsrcname{#8}%
```

```
  \else
   \ifx\@tempb\file@shortprefix
      \xdef\FOsrcname{#6#7#8}%
   \else
      \xdef\FOsrcname{#1#2#3#4#5#6#7#8}%
   \fi \fi}
```

These are the strings needed by the function above.

```
\XMLstring\file@urlprefix<>url(</>
\XMLstring\file@prefix<>file://</>
\XMLstring\file@shortprefix<>file:</>
```

4.21 Markers

We consider here a list of the form X1Y1X2Y2...XnYn. The list ends with Xn being `\relax`. This command puts in the token list `\toks@` all pairs `{Xi}{Yi}` for which `Xi` is not equal to the value of the attribute marker-class-name. The name of the command is misleading; after this command has been evaluated, we are ready to add a marker.

```
\gdef\FOaddmarker#1#2{%
 \ifx\relax#1
 \else
   \def\FOtemp{#1}%
   \ifx\FOtemp\FOmarkerclassname
   \else \toks@\expandafter{\the\toks@{#1}{#2}}  \fi
   \expandafter\FOaddmarker
 \fi}
```

Handling of `<fo:marker marker-class-name=`*`X`*`>`*`mark`*`</fo:marker>`. We consider the content of `\FOmarks`, apply the command defined above, then add a new pair at the end; the pair is defined by *`X`* and *`mark`*. We copy the token list into the `\FOmarks` command (using full expansion, assignment is global), and then put it in a TEX mark.

```
\XMLelement{fo:marker}
 {}
 {\xmlgrab}
 {\toks@{}%
  \expandafter\FOaddmarker\FOmarks\relax{}%
  \toks@\expandafter{\the\expandafter\toks@\expandafter{\FOmarkerclassname}{#1}}%
  \xdef\FOmarks{\the\toks@}%
  \mark{\FOmarks}}
```

Here we assume that the list is terminated by a double `\relax`. The command finds and typesets the value associated to `\FOthisretrieveclassname`. In fact, we assume that `Xi` is in `#1`, `Yi` in `#2`. If the first argument is `\relax`, this is the end of the list, and we are done. If it is not the desired marker, we continue. Otherwise, we call a command that reads everything up to the double `\relax` at high speed. This will first read `#2`, this is `Yi` with a pair of additional braces that will disappear as argument `#1` of the next routine. It will then read `\fi`, two tokens, a second `\fi` and the list. The two `\fi` tokens are read again.

```
\gdef\FOgetmarker#1#2{%
  \ifx\relax#1
  \else
   \def\FOtemp{#1}%
    \ifx\FOtemp\FOthisretrieveclassname
```

```
      \FOmarkergobble{#2}%
     \fi
   \expandafter\FOgetmarker
  \fi}
\gdef\FOmarkergobble#1#2\relax\relax{\fi\fi#1}
```

Handling of `<fo:retrieve-marker retrieve-class-name=X retrieve-position=Y/>`. We fetch one of the marks (`\firstmark` or `\botmark`), and extract what is needed. Changes in V2: commands `\FirstOnPage` and `\LastOnPage` replaced by others with longer names, same value; `\topmark` replaced by `\firstmark`

```
\XMLelement{fo:retrieve-marker}
 {}
 {\xmlgrab}
 {\begingroup
    \utfeight@protect@chars
    \ifx\FOretrieveposition\att@first@starting@within@page
       \xdef\FOthismark{\firstmark}%
    \else \ifx\FOretrieveposition\att@last@starting@within@page
       \xdef\FOthismark{\botmark}%
    \else
       \xdef\FOthismark{\firstmark}%
    \fi \fi
   \xdef\FOthisretrieveclassname{\FOretrieveclassname}
   \endgroup
   \expandafter\FOgetmarker\FOthismark\relax\relax}
```

Constants and attributes used in the code above.

```
\XMLNSAX{fo}{retrieve-class-name}{\FOretrieveclassname}{}
\XMLNSAX{fo}{retrieve-position}
     {\FOretrieveposition}{first-starting-within-page}
\XMLNSAX{fo}{marker-class-name}{\FOmarkerclassname}{}
\XMLstring\FirstOnPage<>first-starting-within-page</>
\XMLstringX\att@first@starting@within@page<>first-starting-within-page</>
\XMLstringX\att@last@starting@within@page<>last-starting-within-page</>
\XMLstring\LastOnPage<>last-starting-within-page</>
\gdef\FOmarks{}
```

4.22 Page numbers and anchors

The command `\FOlabel` will be explained later, it defines the anchor associated to a label A. This defines two quantities: a static and a dynamic one. If a LaTeX file contains a `\pageref` command, associated to a label A, then the printed document will contain some number, say 17, we call this the static quantity associated to the label; this number is written in the auxiliary file, and read at the start of the job, allowing forward references. If you read the document using a computer (in dvi, PostScript, or Pdf format), it can happen that the region containing the number is active, and you can click on it. What happens exactly depends on the software used, and is controlled by the `\hyperref` package; we call this the dynamic part. It happens that the static part is formed of two token lists in standard LaTeX, five lists when using the hyperref package. These are in the command `\r@A`. The `\pageref` command extracts the second item in the list.

The XSL/Format equivalent of `\pageref` is a `<fo:page-number-citation>` defining the static part, in a `<fo:basic-link>` defining the dynamic part. These elements have an attribute X and Y, containing the identification of the anchor. The attribute names are different, but the values

are the same. The code shown here makes the assumption that the static part is used only for getting the correct page number. Hence, all fields but the second are empty, and all we need to do is evaluate the whole thing, it is in `\r@A` for the anchor A.

```
\XMLelement{fo:page-number-citation}
   {\XMLattributeX{ref-id}{\FOrefid}{}}
   {}
   {\fopagecitation}
%\def\fopagecitation{\pageref{\FOrefid}}
\def\fopagecitation{\csname r@\FOrefid\endcsname}
```

Handling of `<fotex:sort>...</fotex:sort>`. The content is a sequence of page-number-citation elements; we want to typeset the numbers, in increasing order, removing duplicates, and compacting them. The idea is the following: We redefine `\fopagecitation`, the command that handles the citations, it will construct a sorted list. When the end tag is seen, we shall compact and print the list.

```
\XMLelement{fotex:sort}
   {\XMLattributeX{range-char}{\FOrangechar}{\textendash}}
   {\let\fopagecitation\fosortpagecitation
    \global\sorttoks{}}
   {\global\sortcount-2\let\@elt\focompress@elt
    \let\fosep\@empty
    \let\foheld\relax
    \the\sorttoks
    \ifx\foheld\relax\else\FOrangechar\fi
    \foheld}
```

We are faced here with the following problem: insert a number in an ordered list (increasing order). Assume that the list contains X_1, X_2, X_3, etc., and the number to insert is Y. The idea is to consider each X_i, renaming it locally X. There are three cases to consider. If $X < Y$, it is too early, if $X = Y$ we do nothing, if $X > Y$ we insert Y here. Note that, if $X_i > Y$, then $X_{i+1} > Y$. As a consequence, we shall replace Y by ∞ after insertion. At the end of the loop, we can consult the value of Y: if it is not ∞, we have to insert it at the end.

In practice, we proceed as follows. First, each X_i is of the form `\@elt{3}` or `\@elt{12}`. These quantities are in the token list `\sorttoks`. We shall see in a minute how to evaluate this list in such a way that, at the start of the evaluation, the list is empty. Moreover we change the meaning of `\@elt` to `\fosort@elt`. The quantity Y is in the counter `\sortcount`. In the case $X < Y$, we have to re-insert `\@elt{X}` in the list (line 2408). In the case $X > Y$, we insert `\@elt{Y}` and `\@elt{X}` in the list. The non trivial point is to evaluate Y (there is no problem for X, since this is in `#1`), this requires four `\expandafter` commands on lines 2403 and 2404. Remember: since `\sorttoks` is a reference to a token list, if you say `\global\sorttoks\foo`, then `\foo` is expanded, but the tokens in the list are not. Here we have `\expandafter` instead of `\foo`, so that the `\the` after the brace is expanded. Since `\the` fully expands what follows, its effect is first to expand the `\expandafter` (and as a consequence the next `\the`) and second, to replace `\sorttoks` by its value (this is a bit unusual).

```
\newcount\sortcount
\newtoks\sorttoks
\def\fosort@elt#1{%
 \ifnum#1>\sortcount
   \global\sorttoks\expandafter{\the\expandafter\sorttoks\expandafter\@elt
       \expandafter{\the\sortcount}\@elt{#1}}%
  \global\sortcount\maxdimen
  \else
```

```
    \ifnum#1<\sortcount
    \global\sorttoks\expandafter{\the\sorttoks\@elt{#1}}%
    \fi
  \fi}
```

The `\fosortpagecitation` is used to add a page reference into a sorted list. Assume that `\label{foo}` has been executed somewhere. This defines `\r@foo`, to be a command that can be used by `\ref` or `\pageref`. In what follows, we want the number used by `\pageref`. For the case where `\pageref` is used before `\label`, LaTeX provides a mechanism in which information is printed in the auxiliary file, and read at the start of the job (and also at the end; sometimes you will see "labels may have changed"). If you compile a file for the first time, the `\r@foo` command is undefined, and `\pageref` has to consider this case; this is the reason why we cannot use this command directly. We shall assume here that the hyperref package has been loaded, so that, when defined, `\r@foo` contains a list of five items, the second one being the page number. On line 2413 we call the command that returns this second argument. On the line that follows, we insert five `\relax` tokens, in order to make sure that there are at least five tokens. On the preceding line, there are three `\expandafter`: if we had only one, the effect would be to replace `\csname` by `\r@foo`; since we have three, the effect is to replace it by the value of `\r@foo`. There is a little trick. If the command is undefined, `\csname` sets `\r@foo` to `\relax`, and `\@secondoffive` produces `\relax`. In some cases, the command might produce some junk. We can avoid the "missing number" error by inserting a zero in front of the token list. We get rid of the junk by evaluating everything else in a `\hbox`, that is copied in `\box0`, and discarded. Thus, this converts our page number in an integer, stored in `\sortcount`.

Our second job is then to sort. What we do is: we define `\sortoks` to be the empty token list, with an `\expandafter` that puts the content of the token list in the input stream. This means that we evaluate this token list in a context where the list is empty. Evaluating the list may have as side effect to insert this new element and replace the value by `\maxdimen`. Otherwise, we must insert it at the end.

```
\def\fosortpagecitation{%
    \setbox0\hbox{\global\sortcount=0\expandafter\expandafter\expandafter
    \@secondoffive\csname r@\FOrefid\endcsname
       \relax\relax\relax\relax\relax}%
     \let\@elt\fosort@elt
     \global\sorttoks\expandafter{\expandafter}\the\sorttoks
     \ifnum\sortcount<\maxdimen
    \global\sorttoks\expandafter{%
        \the\expandafter\sorttoks\expandafter\@elt\expandafter{\the\sortcount}}%
  \fi}
```

Let's compress the list. Consider the case where we have a list of numbers, say 1, 2, 3, 6, 8, and 9. We apply the next command to every element, in order. We assume that our numbers are positive, the counter is initialized to -2, so that the initial test is false. This means that the first number, 1, is typeset and remembered. The original code was wrong. We give here a correct version.

```
\gdef\focompress@elt#1{%
 \global\advance\sortcount\@ne
 \ifnum#1=\sortcount\relax
    \edef\foheld{\#1}%
  \else
    \ifx\foheld\relax\else\FOrangechar\fi
    \foheld\fosep#1\relax
    \let\foheld\relax
 \fi
```

```
\global\sortcount#1\relax
\def\fosep{, }}
```

This converts a XSLT format into a LaTeX command. There are examples in chapter 7 of this feature. It can be used to convert a section reference (defined by some numbers) into something like IV.7-2. If you say format = '[1.a]', and if the numbers are 4 and 2, the result is '`[4.b]`'. If we have only one number, the result is '`[4]`'. This mechanism is very complicated. We need only to typeset pages numbers (everything else is done by the XSLT processor), so that only one number has to be converted.

```
\expandafter\let\csname Format-1\endcsname\@arabic
\expandafter\let\csname Format-i\endcsname\@roman
\expandafter\let\csname Format-I\endcsname\@Roman
\expandafter\let\csname Format-a\endcsname\@alph
\expandafter\let\csname Format-A\endcsname\@Alph
```

Handling of `<fo:page-number/>`. We call a command that formats the page number.

```
\XMLelement{fo:page-number}
  {}
  {}
  {\expandafter\FOgeneratePage\FOformat\@null}
```

The call to the `\FOgeneratePage` command is a bit strange. The arguments `#1#2` is the result of the expansion of the format. Hence, `#1` is the first character of the format. This piece of code works if the format has the form '`1.`': it applies `\@arabic` to the page number and puts a dot after it. For later use, we put the interesting part in a command.

```
\def\FOgeneratePage#1#2\@null{\csname Format-#1\endcsname{\c@page}#2}
\def\jgFOlabel#1{\csname Format-#1\endcsname{\c@page}}
```

We define now `\FOlabel`. The idea of the command is to create a label associated to the current id, stored in `\FOid`, provided that this is not empty (the command may be called more than once for the same id, but only one label will be created). We do not execute the `\label` command, but instead, we write directly something in the .aux file, and create an anchor. As explained above, what we print is the command `\newlabel`, followed by the id, followed by a list of five items, all empty but the second one, containing the formatted page number. Note: the sort-page-citation shown above assumes that integer values are used,

The implementation was modified. Originally, the quantity printed on the .aux file was the same as the body of `<fo:page-number>`. The trouble is that all quantities in the `\write` command are expanded; thus `\@arabic` is expanded; its value is `\number`, so that the current page number (from `\c@page`) is used. This is wrong. The `\protected@write` command redefines temporarily `\thepage` to be `\relax`, so that, in the case of `\label`, the page number is not evaluated (it will be later, when the page is shipped out). We corrected this in the following way. We add a local redefinition of `\jgFOlabel` to `\relax`. As a consequence, the `\write` will store the command and the value of the format attribute, the command will be expanded later. There is no need to this `\@null` hacking.

```
\def\FOlabel{%
 \ifx\@empty\FOid\else
  \@bsphack
  \protected@write\@mainaux{\let\jgFOlabel\relax}%
    {\string\newlabel{\FOid}{{}{\jgFOlabel\FOformat}{}{}{}}}%
  \@esphack
  \hyper@@anchor{\FOid}{\relax}%
  \global\let\FOid\@empty
 \fi
}
```

4.23 Other elements

The fotex.sty file defines a command `\@declaredcolor` that is unused. It defines this one:

```
\def\HTMLColor#1#2#3#4#5#6#7#8{%
 \definecolor{#1}{RGB}{"#3#4, "#5#6, "#7#8}}
```

This predefines some colors.

```
\HTMLColor{aqua}.00FFFF
\HTMLColor{black}.000000
\HTMLColor{blue}.0000FF
\HTMLColor{fuchsia}.FF00FF
\HTMLColor{gray}.808080
\HTMLColor{green}.008000
\HTMLColor{lime}.00FF00
\HTMLColor{maroon}.800000
\HTMLColor{navy}.000080
\HTMLColor{olive}.808000
\HTMLColor{purple}.800080
\HTMLColor{red}.FF0000
\HTMLColor{silver}.C0C0C0
\HTMLColor{teal}.008080
\HTMLColor{white}.FFFFFF
\HTMLColor{yellow}.FFFF00
\definecolor{orange}{cmyk}{0,0.61,0.87,0}
```

Leaders: This might produce − − − − − − − − − ..... _____ in the case where the style is dashed, or dotted, or if the thickness is 1pt. If the width is zero, we emit the leader command, otherwise put it in a hbox of the desired width.

```
\XMLelement{fo:leader}
  {}
  {
 \leavevmode
  \ifx\FOleaderpattern\leader@pattern@rule
   \ifx\FOrulestyle\rule@style@dashed
    \def\w@t{\cleaders\hbox{$\m@th \mkern1.5mu-\mkern1.5mu$}\hfil}%
   \else
    \ifx\FOrulestyle\rule@style@dotted
      \def\w@t{\cleaders\hbox{$\m@th \mkern1.5mu.\mkern1.5mu$}\hfil}%
    \else
      \ifdim\FOrulethickness>\z@
        \def\w@t{\leaders\hrule height \FOrulethickness\hfill}%
      \else
        \def\w@t{\hfill}%
      \fi
    \fi
   \fi
  \else
  \ifx\FOleaderpattern\leader@pattern@dots
      \def\w@t{\cleaders\hbox{$\m@th \mkern1.5mu.\mkern1.5mu$}\hfill}%
  \else % space
      \def\w@t{\hfill}%
  \fi
 \fi
```

```
 \jg@leaderaux
   }
  {}
```

Changes in V2. This computes the leader width correctly, using minimum, optimum and maximum values. The macro `\w@t` contains the leader box.

```
\def\jg@leaderaux{
 \ifx\@empty\FOleaderlength
   \PercentToDimen{\FOleaderlengthoptimum}%
   \edef\leaderopt{\the\@tempdima}%
   \PercentToDimen{\FOleaderlengthminimum}%
   \edef\leadermin{\the\@tempdima}%
   \PercentToDimen{\FOleaderlengthmaximum}%
   \edef\leadermax{\the\@tempdima}%
   \@tempdima\leaderopt\relax
   \advance\@tempdima-\leadermin\relax
   \@tempdimb\leadermax\relax
   \advance\@tempdimb-\leaderopt\relax
   \LeaderLength\leaderopt plus \@tempdimb minus \@tempdima\relax
 \else
   \PercentToDimen{\FOleaderlength}%
   \LeaderLength\@tempdima\relax
 \fi
 \ifdim\LeaderLength=\z@
   \w@t
 \else
   \hbox to \LeaderLength{\w@t}%
\fi}
```

Attributes and commands for handling leaders.

```
\XMLNSAX{fo}{leader-alignment}{\FOleaderalignment}{\inherit}
\XMLNSAX{fo}{leader-length}{\FOleaderlength}{\inherit}
\XMLNSAX{fo}{leader-pattern}{\FOleaderpattern}{\inherit}
\XMLNSAX{fo}{leader-pattern-width}{\FOleaderpatternwidth}{\inherit}
\XMLNSAX{fo}{rule-style}{\FOrulestyle}{\inherit}
\XMLNSAX{fo}{rule-thickness}{\FOrulethickness}{\inherit}
\XMLNSAX{fo}{leader-length.maximum}{\FOleaderlengthmaximum}{\textwidth}
\XMLNSAX{fo}{leader-alignment}{\FOleaderalignment}{\inherit}
\XMLNSAX{fo}{leader-length}{\FOleaderlength}{\inherit}
\XMLNSAX{fo}{leader-pattern}{\FOleaderpattern}{\inherit}
\XMLNSAX{fo}{leader-pattern-width}{\FOleaderpatternwidth}{\inherit}
\XMLNSA{fo}{leader-length.optimum}{\FOleaderlengthoptimum}{0pt} % no X?
\XMLNSAX{fo}{leader-length.minimum}{\FOleaderlengthminimum}{\z@}
\gdef\FOleaderalignment{none}
\gdef\FOleaderlength{}
\gdef\FOleaderpattern{space}
\gdef\FOleaderpatternwidth{}
\gdef\FOrulestyle{solid}
\gdef\FOrulethickness{1.0pt}
\XMLstringX\rule@style@dashed<>dashed</>
\XMLstringX\rule@style@dotted<>dotted</>
\XMLstringX\leader@p@attern@space<>space</>
\XMLstringX\leader@pattern@rule<>rule</>
```

```
\XMLstringX\leader@pattern@dots<>dots</>
\newskip\LeaderLength
```

These are trivial.

```
\XMLelement{fo:block-container}
  {}  {}   {}
\XMLelement{fo:inline-container}
  {}  {}   {}
\XMLelement{fo:wrapper}
  {}
  {\FOSetFont{wrapper}\FOlabel}
  {}
```

These are undefined.

```
\XMLelement{fo:bidi-override}
\XMLelement{fo:initial-property-set}
\XMLelement{fo:instream-foreign-object}
\XMLelement{fo:multi-case}
\XMLelement{fo:multi-properties}
\XMLelement{fo:multi-property-set}
\XMLelement{fo:multi-switch}
\XMLelement{fo:multi-toggle}
\XMLelement{fo:table-footer}
```

Handling of `<fotex:bookmark FB-level=`*A* `FB-label=`*B*`>`*text*`</fotex:bookmark>`. We have written FB instead of 'fotex-bookmark', it's shorter. The translation is `\pdfbookmark[A]{text}{B}`.

```
\XMLelement{fotex:bookmark}
 {
  \XMLattributeX{fotex-bookmark-level}{\FOTEXbookmarklevel}{0}
  \XMLattributeX{fotex-bookmark-label}{\FOTEXbookmarklabel}{}
 }
 {\xmlgrab}
 {\protectCS\FOTEXbookmarklabel
   \let\ignorespaces\@empty
   \pdfbookmark[\FOTEXbookmarklevel]{#1}{\FOTEXbookmarklabel}}
```

What is the purpose of this?

```
\let\@@ReadBookmarks\ReadBookmarks
\def\ReadBookmarks{{\let\InputIfFileExists\@input\@@ReadBookmarks}}
```

Implementation of `<fo:character character=`*C*`/>`. This is a bit strange. We use more or less the same method as `<fo:inline>`.

```
\XMLNSA{fo}{character}{\FOcharacter}{}
\XMLelement{fo:character}
  {}
  {
  \ifx\FOverticalalign\att@auto \let\FOverticalalign\FObaselineshift \fi
  \FO@character{\FOcharacter}}
  {}
```

What we do is look at the vertical alignment attribute, and use some more or less standard functions for vertical placement.

```
\def\FO@character#1{%
 \ifx\FOverticalalign\att@baseline #1%
```

```
 \else \ifx\FOverticalalign\att@super \textsuperscript{#1}%
 \else \ifx\FOverticalalign\att@sub  \textsubscript{#1}%
 \else \PlayWithShift \raisebox{\dimen@}{#1}%
 \fi \fi \fi}
```

The `\textsuperscript` command is standard, `\textsubscript` is not. We use the same idea. Call to `\fontsize` removed in both commands.

```
\DeclareRobustCommand*\textsubscript[1]{%
  \@textsubscript{\selectfont#1}}
\def\@textsubscript#1{%
  {\m@th\ensuremath{_{\mbox{#1}}}}}
\def\@textsuperscript#1{%
  {\m@th\ensuremath{^{\mbox{#1}}}}}
```

This piece of code redefines `\obeyspaces`, so as to make space an active character, with value `\FOdiscretionary`. This fonction considers the value of `\FOwrapoption`. If it is 'nowrap', then space is a discretionary character, if a line is split at this position, an arrow with a hook pointing to the left is printed. Otherwise, a normal space is used.

```
\gdef\FOdiscretionary{%
 \ifx\FOwrapoption\att@nowrap
     \discretionary{\kern-.5ex\lower1ex\hbox{$\hookleftarrow$}}{}{\kern1ex}%
 \else\space\fi}
\def\obeyspaces{\catcode`\ =\active}
{\obeyspaces\global\let =\FOdiscretionary}
```

4.24 Bootstrap code

Some integers, boxes and dimensions.

```
\newcount\FOTableNesting  \FOTableNesting0  % unused...
\newcount\FOinList        \FOinList0
\newdimen\FOspaceleft
\newdimen\MasterBottomMargin     \MasterBottomMargin\z@
\newdimen\MasterLeftMargin       \MasterLeftMargin\z@
\newdimen\MasterRightMargin      \MasterRightMargin\z@
\newdimen\MasterTopMargin        \MasterTopMargin\z@
\newdimen\XFOendindent  \newdimen\XFOstartindent % unused
\newdimen\bottommargin
\newdimen\FOtempdim
\newdimen\@default \@default=10pt
\newsavebox\BlockBox
\newsavebox\CellBox
\newsavebox\FOBOX
```

Some conditionals.

```
\newif\ifFOBlockGrab   \FOBlockGrabfalse
\newif\ifFODebug       \FODebugfalse
\newif\ifFOListBody    \FOListBodyfalse
\newif\ifFOListInnerPar\FOListInnerParfalse
\newif\ifFOinOutput    \FOinOutputfalse
```

```
\def\DEBUG#1{%
 \ifFODebug \typeout{#1, at \the\inputlineno} \fi
}
```

These are defined, but currently ignored.

```
\def\usewhitespace{%
 \UnicodeCharacter{13}{ \ignorespaces}%
 \UnicodeCharacter{32}{ \ignorespaces}%
 \UnicodeCharacter{9}{ \ignorespaces}%
}
\def\ignorewhitespace{%
 \UnicodeCharacter{13}{}%
 \UnicodeCharacter{32}{}%
 \UnicodeCharacter{9}{}%
}
```

Some strings, used everywhere.

```
\XMLstring\LINK<>LINK</>
\XMLstringX\att@all<>all</>
\XMLstringX\att@any<>any</>
\XMLstringX\att@auto<>auto</>
\XMLstringX\att@blank<>blank</>
\XMLstringX\att@black<>black</>
\XMLstringX\att@bottom<>bottom</>
\XMLstringX\att@centered<>center</>
\XMLstringX\att@false<>false</>
\XMLstringX\att@first<>first</>
\XMLstringX\att@no<>no</>
\XMLstringX\att@none<>none</>
\XMLstringX\att@normal<>normal</>
\XMLstringX\att@page<>page</>
\XMLstringX\att@pre<>pre</>
\XMLstringX\att@repeat<>repeat</>
\XMLstringX\att@true<>true</>
```

Some attributes.

```
\XMLNSA{fo}{color}{\FOcolor}{\inherit}    \gdef\FOcolor{black}
\XMLNSAX{fo}{id}{\FOid}{}
\XMLNSAX{fo}{role}{\FOrole}{none}
\XMLNSAX{fo}{size}{\FOsize}{auto}
\XMLNSAX{fo}{src}{\FOsrc}{}
\XMLNSAX{fo}{text-indent}{\FOtextindent}{\inherit} \gdef\FOtextindent{\z@}
\XMLNSAX{fo}{top}{\FOtop}{auto}
\XMLNSA{fo}{vertical-align}{\FOverticalalign}{auto}
\XMLNSAX{fo}{width}{\FOwidth}{auto}
\XMLNSAX{fotex}{column-align}{\FOcolumnalign}{}
\XMLNSAX{fo}{format}{\FOformat}{\inherit}    \gdef\FOformat{1}
```

Some names.

```
\XMLname{fo:list-item-label}{\FOListItemLabel}
\XMLname{fo:list-item-body}{\FOListItemBody}
\XMLname{fo:table-cell}{\FOTableCell}
\XMLname{fo:table-row}{\FOTableRow}
```

This is executed when we load the file. Comments of the form D=foo indicate the default value in LaTeX. Changes in V2: `\widowpenalty` and `\clubpenalty` changed from 8000 to 0. We think that a value like 0pt plus 2pt is better for the `\parskip` glue.

```
\def\fps@table{!htbp} %D=tbp
\def\fps@figure{!htbp} %D=tbp
\parindent\z@      %D=20pt
\parskip\z@        %D=0pt plus 1pt
\emergencystretch 3em  %D=0.0pt
\tabcolsep3pt      %D=6pt
\hbadness=4000     %D=1000
\hyphenpenalty=400 %D=50
\pretolerance=500  %D=100
\relpenalty=500
\tolerance=1000    %D=200
\vbadness=3000     %D=1000
\widowpenalty=0 %D=150
\clubpenalty=0  %D=150
\@twosidetrue
\fboxsep0pt        %D=3pt
\setcounter{topnumber}{5}        %D=2
\renewcommand\topfraction{.9}    %D=.7
\setcounter{bottomnumber}{12}    %D=1
\renewcommand\bottomfraction{.9} %D=.3
\setcounter{totalnumber}{6}      %D=3
\renewcommand\textfraction{.1}   %D=.2

\DefineCharacter{8232}{2028}{\newline}
\DefineCharacter{8208}{2010}{-\/}
\@ifundefined{pdfoutput}{}{\def\pdfBorderAttrs{/Border [0 0 0]}}
\long\def\@firstoffive#1#2#3#4#5{#1}%
\long\def\@secondoffive#1#2#3#4#5{#2}%
\long\def\@thirdoffive#1#2#3#4#5{#3}%
\long\def\@fourthoffive#1#2#3#4#5{#4}%
\long\def\@fifthoffive#1#2#3#4#5{#5}%

\def\supppdf{supp-pdf}
\let\FOinputIfFileExists\InputIfFileExists
\def\InputIfFileExists#1#2#3{%
 {\def\@tempa{#1}\ifx\@tempa\supppdf\else
   \FOinputIfFileExists{#1}{#2}{#3}\fi}}
\providecommand\textasciitilde{~}
```

Some booleans.

```
\newif\ifFODefiningPage  \FODefiningPagefalse
\newif\ifFOinLayout      \FOinLayoutfalse
\newif\ifMulticolPending \MulticolPendingfalse
\newif\ifStartWithOmit   \StartWithOmitfalse
\newif\ifForcePageSetup
\newif\ifBlankPage
\newif\ifInInsertion
```

Chapter 5

Converting XML to XML

This chapter, and the next one, describes three types of style sheets: they convert XML to XML, to XSL/Format and HTML. Originally, in 2002 and 2003, the XML files created by Tralics were used directly for production of the HTML and Pdf version, but in 2004, a new DTD was designed for the Raweb. The name of this new DTD was unclear for a long time; it is now 'raweb2.dtd' and the old name is 'raweb3.dtd' (the 3 here is for 2003). We shall explain here the style sheets that convert from the old DTD to the new one, and from this to HTML; we shall also explain the style sheets that convert from the old DTD to XSL/Format (those for the new one are similar).

The style sheets for converting into XSL/Format are adaptations by José Grimm of the TEI code (by Sebastian Rahtz). These are part of the Tralics distribution. Other files were written by J. Grimm (conversion to HTML) or Tahia Benhaj Abdellatif (conversion to XML) and maintained by Marie-Pierre Durollet and Bruno Marmol. The Tralics files have a Copyright notice that looks like this:

```
<!-- Copyright Inria 2003-2004 Jose Grimm. This file is an adaptation of
files from the TEI distribution. See original Copyright notice below.
-->
```

The "original Copyright notice" is given here:

```
<!--
 Copyright 1999-2001 Sebastian Rahtz/Oxford University
    <sebastian.rahtz@oucs.ox.ac.uk>

 Permission is hereby granted, free of charge, to any person obtaining
 a copy of this software and any associated documentation files (the
 ''Software''), to deal in the Software without restriction, including
 without limitation the rights to use, copy, modify, merge, publish,
 distribute, sublicense, and/or sell copies of the Software, and to
 permit persons to whom the Software is furnished to do so, subject to
 the following conditions:

 The above copyright notice and this permission notice shall be included
 in all copies or substantial portions of the Software.
-->
```

Let's consider an example. This is the start of a document created by Tralics:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<!DOCTYPE raweb SYSTEM 'raweb3.dtd'>
<!-- translated from latex by tralics 2.4-->
```

```
<raweb language='english' creator='Tralics version2.4' year='2004'>
<accueil isproject='false' html='apics'>
```

Two years later, the team has become a project, and the header is:

```
<!-- translated from latex by tralics 2.8.1-->
<raweb language='english' creator='Tralics version 2.8.1' year='2006'>
<accueil isproject='true' html='apics'>
```

This is the start of the translation to the new DTD:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--translated from old xml 2003 by with 2XMLvalideDTD2.xsl-->
<!DOCTYPE raweb PUBLIC "-//INRIA//DTD Raweb 2" "raweb2.dtd">
<raweb xmlns:html="http://www.w3.org/1999/xhtml" xml:lang="en" year="2006">
  <identification isproject="true" id="apics">
    <shortname>Apics</shortname>
```

There is a second style sheet that adds ids to all elements. The resulting file starts like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE raweb SYSTEM "raweb.dtd">
<raweb xmlns:html="http://www.w3.org/1999/xhtml"
       xmlns:xlink="http://www.w3.org/1999/xlink" id="id2243496"
       xml:lang="en" year="2004">
  <identification id="apics" isproject="false">
    <shortname id="id2267539">apics</shortname>
```

In 2006, 'SYSTEM' was replaced by 'PUBLIC', and ids are added only when needed.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE raweb PUBLIC "-//INRIA//DTD Raweb 2" "raweb2.dtd">
<raweb xmlns:html="http://www.w3.org/1999/xhtml"
       xmlns:xlink="http://www.w3.org/1999/xlink"
       xml:lang="en" year="2006">
  <identification id="apics" isproject="true">
    <shortname>Apics</shortname>
```

Note the following details: Tralics uses simple quotes when it outputs its tree; the XSLT processor used by the Raweb uses double quotes; in the style files random values are used. In this exemple the output of the XSLT processor is 'indented', this is an option that depends on the style sheet. Tralics never indents.

5.1 Converting the XML to the new DTD

All files start like this, we shall not repeat this line.

```
1 <?xml version="1.0" encoding="iso-8859-1" ?>
```

The root element here is `<xsl:transform>`; in some other files, it can be `<xsl:stylesheet>`, this is the same. We declare the namespaces. The 'html' namespace is not used here.

```
2 <xsl:transform
3         xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
4         xmlns:xlink="http://www.w3.org/1999/xlink"
5         xmlns:html="http://www.w3.org/1999/xhtml"
6         exclude-result-prefixes="xlink">
```

We use an auxiliary file for the bibliography; this will be defined in the next section.

```
7 <xsl:import href="2XMLvalideDTD2-biblio.xsl"/>
```

The style sheet contains this line, but it is useless since we will use `<xsl:document>`.

```
<xsl:output method='xml' doctype-system='raweb2.dtd' indent='yes'
    encoding='iso-8859-1'/>
```

Spaces are removed in some elements, listed here. The list contains a single name, `<UR>`.

```
<xsl:strip-space elements="UR"/>
```

The root of the Raweb is the `<raweb>` element. Its children are `<accueil>` (that gives some information about the team; this is used to produce a title page[1]), `<moreinfo>` (this is optional, it is a short piece of text, annexed to a section, or the whole document), `<composition>` (this describes the members of the team), `<presentation>`, `<fondements>`[2], `<domaine>`, `<logiciels>`[3], `<resultats>`, `<contrats>`, `<international>`, `<diffusion>` (these eight elements are called 'sections', they have the same structure and contain text, that can be divided into subsections, etc.), and `<biblio>` (this is the bibliography).

The table of contents of the Raweb has ten entries: Members, Overall Objectives, Scientific Foundations, Application Domains, Software, New Results, Contracts and Grants with Industry, Other Grants and Activities, Dissemination, Bibliography. Each entry comes from one of these elements, except `<accueil>` and `<moreinfo>`, that play a special role. In particular the html attribute of `<accueil>` is the Team's name in lower case ASCII 7bits. This is also the name of the source file, and a prefix for output files. We store in `$LeProjet` this name[4].

```
<xsl:variable name="LeProjet" select="/raweb/accueil/@html"/>
```

The `<raweb>` element has two important attributes, language and year. We put in the variable `$year` this quantity with a default value of 2004.

```
<xsl:variable name="year">
 <xsl:choose>
  <xsl:when test='/raweb/@year'><xsl:value-of select="/raweb/@year"/></xsl:when>
  <xsl:otherwise>2004</xsl:otherwise>
 </xsl:choose>
</xsl:variable>
```

This is the main rule. The translation of `<raweb>` is the file apics-dtd2.xml (assuming that `$LeProjet` is 'apics'), its root element is `<raweb>`. This element has some attributes, namely namespaces, year (from the `$year` variable) and xml:lang from the language attribute. The content is formed by the transformation of the eight standard sections, followed by the bibliography, preceded by the transformation of `<accueil>` and `<composition>`.

```
<xsl:template match="/raweb">
 <xsl:document href="{$LeProjet}-dtd2.xml" method="xml"
    doctype-public="-//INRIA//DTD Raweb 2"  doctype-system="raweb2.dtd"
    indent='yes' encoding='iso-8859-1'>
 <xsl:comment>translated from old xml 2003 by with 2XMLvalideDTD2.xsl</xsl:comment>
 <raweb
       xmlns:xlink="http://www.w3.org/1999/xlink"
       xmlns:html="http://www.w3.org/1999/xhtml">
   <xsl:attribute name="lang" namespace="http://www.w3.org/XML/1998/namespace">
      <xsl:choose>
         <xsl:when test="@language='english'">en</xsl:when>
         <xsl:when test="@language='french'">fr</xsl:when>
         <xsl:otherwise><xsl:value-of select="@language"/></xsl:otherwise>
      </xsl:choose>
   </xsl:attribute>
   <xsl:attribute name="year"><xsl:value-of select="{$year}"/></xsl:attribute>
```

[1] In French, accueil means reception, welcome.
[2] Fondements = foundations
[3] Logiciel = software
[4] 'LeProjet' is French for 'TheTeam'.

```
34     <xsl:apply-templates select="accueil"/>
35     <xsl:call-template name="topic"/>
36     <xsl:apply-templates select="presentation |
37                          fondements | domaine | logiciels | resultats |
38                          contrats | international | diffusion"/>
39     <xsl:apply-templates select="biblio"/>
40    </raweb>
41   </xsl:document>
42  </xsl:template>
```

A topic is an attribute of a module. Modules can be ordered by section or by topic; for this reason there are two style sheets that convert the XML into HTML. The table of contents of the HTML version thas a button that switches form one view to the other; there is only one style sheet for the Pdf, it ignores these topic attributes. A topic attribute is a reference to a topic declaration like `<topic num='1'><t_titre> High-level modeling</t_titre></topic>`. The value of the num attribute is computed by Tralics; specifications say that it should be an integer. In the new DTD, it should be an ID, so that we transform it to t_1 (the IDs generated by Tralics are of the form uid125 or bid125, so that this transformation does not conflict with already existing IDs). The `<t_titre>` element is useless here; it was removed in the new DTD. The topic declarations are moved from inside `<accueil>` to after `<identification>`.

```
43  <xsl:template name="topic">
44   <xsl:for-each select="accueil/child::topic">
45    <xsl:element name="topic">
46     <xsl:attribute name="id">t_<xsl:value-of select="@num"/></xsl:attribute>
47       <xsl:value-of select="./t_titre"/>
48     </xsl:element>
49   </xsl:for-each>
50  /xsl:template>
```

This piece of code is a copy from the TEI. The idea is to leave math formulas unchanged. It will be used in all style sheets.

```
51  <xsl:template match="*|@*|comment()|processing-instruction()|text()" mode="math">
52   <xsl:copy>
53    <xsl:apply-templates mode="math" select="*|@*|processing-instruction()|text()"/>
54   </xsl:copy>
55  </xsl:template>
```

A `<formula>` is a wrapper for a `<math>` expression. The action here is to copy the attributes and the content.

```
56  <xsl:template match="formula">
57      <xsl:element name="formula">
58         <xsl:copy-of select="@*"/><xsl:apply-templates mode="math"/>
59      </xsl:element>
60  </xsl:template>
```

If a formula has the attribute type = 'display', it is a display math formula, outside any paragraph. We put it in a `<p>` element.

```
61  <xsl:template match="formula[@type='display']">
62   <p>
63    <xsl:element name="formula">
64       <xsl:copy-of select="@*"/><xsl:apply-templates mode="math"/>
65    </xsl:element>
66   </p>
67  </xsl:template>
```

We convert `<accueil>` into `<identification>`. We copy the isproject attribute (this is 'true' in the case where the team is a project, 'false' otherwise). We copy the html trait, renaming it id.

After that, we add the transformations of the following elements: <projet>, <projetdeveloppe>, <theme>, <composition> (this is a sibling), <UR>, and <moreinfo> (this is optional).

```
<xsl:template match="accueil">
 <xsl:element name="identification">
  <xsl:attribute name="isproject"><xsl:value-of select="@isproject"/></xsl:attribute>
  <xsl:attribute name="id"><xsl:value-of select="@html"/></xsl:attribute>
  <xsl:apply-templates select="projet"/>
  <xsl:apply-templates select="projetdeveloppe"/>
  <xsl:apply-templates select="theme"/>
  <xsl:apply-templates select="../composition"/>
  <xsl:apply-templates select="UR"/>
  <xsl:if test="/raweb/moreinfo">
    <xsl:apply-templates select="/raweb/moreinfo"/>
  </xsl:if>
 </xsl:element>
</xsl:template>
```

We copy the <theme> element, replacing lowercase letters by uppercase ones.

```
<xsl:template match="accueil/theme">
 <xsl:element name="theme">
   <xsl:value-of select="translate(.,'abcdefghijklmnopqrstuvwxyz',
                                    'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
 </xsl:element>
</xsl:template>
```

We copy the <projet> element, renaming it <shortname>. Note that, for the LaTeX team, this could be <LaTeX>, so that a simple copy is not enough.

```
<xsl:template match="accueil/projet">
 <xsl:element name="shortname"> <xsl:apply-templates />  </xsl:element>
</xsl:template>
```

We copy the <projetdeveloppe> element, renaming it <projectName>[5]. Note that this element can have font changes, hence we must process the content.

```
<xsl:template match="accueil/projetdeveloppe">
 <xsl:element name="projectName"> <xsl:apply-templates /> </xsl:element>
</xsl:template>
```

In the case of <UR>, we consider only the content; it should be a sequence of elements of the form <URxxx>, these elements are listed below.

```
<xsl:template match="UR">
   <xsl:apply-templates />
</xsl:template>
```

We replace <URRocquencourt>, <URRhoneAlpes>, <URRennes>, <URLorraine>, <URFuturs>, and <URSophia> by <UR name='Rocquencourt'/>, etc. These elements are empty, they represent one of the six INRIA's research units. We use here a litteral result element, the actual code uses <xsl:element> and <xsl:attribute>.

```
<xsl:template match="URRocquencourt">
   <UR name="Rocquencourt" />
</xsl:template>
<xsl:template match="URRhoneAlpes">
   <UR name="RhoneAlpes"/>
</xsl:template>
<xsl:template match="URRennes">
   <UR name="Rennes"/>
```

[5]Here the French word développé is used in the sense of expanded. All tag names are ASCII 7bits, although XML allows any Unicode character.

```
105 </xsl:template>
106 <xsl:template match="URLorraine">
107    <UR name="Lorraine"/>
108 </xsl:template>
109 <xsl:template match="URFuturs">
110    <UR name="Futurs" />
111 </xsl:template>
112 <xsl:template match="URSophia">
113    <UR name="Sophia" />
114 </xsl:template>
```

Translation of `<presentation>`, `<fondements>`, `<domaine>`, `<logiciels>`, `<resultats>`, `<contrats>`, `<international>`, and `<diffusion>`. The result is an element of the same name. The only difference is that we convert the titre attribute in a `<bodyTitle>` element. This attribute is constant, defined in the DTD, see page 337 and following, lines 181, 186, 191, 196, 201, 206, 211, 216, and 221.

```
115 <xsl:template match="presentation | fondements | domaine | logiciels |
116     resultats | contrats | international | diffusion">
117  <xsl:variable name="nodename" select="name()"/>
118  <xsl:element name="{$nodename}">
119    <xsl:attribute name="id">  <xsl:value-of select="@id"/>  </xsl:attribute>
120    <xsl:element name="bodyTitle">
121        <xsl:value-of select="@titre"/>
122    </xsl:element>
123    <xsl:apply-templates/>
124  </xsl:element>
125 </xsl:template>
```

We simplified a bit the code above by assuming that the id attribute is present. The following code should be used instead of line 119.

```
<xsl:choose>
   <xsl:when test="@id">
      <xsl:attribute name="id">
         <xsl:value-of select="@id"/>
      </xsl:attribute>
   </xsl:when>
   <xsl:when test="@num">
      <xsl:attribute name="id">
         <xsl:value-of select="@num"/>
      </xsl:attribute>
   </xsl:when>
   <xsl:otherwise>
      <xsl:attribute name="id">
         <xsl:value-of select="position()"/>
      </xsl:attribute>
   </xsl:otherwise>
 </xsl:choose>
```

Translation of `<module>`. The result is a `<subsection>`. We convert, in order, `<head>` (this is the title of the module), `<participant>`, `<participante>`, `<participants>`, `<participantes>` (four variants that indicate the participants to the action described in the module), `<keywords>` (the keywords), `<moreinfo>` (the 'moreinfo' data structure; we grab the first element, its transformation will read the other siblings), and finally everything else. A module has two attributes id and topic that are copied. If the topic is present, we have to convert the value as above line 46 ('12' replaced by 't_12').

```
126 <xsl:template match="module">
127  <xsl:element name="subsection">
128    <xsl:if test="@topic and @topic!=''">
129      <xsl:attribute name="topic">t_<xsl:value-of select="@topic"/>
130      </xsl:attribute>
131    </xsl:if>
132    <xsl:call-template name="id"/>
133    <xsl:apply-templates select="head" mode="caption"/>
134    <xsl:apply-templates
135       select="participants | participant | participantes | participante"/>
136    <xsl:apply-templates select="keywords"/>
137    <xsl:apply-templates select="moreinfo[position()=1]"/>
138    <xsl:apply-templates select="node()[local-name() != 'moreinfo'
139        and local-name()!='keywords' and local-name()!='head'
140        and local-name()!='participants' and local-name()!='participant'
141        and local-name()!='participante' and local-name()!='participantes' ]"/>
142  </xsl:element>
143 </xsl:template>
```

Translation of `<div0>`, `<div1>`, `<div2>`, `<div3>`, and `<div4>`. The result is a `<subsection>`, the code is the same as for a module, except that these elements have no topic attribute.

```
144 <xsl:template match="div0 | div1 | div2 | div3 | div4">
145  <xsl:element name="subsection">
146    <xsl:call-template name="id"/>
147    <xsl:apply-templates select="head" mode="caption"/>
148    <xsl:apply-templates
149         select="participants | participant | participantes | participante"/>
150    <xsl:apply-templates select="keywords"/>
151    <xsl:apply-templates select="moreinfo[position()=1]"/>
152    <xsl:apply-templates select="node()[...]" />         <!-- as above l. 139-142-->
153  </xsl:element>
154 </xsl:template>
```

Transformation of `<moreinfo>`. The result is a `<moreinfo>` that contains the content of the element and all the following siblings.

```
155 <xsl:template match="moreinfo">
156  <xsl:element name="moreinfo">
157    <xsl:apply-templates/>
158    <xsl:for-each select="following-sibling::moreinfo">
159      <xsl:apply-templates/>
160    </xsl:for-each>
161  </xsl:element>
162 </xsl:template>
```

Transformation of `<composition>`. The result is a `<team>` element, containing the `<catperso>` children and an optional `<moreinfo>` (let's hope there is only one, because of the code line 158).

```
163 <xsl:template match="composition">
164  <xsl:element name="team">
165    <xsl:call-template name="id"/>
166    <xsl:call-template name="catperso"/>
167    <xsl:apply-templates select="moreinfo"/>
168  </xsl:element>
169 </xsl:template>
```

The transformation of `<catperso><head>foo</head>etc</catperso>` is a `<participants>` element with an attribute category = 'foo', with spaces replaced by underscores, and whose content is the translation of all `<pers>` elements it contains (the semantics is: a `<catperso>` contains a

title in <head>, that could be 'Ph.D. Students', followed by some <pers> elements, all the students of the team).

```
<xsl:template name="catperso">
 <xsl:for-each select="catperso">
   <xsl:element name="participants">
     <xsl:attribute name="category">
       <xsl:value-of select="translate(./head, ' ', '_')"/>
     </xsl:attribute>
     <xsl:apply-templates select="pers"/>
   </xsl:element>
 </xsl:for-each>
</xsl:template>
```

The transformation of <participants> is also a <participants> element, where the category attribute has value 'None'. Originally, we had four elements, this one and <participant>, <participante>, <participantes>. This was simplified: the difference between masculine and feminine does not appear in English; the final s is removed, it will be added later if the list contains more than one element.

```
<xsl:template match="participants | participant | participantes | participante">
  <xsl:element name="participants">
    <xsl:attribute name="category">None</xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

The transformation of <pers prenom='Donald' nom='Knuth'>Author of <TeX/> </pers> is a <person> element, with three children, the first is <firstname>, the second is <lastname>, they contain the prenom and nom, and the last one is a <moreinfo> element that contains the content of this element; it is optional. The test is strange because later on, see lines 1133 and 1147 in the next chapter, we test again for emptyness, but white space is normalised there, not here. The code has changed in 2006, because two required attributes affiliation and profession were added. Moreover, the LaTeX command has two optional arguments, producing the value of the hdr attribute and the content of the <pers> element. If only one optional arguments is given, it is the value of the element; this piece of code allows the case where one optional argument is given, with value 'habilite',[6], which is handled as if there were two optional arguments, empty content, non-empty attribute.

```
<xsl:template match="pers">
 <xsl:element name="person">
   <xsl:call-template name="id"/>
   <xsl:element name="firstname"><xsl:value-of select="./@prenom"/></xsl:element>
   <xsl:element name="lastname"><xsl:value-of select="./@nom"/> </xsl:element>
   <xsl:element name="affiliation">
       <xsl:value-of select="./@affiliation"/>
   </xsl:element>
   <xsl:element name="categoryPro">
       <xsl:value-of select="./@profession"/>
   </xsl:element>
   <xsl:if test="string-length(.) > 0 and .!='habilite'">
     <xsl:element name="moreinfo">
        <xsl:apply-templates/>
     </xsl:element>
   </xsl:if>
   <xsl:if test="string-length(./@hdr) > 0 or .='habilite'">
```

[6] The French word 'habilité', with an acute accent, means 'entitled', hence 'habilité à diriger des recherches', in short HDR, means entitled to be a Phd thesis supervisor

```
      <xsl:element name="hdr">
        <xsl:text>oui</xsl:text>
      </xsl:element>
    </xsl:if>
 </xsl:element>
</xsl:template>
```

The element `<refperson>` is not defined in the old DTD. We can leave it unchanged.

```
<xsl:template match="refperson"> <xsl:copy-of select="."/> </xsl:template>
```

Transformation of `<hi rend=XX>text</hi>`. The result depends on the value of the attribute. If the attribute is 'sup', we construct a `<sup>` element; if the attribute is 'sub', we construct a `<sub>` element; if the attribute is 'bold', we construct a `<b>` element, with a hack: if you use the obsolete environments body and abstract, Tralics inserts a warning in the document, this is removed here[7]; if the attribute is 'small', we construct a `<small>` element; if the attribute is 'large', we construct a `<big>` element; if the attribute is 'tt', we construct a `<tt>` element; if the attribute is 'sc', we construct a `<span>` element[8]; if the attribute is 'center'[9], we construct a `<span>` element; if the attribute is 'underline' we construct a `<em>` element; otherwise, the result is a `<i>` element.

```
<xsl:template match="hi">
  <xsl:choose>
   <xsl:when test="@rend = 'sup'"> <sup><xsl:apply-templates/></sup></xsl:when>
   <xsl:when test="@rend = 'sub'"> <sub><xsl:apply-templates/></sub></xsl:when>
   <xsl:when test="@rend = 'bold'">
    <xsl:if test=".!='Body (obsolete)'">
       <xsl:if test=".!='Abstrat (obsolete)'">
         <b><xsl:apply-templates/></b>
       </xsl:if>
    </xsl:if>
   </xsl:when>
   <xsl:when test="@rend = 'small'">
     <small><xsl:apply-templates/></small> </xsl:when>
   <xsl:when test="@rend = 'sc'">
     <span class="smallcap"> <xsl:value-of select="."/> </span>
   </xsl:when>
   <xsl:when test="@rend = 'large'"><big><xsl:apply-templates/></big> </xsl:when>
   <xsl:when test="@rend = 'center'">
      <span align="center"><xsl:apply-templates/></span>
   </xsl:when>
   <xsl:when test="@rend = 'underline'">
      <em style="UNDERLINE"><xsl:apply-templates/></em>
   </xsl:when>
   <xsl:when test="@rend = 'tt'"> <tt><xsl:apply-templates/></tt> </xsl:when>
   <xsl:otherwise> <i><xsl:apply-templates/></i> </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Transformation of `<keywords>`. We consider only the `<term>` children, changing the name to `<keyword>` (there should be no other children). Note: in 2006, a test was added, if the value of the term is empty, nothing happend

```
<xsl:template match="keywords">
  <xsl:for-each select="term">
    <xsl:if test="string-length(.)>0">
      <xsl:element name="keyword">
        <xsl:call-template name="id"/>
```

[7] This should be removed for 2005. The error message changed.
[8] There is no apply-templates here; this is wrong (JG).
[9] A paragraph can be centered, as well as a cell in table, but not inline elements like `<hi>`.

```
242          <xsl:value-of select="."/>
243        </xsl:element>
244      </xsl:if>
245    </xsl:for-each>
246  </xsl:template>
```

Transformation of <code>. This is trivial.

```
247  <xsl:template match="code">
248     <xsl:element name="code"> <xsl:apply-templates/> </xsl:element>
249  </xsl:template>
```

Transformation of <ref>. The result is an element of the same name. It has the same id (does anybody reference a reference?). The target attribute is replaced by a xlink:href attribute, with the same value, but it has a # in front. There is also a location attribute whose value is 'intern', except when the parent is a <cit>, case where 'biblio' is used.

```
250  <xsl:template match="ref">
251     <xsl:element name="ref">
252        <xsl:call-template name="id"/>
253        <xsl:attribute name="xlink:href" namespace="http://www.w3.org/1999/xlink">
254           <xsl:value-of select="concat('#', @target)"/>
255        </xsl:attribute>
256        <xsl:attribute name="location">
257           <xsl:choose>
258              <xsl:when test="parent::cit">biblio</xsl:when>
259              <xsl:otherwise>intern</xsl:otherwise>
260           </xsl:choose>
261        </xsl:attribute>
262        <xsl:apply-templates/>
263     </xsl:element>
264  </xsl:template>
```

Transformation of <cit>. We transform only the content, which should be a single <ref>. It is possible to know that the <ref> comes from a <cit> because of its location attribute.

```
265  <xsl:template match="cit">  <xsl:apply-templates/> </xsl:template>
```

Transformation of <xref>. We simplified the code by removing a test that was wrong, hence always false. The result is the same as <ref>, but location is always external, and the link is unchanged. The test that was removed is: if the name of the ancestor of the element is 'citation', then generate the same code, but in <biblScope> element.

```
266  <xsl:template match="xref">
267       <xsl:element name="ref">
268        <xsl:call-template name="id"/>
269        <xsl:attribute name="xlink:href" namespace="http://www.w3.org/1999/xlink">
270           <xsl:value-of select="@url"/>
271        </xsl:attribute>
272        <xsl:attribute name="location">extern</xsl:attribute>
273        <xsl:apply-templates/>
274       </xsl:element>
275  </xsl:template>
```

The <ident> element is unused. It should contain only text.

```
276  <xsl:template match="ident"> <xsl:copy/> </xsl:template>
```

Transformation of <note>; the result is <footnote>.

```
277  <xsl:template match="note">
278     <xsl:element name="footnote"><xsl:copy-of select="@*"/>
279        <xsl:call-template name="id"/>
280        <xsl:apply-templates/>
```

```
281    </xsl:element>
282 </xsl:template>
```

Transformation of <p>. Trivial. Note: we shall see later that there is a second rule for this element.

```
283 <xsl:template match="p">
284    <xsl:element name="p">
285       <xsl:copy-of select="@*" />
286       <xsl:apply-templates/>
287    </xsl:element>
288 </xsl:template>
```

Transformation of <list>. The result is <descriptionlist>, <glosslist>, <orderedlist>, <simplelist>, depending on the value of the type attribute.

```
289 <xsl:template match="list">
290   <xsl:choose>
291    <xsl:when test="@type='description'">
292      <xsl:element name="descriptionlist">
293         <xsl:call-template name="id"/>  <xsl:apply-templates/>
294      </xsl:element>
295    </xsl:when>
296    <xsl:when test="@type='gloss'">
297      <xsl:element name="glosslist">
298        <xsl:call-template name="id"/> <xsl:apply-templates/>
299      </xsl:element>
300    </xsl:when>
301    <xsl:when test="@type='ordered'">
302      <xsl:element name="orderedlist">
303        <xsl:call-template name="id"/> <xsl:apply-templates/>
304      </xsl:element>
305    </xsl:when>
306    <xsl:otherwise>
307      <xsl:element name="simplelist">
308        <xsl:call-template name="id"/> <xsl:apply-templates/>
309      </xsl:element>
310    </xsl:otherwise>
311   </xsl:choose>
312 </xsl:template>
```

Transformation of <item>: the result is <li>.

```
313 <xsl:template match="item">
314    <xsl:element name="li">
315       <xsl:call-template name="id"/> <xsl:apply-templates/>
316    </xsl:element>
317 </xsl:template>
```

Transformation of <label>: the result is <label>.

```
318 <xsl:template match="label">
319    <xsl:element name="label">
320       <xsl:call-template name="id"/>  <xsl:apply-templates/>
321    </xsl:element>
322 </xsl:template>
```

Transformation of <table>. The result is a <table>. We set the attribute border to 'solid' in case one of the cells in the table has a bottom-border attribute that is true.[10] The rend attribute

[10] This is strange; the test should be done on the rows, and all borders.

is copied. We copy all <row> children, followed by the <caption>, if there is one (normally, there is none), followed by <head>, renamed to <caption>[11].

```
<xsl:template match="table">
 <xsl:element name="table">
   <xsl:if test="./row/cell/@bottom-border='true'">
     <xsl:attribute name="border">solid</xsl:attribute>
   </xsl:if>
   <xsl:if test="./@rend">
    <xsl:attribute name="rend"><xsl:value-of select="./@rend" /></xsl:attribute>
   </xsl:if>
   <xsl:call-template name="id"/>
   <xsl:apply-templates select="row" />
   <xsl:apply-templates select="caption"/>
   <xsl:element name="caption"> <xsl:value-of select="head"/> </xsl:element>
  </xsl:element>
</xsl:template>
```

Transformation of <row>. The test here is strange. In the case where the test is false, the result is a <tr>, with the same content as the row. There is one attribute style[12] obtained from the right-border, top-border, left-border, and bottom-border attributes.

```
<xsl:template match="row">
  <xsl:choose>
    <xsl:when test="normalize-space(.) = '' and not(cell/child::*)">

    </xsl:when>
    <xsl:otherwise>
     <xsl:element name="tr">
       <xsl:attribute name="style">
        <xsl:if test="@right-border='true'"
                >border-right-style:solid;border-right-width:1px;</xsl:if>
        <xsl:if test="@top-border='true'"
                >border-top-style:solid;border-top-width:1px;</xsl:if>
        <xsl:if test="@left-border='true'"
                >border-left-style:solid;border-left-width:1px;</xsl:if>
        <xsl:if test="@bottom-border='true'"
                >border-bottom-style:solid; border-bottom-width:1px;</xsl:if>
       </xsl:attribute >
       <xsl:apply-templates/>
     </xsl:element>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

The transformation of <cell> is <td>, with the same content. There is one attribute style obtained from the halign, right-border, top-border, left-border, and bottom-border attributes. Attributes rows and cols are copied if the value is greater than one (this is the row span or column span of the cell).

```
<xsl:template match="cell">
 <xsl:element name="td">
   <xsl:attribute name="style">
    <xsl:if test="@halign"
       >text-align:<xsl:value-of select="@halign"/>;</xsl:if>
```

[11] This seems to be a bug. Tralics converts both 'table' and 'tabular' environments to <table>, in the first case, there is no caption.

[12] The result is so complicated that the XSL/Format style sheet ignores this attribute. This should be changed in 2005.

```
364      <xsl:if test="@right-border='true'"
365         >border-right-style:solid;border-right-width:1px;</xsl:if>
366      <xsl:if test="@top-border='true'"
367         >border-top-style:solid;border-top-width:1px;</xsl:if>
368      <xsl:if test="@left-border='true'"
369         >border-left-style:solid;border-left-width:1px;</xsl:if>
370      <xsl:if test="@bottom-border='true'"
371         >border-bottom-style:solid; border-bottom-width:1px;</xsl:if>
372     </xsl:attribute >
373     <xsl:if test="./@cols>1">
374       <xsl:attribute name="cols"><xsl:value-of select="./@cols" /></xsl:attribute>
375     </xsl:if>
376     <xsl:if test="./@rows>1">
377       <xsl:attribute name="rows"><xsl:value-of select="./@rows" /></xsl:attribute>
378     </xsl:if>
379    <xsl:apply-templates/>
380   </xsl:element>
381 </xsl:template>
```

Attributes halign are always copied. This should be explained, because, a priori, all these attributes were converted to a style attribute.

```
382 <xsl:template match="@halign">
383    <xsl:attribute name="halign"><xsl:value-of select="." /></xsl:attribute>
384 </xsl:template>
```

This converts a <figure> element into a <ressource> element. The rend attribute is copied under the name type. Other attributes like width, height, scale, angle, and framed are just copied.

```
385 <xsl:template name="ressource">
386   <ressource xlink:href="{@file}">
387     <xsl:if test="@rend">
388       <xsl:attribute name="type"><xsl:value-of select="@rend"/></xsl:attribute>
389     </xsl:if>
390     <xsl:if test="@width"> <xsl:copy-of select="@width"/> </xsl:if>
391     <xsl:if test="@height"> <xsl:copy-of select="@height"/> </xsl:if>
392     <xsl:if test="@scale">  <xsl:copy-of select="@scale"/> </xsl:if>
393     <xsl:if test="@angle"> <xsl:copy-of select="@angle"/>  </xsl:if>
394     <xsl:if test="@framed"> <xsl:copy-of select="@framed"/> </xsl:if>
395     <xsl:if test="head and ((ancestor::figure) or not(@file))">
396       <xsl:apply-templates select="head"  mode="caption"/>
397     </xsl:if>
398   </ressource>
399 </xsl:template>
```

In the case where a <figure> is in a <table> which is in a <figure>, and if it has a file attribute, then the result is a 'ressource'.

```
400 <xsl:template match="figure//table//figure[@file]" priority="5">
401   <xsl:call-template name="ressource"/>
402 </xsl:template>
```

In the case where a <figure> has a file attribute, is below a figure, but does not match the rule above, then the result is a 'ressource', as above, but in a <td>.

```
403 <xsl:template match="figure[(ancestor::figure) and @file]">
404   <td><xsl:call-template name="ressource"/></td>
405 </xsl:template>
```

This is the last rule for a <table>. Let's hope no case is forgotten. The result is a <object>. It contains a <table>: in the case where there is a file attribute, the element is empty, and the translation is a <table> with a single <tr> with a single <td> with the ressource. In the case where

there is a <p> with a table, we consider only these elements (let's hope for the best). Otherwise, we add a <table>, and each <p> will produce a row. A caption is put at the end.

```
406 <xsl:template match="figure[not(ancestor::figure)]">
407    <xsl:element name="object">
408       <xsl:call-template name="id"/>
409       <xsl:choose>
410          <xsl:when test="@file">
411             <table>
412               <tr><td>
413               <xsl:call-template name="ressource"/>
414               </td></tr>
415             </table>
416          </xsl:when>
417          <xsl:when test="p/table">
418             <xsl:apply-templates select="p/table" />
419          </xsl:when>
420          <xsl:otherwise>
421             <table> <xsl:apply-templates /> </table>
422          </xsl:otherwise>
423       </xsl:choose>
424       <xsl:apply-templates select="head"  mode="caption"/>
425    </xsl:element>
426 </xsl:template>
```

This code is applied only in the 'otherwise' case of the previous template. For 2005, the best thing to do should be to modify Tralics so that this style sheet can be made more robust.

```
427 <xsl:template match="figure/p">
428    <tr><xsl:apply-templates/></tr>
429 </xsl:template>
```

This copies the id attribute if present.[13]

```
430 <xsl:template name="id">
431  <xsl:if test="./@id">
432    <xsl:attribute name="id"><xsl:value-of select="./@id"/> </xsl:attribute>
433  </xsl:if>
434 </xsl:template>
```

This interprets <head>. If the parent is <list>, we put the content of the element in the title attribute of the current element.[14] If the parent is <figure> or <table> we put the content in a <caption> element. Otherwise, we put it in a <bodyTitle> element. Note that Tralics replaces some empty titles by '(Sans Titre)'; Code on line 447 was changed in 2005: if the section has a single module, the name of the section is used, otherwise 'Introduction' will be used. You will see twice Xsl instead of xsl, in both these cases, the code contained a <xsl:text></xsl:text> that is not shown here (it seems useless to me).

```
435 <xsl:template match="head" mode="caption">
436   <xsl:choose>
437     <xsl:when test="parent::list" >
438       <xsl:attribute name="title"> <xsl:apply-templates/> </xsl:attribute>
439     </xsl:when>
440     <xsl:when test="parent::figure | parent::table">
441       <xsl:element name="caption"> <xsl:apply-templates/> </xsl:element>
442     </xsl:when>
443     <xsl:otherwise>
444       <xsl:element name="bodyTitle">
```

[13] Why not a simple copy-of ?

[14] This is strange; Tralics does not produce such a thing.

```
445         <xsl:choose>
446           <xsl:when test=".='(Sans Titre)'">
447             <xsl:choose>
448               <xsl:when test="count(../../module)=1">
449                 <xsl:value-of select="../../@titre"/>
450               </xsl:when>
451               <xsl:otherwise> <Xsl:text>Introduction</xsl:text></xsl:otherwise>
452             </xsl:choose>
453           </xsl:when>
454           <xsl:otherwise> <Xsl:apply-templates/> </xsl:otherwise>
455         </xsl:choose>
456       </xsl:element>
457     </xsl:otherwise>
458   </xsl:choose>
459 </xsl:template>
```

We do nothing with <head>, because this element should be handled by the routines given above.

```
460 <xsl:template match="head"></xsl:template>
```

We leave the <LaTeX> element unchanged.

```
461 <xsl:template match="LaTeX"> <LaTeX/></xsl:template>
```

We leave the <TeX> element unchanged.

```
462 <xsl:template match="TeX"> <TeX/> </xsl:template>
```

This is the end of the file.

```
463 </xsl:transform>
```

5.2 Addings Ids

There is a style sheet that adds some Ids. Original version (denoted by V1 hereafter) is in add-id.xsl, revised one (denoted by V2) in add-idDTD2.xsl. There is a comment that says: "Some Ids are missing in some XML files (that were not generated by Tralics) these are required in the bibliography, we add them everywhere". The revised version has "they are required for the bibliography and subsection, we add them where needed". As we shall see, Ids are added only for subsections. We propose additional simplifications, see comments below; this gives V3.

This is the header of the file. It declares a namespace (it binds xmlns:m to thez MathML namespace), but this declaration is not used.

```
501 <xsl:transform
502   xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
503   xmlns:m="http://www.w3.org/1998/Math/MathML"
504   exclude-result-prefixes="m">
```

The result is a XML file, conforming the Raweb DTD version 2.

```
505 <xsl:output method='xml'
506      doctype-public="-//INRIA//DTD Raweb 2" doctype-system='raweb2.dtd'
507      indent='yes' encoding='iso-8859-1'/>
```

We do not want to add an ID to each character of a math formula; for this reason we copy recursively the formula. We could remove this code, however a 'diff' between the original XML and the resulting file shows that this code does not a simple copy: each <math> element in the formula has a useless xmlns:xlink attribute, that is removed here. Moreover, the DTD specifies some attributes (like TEIform) with a default value; whenever the attribute is missing, the default value is added; for instance, in the case of the 'apics' file, the size changes from 377517 to 406132 bytes.

```
<xsl:template match="formula">
  <formula>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates mode="math"/>
  </formula>
</xsl:template>
```

This is a copy of a rule defined elsewhere.

```
<xsl:template match="*|@*|comment()|processing-instruction()|text()" mode="math">
 <xsl:copy>
  <xsl:apply-templates mode="math" select="*|@*|processing-instruction()|text()"/>
 </xsl:copy>
</xsl:template>
```

The whole document is converted.

```
<xsl:template match="/">
  <xsl:apply-templates />
</xsl:template>
```

We copy all attributes, and text nodes.

```
<xsl:template match="@*">
  <xsl:copy />
</xsl:template>

<xsl:template match="text()">
  <xsl:copy />
</xsl:template>
```

The default template rule says to copy everything.

```
<xsl:template match="*">
  <xsl:copy>
     <xsl:apply-templates select="node()|@*" />
  </xsl:copy>
</xsl:template>
```

In the case of `<subsection>`, we add an id attribute, if there is none, before copying.

```
<xsl:template match="subsection">
  <xsl:copy>
    <xsl:if test="not(./@id)">
      <xsl:attribute name="id">
        <xsl:value-of select="generate-id()" />
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select="node()|@*" />
  </xsl:copy>
</xsl:template>
```

This is the end of the file.

```
</xsl:transform>
```

The code that follows appears in one of the two style sheets distributed with the Raweb2006; we modified them in order to make the code shorter.

The following rule appears in the style sheets V1 and V2, removed in V3. Its effect is to *not* copy the id attribute; since the purpose of the file is to add missing ids, keeping existing ones, we have to add another rule. According to [5], "Duplicate attribute nodes are removed. If several

attributes in the sequence have the same name, all but the last are discarded". Thus we can safely remove this rule, as well as the additional rules.

```
545 <xsl:template match="@id">
546 </xsl:template>
```

This is the code of V1. The effect is to add an id to every node; but this is overkill.

```
547 <xsl:template match="*">
548   <xsl:copy use-attribute-sets="ID">
549     <xsl:apply-templates select="node()|@*" />
550   </xsl:copy>
551 </xsl:template>
```

This is the code of V2. The effect is a simple copy of the element and its attributes. By default, the id attribute is not copied, hence an explicit copy is needed.

```
552 <xsl:template match="*">
553    <xsl:copy>
554      <xsl:if test="./@id">
555        <xsl:attribute name="id"><xsl:value-of select="./@id" /></xsl:attribute>
556      </xsl:if>
557      <xsl:apply-templates select="node()|@*" />
558    </xsl:copy>
559 </xsl:template>
```

This rule is added in V2; it effectively adds an id to each `<subsection>`. This is equivalent to the V3 code shown above (but V3 does not use the use-attribute-set feature).

```
560 <xsl:template match="subsection">
561   <xsl:copy use-attribute-sets="ADDID">
562     <xsl:apply-templates select="node()|@*" />
563   </xsl:copy>
564 </xsl:template>
```

This rule was named 'ID' in V1 (used on line 548), renamed to 'ADDID' (used on line 561). It uses the 'generate-id' function in order to create a unique id, in the case where there is none.

```
565 <xsl:attribute-set name="ADDID">
566   <xsl:attribute name="id">
567     <xsl:choose>
568       <xsl:when test="./@id"> <xsl:value-of select="./@id" />  </xsl:when>
569       <xsl:otherwise> <xsl:value-of select="generate-id()" />  </xsl:otherwise>
570     </xsl:choose>
571   </xsl:attribute>
572 </xsl:attribute-set>
```

Since putting Ids everywhere is overkill, some templates were added in V2 to inhibit this.

```
573 <xsl:template match="LaTeX"> <LaTeX/> </xsl:template>
574 <xsl:template match="TeX">  <TeX/> </xsl:template>
```

These two variables were used to compute the name of the output file.

```
575 <xsl:variable name="LeProjet"/>
576 <xsl:variable name="year"/>
```

Chapter 6

Converting XML to HTML

The Inria's scientific activity report is constructed from contributions by the research teams. Each team writes a LaTeX file converted to XML by Tralics (or writes it directlty in XML); the file is formed of a number of modules, grouped by sections, and each module is transformed into a HTML page. Each module can be read independently of all others; hence it should be possible to take modules from different teams, and group them in a single page (dynamic version of the RA). On the other hand, the home page of each team has a link to its activity report, formed of these modules, chained in some order (static version of the RA).

The raweb package provides two style sheets that produce HTML from an XML source, they are called dynamic.xsl and static.xsl. The static version is described here, it constructs the document. This is the start of the file.

```
601 <xsl:stylesheet
602   xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
603   xmlns:m="http://www.w3.org/1998/Math/MathML"
604   xmlns:xlink="http://www.w3.org/1999/xlink"
605   xmlns:html="http://www.w3.org/1999/xhtml"
606   exclude-result-prefixes="m html xlink" >
```

Some years ago, it was decided to add the notion of 'topic'. If a team has three main research subjects, each one will be a topic. Thus, a module is associated to a topic and a section, and the reader might want to read the document in section order or in topic order. For this reason, if topics are given, the document is translated twice, the only difference being how individual pages are linked together. The file Topic.xsl contains code to be executed when topics are present, the file sansTopic.xsl contains code for the case without topics, and biblio.xsl contains code for the bibliography. Finally, all other template rules are in common.xsl if they apply as well to the static and non-static case, or are in static.xsl or dynamic.xsl.

```
607   <xsl:import href="common.xsl"/>
608   <xsl:import href="Topic.xsl"/>
609   <xsl:import href="sansTopic.xsl"/>
610   <xsl:import href="biblio.xsl"/>
```

6.1 Common code for HTML conversion

We describe here the code that is either in static.xsl or common.xsl. We start with some boolean variables. These are defined in the imported file.

```
611   <xsl:param name="noframe" select="'0'" />
612   <xsl:param name="notoc" select="'0'" />
```

```
613   <xsl:param name="isTopic" select="'0'" />
614   <xsl:param name="xyleme" select="'0'" />
```

These are defined in the main file. The variable `$xyleme` is set to true in the dynamic version, to false in the static version. The variable `$isTopic` is true if we have topics (it will be set from outside the style sheet). The variable `$noframe` is true if there are no frames (this is the case of the dynamic version); some years ago there were three frames: navigation buttons, TOC and main text, in the current version, there are only two frames, the TOC on the left, the main text of the right. The variable `$notoc` controls whether a TOC should be inserted on each page, assuming that there is no separate frame with the TOC; finally, `$allHtml` is set in the case where the team produces it activity report in HTML, case there conversion from XML to HTML should be inhibited.

```
615   <xsl:param name="xyleme"  select="'0'"/>
616   <xsl:param name="noframe" select="'0'" />
617   <xsl:param name="notoc"   select="'0'" />
618   <xsl:param name="isTopic" select="'0'" />
619   <xsl:param name="allHtml" select="'0'" />
```

A very important point is the difference between a 'Project-Team' and a 'Team'. Of the 160 teams that have written a RA in 2004, 110 were Projects (*Research-teams recognized as project-team by INRIA*, according to the web) at the start of the year. This code uses the isproject attribute of `<identification>`.

```
620 <xsl:variable name="LeTypeProjet">
621   <xsl:if test='/raweb/identification/@isproject="false"'>Team</xsl:if>
622   <xsl:if test='/raweb/identification/@isproject="true"'>Project</xsl:if>
623 </xsl:variable>
```

The main file defines the same variable differently.

```
624 <xsl:variable name="LeTypeProjet">
625  <xsl:choose>
626   <xsl:when test='/raweb/identification/@isproject="true"'>Project-Team</xsl:when>
627   <xsl:otherwise>Team</xsl:otherwise>
628  </xsl:choose>
629 </xsl:variable>
```

The variable `$LeProjet` contains the name of the team and `$DirectoryPasTop` contains the directory of the version without topics.

```
630 <xsl:variable name="LeProjet" select="/raweb/identification/@id"/>
631 <xsl:variable name="DirectoryPasTop" select="$LeProjet"/>
```

More variables, defining essentially paths.

```
632 <xsl:variable name="indexPath">../index.html</xsl:variable>
633 <xsl:variable name="iconspath" select="'../icons'" />
634 <xsl:variable name="imgpath" select="'../'" />
635 <xsl:variable name="helpdir"  select="'..'" />
636 <xsl:variable name="javadir"  select="'..'" />
637 <xsl:param name="displaycd" />
```

This templates adds a target attribute to the current element, unless the value of the parameter `$theTarget` is empty. It is useful if the current element is an anchor.

```
638 <xsl:template name="targetAttrib">
639   <xsl:param name="theTarget" />
640   <xsl:if test="$theTarget!=''">
641     <xsl:attribute name="target"><xsl:value-of select="$theTarget" /></xsl:attribute>
642     </xsl:if>
643 </xsl:template>
```

These three variables are target attributes, used as parameters of the template above.

```
<xsl:variable name="MainwindowTarget" select="'mainraweb06'" />
<xsl:variable name="AltwindowTarget"  select="'_alt'" />
<xsl:variable name="TopwindowTarget"  select="'_top'" />
```

In order to make this document a bit shorter, we have introduced the following templates.

```
<xsl:template name="targetAttrib.alt">
  <xsl:call-template name="targetAttrib">
    <xsl:with-param name="theTarget" select="$AltwindowTarget" />
  </xsl:call-template>
</xsl:template>

<xsl:template name="targetAttrib.top">
  <xsl:call-template name="targetAttrib">
    <xsl:with-param name="theTarget" select="$TopwindowTarget" />
  </xsl:call-template>
</xsl:template>

<xsl:template name="targetAttrib.main">
  <xsl:call-template name="targetAttrib">
    <xsl:with-param name="theTarget" select="$MainwindowTarget" />
  </xsl:call-template>
</xsl:template>
```

The variable `$year` contains the current year or a default value. It is used to construct `$FTPDirectory`, the web location of the Activity Report.

```
<xsl:variable name="year">
 <xsl:choose>
   <xsl:when test='/raweb/@year'>
     <xsl:value-of select="/raweb/@year"/>
   </xsl:when>
   <xsl:otherwise>2006</xsl:otherwise>
 </xsl:choose>
</xsl:variable>
<xsl:variable name="FTPDirectory"
  select="concat('http://www.inria.fr/rapportsactivite/RA', $year,'/',$LeProjet)"/>
```

This line is useless: all pages are created via `<xsl:document>`, and use 'xhtml' as document type.

```
 <xsl:output method='html' encoding='iso-8859-1'
  doctype-public='-//W3C//DTD HTML 4.0//EN'/>
```

6.1.1 The main translation rule

The toplevel element is `<raweb>`, it is translated according to the following rule. The variable `$allHtml` when set to one, inhibits translation, assuming that the HTML version of the document has been produced by other means.

```
<xsl:template match="/raweb">
  <xsl:choose>
    <xsl:when test="$allHtml='1'"/>
    <xsl:otherwise>
...
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

The global action is rather obvious: we call a routine for the title page, one for the presentation, one for each subsection of every main subsection, and one for the bibliography.

```
  <xsl:apply-templates select="identification" mode="statique"/>
  <xsl:apply-templates select="identification/team"  mode="statique"/>
  <xsl:for-each select="presentation|fondements|domaine|logiciels|
          resultats|contrats|international|diffusion">
     <xsl:for-each select="subsection">
        <xsl:apply-templates select="."  mode="statique"/>
     </xsl:for-each>
  </xsl:for-each>
  <xsl:for-each select="biblio">
        <xsl:apply-templates select="."  mode="statique"/>
  </xsl:for-each>
```

In each case, a HTML page is constructed. We start with the easy case: the bibliography. The resulting page is in the directory defined by `$Directory` and is named bibliography.html.

```
<xsl:template match="biblio" mode="statique">
  <xsl:document href="{$Directory}/bibliography.html"
     indent="yes" method="xml"
     doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
     doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html>
      <xsl:apply-templates  select="." />
    </html>
  </xsl:document>
</xsl:template>
```

In the case of a `<subsection>`, we distinguish between sections of level one (originally called `<module>`) or below (`<div2>`, `<div3>`, etc.). This piece of code handles only modules, so that the test is useless: the 'otherwise' is not used.

```
<xsl:template match="subsection" mode="statique">
 <xsl:choose>
  <xsl:when test="not(parent::subsection)">
    <xsl:document href="{$Directory}/{./@id}.html"
      method="xml"  indent="yes" encoding="iso-8859-1"
      doctype-public='-//W3C//DTD XHTML 1.0 Transitional//EN'
      doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
      <html>
        <xsl:apply-templates  select="." />
      </html>
    </xsl:document>
  </xsl:when>
  <xsl:otherwise>
    <html>  <xsl:apply-templates  select="." /> </html>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

The page with the composition of the team is the second page, it is constructed here, by applying a template to the `<team>` element.

```
<xsl:template match="identification/team" mode="statique">
   <xsl:document href="{$Directory}/{@id}.html"
      method="xml"  encoding="iso-8859-1" indent="yes"
      doctype-public='-//W3C//DTD XHTML 1.0 Transitional//EN'
      doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html>
       <xsl:apply-templates select="." />
    </html>
```

```
730   </xsl:document >
731 </xsl:template>
```

Translation of the `<identification>` element consists of four HTML pages, one of them containing the team (previous rule). We show here how the three others are created. If the variable `$noframe` is true (there is some inconsistency here, let's assume that the value is either one, meaning true, or zero, meaning false) we have no frames. If it is false, we have frames, one of them containing the text, the other containing the table of contents.

```
732 <xsl:template match="identification" mode="statique">
733   <xsl:if test="$noframe!='1'">
734     <xsl:call-template name="creer.frameset" />
735   </xsl:if>
```

We construct here the page containg the TOC (tdm in French). This is the only page that has no navigation buttons.

```
736     <xsl:document href="{$Directory}/{$LeProjet}_tdm.html"
737       method="xml" encoding="iso-8859-1" indent="yes"
738       doctype-public='-//W3C//DTD XHTML 1.0 Transitional//EN'
739       doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
740       <html>
741         <xsl:call-template name="page.head" />
742         <body>
743           <xsl:call-template name="tdm" />
744         </body>
745       </html>
746     </xsl:document>
```

This is now the main page. Its name is uid0.html, the assumption being that the first ID generated by Tralics is 'uid1'. Before 2005, the value of the variable `$LeProjet` was used instead. Thus adage2004/adage.html is now replaced by apics/uid0.html.

```
747     <xsl:document href="{$Directory}/uid0.html" method="xml"
748       indent="yes" encoding="iso-8859-1"
749       doctype-public='-//W3C//DTD XHTML 1.0 Transitional//EN'
750       doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
751       <html>
752         <xsl:apply-templates select="." />
753       </html>
754     </xsl:document>
755   </xsl:template>
```

6.1.2 Creating pages

Converting XML to HTML is rather easy, compared to conversion into Pdf. The non-trivial point concerns the layout of the page, navigation buttons, meta data, etc. One question is: should we put navigation buttons on the top and bottom of the page? If a page is large, it can be interesting to have a 'next' button near the end, this avoids the need to scroll to the top, if you want to continue reading. Starting in 1995, the Raweb was produced by `latex2html`, which can conditionnaly insert navigation buttons[1]; but this depends on the number of characters in the page, and not the effective size; as a consequence, the layout of the page seems to be random. The situation changed in 1999[2], the text is in a frame, and the navigation buttons are in an another frame, placed above the text; the buttons appear only in the first frame. In 2003[3] the situation changed again. There

[1] See for instance http://www.inria.fr/rapportsactivite/RA96/algo/algo.html

[2] See for instance http://www.inria.fr/rapportsactivite/RA99/algo/algo.html.

[3] See for instance http://www.inria.fr/rapportsactivite/RA2003/algo2003/algo.html; this is the version without the TOC.

are two possible views. By default, there are no frames, but a button (the 'TOC' button) that creates two frames: the TOC on the left, the text on the right. The text has navigation buttons on the top and the bottom (but there are fewer buttons on the bottom). The same idea is used in 2004 and explained in this document. The first page of each Team contains a link to the previous and next year (when available). See figure 6.2.

Here is a helper for producing an `<img>`. It takes two arguments `$alt` and `$src`. Icons are in a shared directory defined by `$iconpath`.

```
<xsl:template name="icon.image">
  <xsl:param name="alt" />
  <xsl:param name="src" />
  <img align="bottom" border="0" alt="{$alt}" src="{$iconpath}/{$src}.gif />
</xsl:template>
```

The next command takes three arguments, `$nom`, `$position`, and `$accesskey`. If `$nom` is empty, the result is a simple image (for instance named `next_motif_gr.gif`, where 'gr' stands for 'grey' instead of black and white). Otherwise, it is the name of a HTML file, and the result is an anchor `<a>`, whose href attribute is this file name. The image is different (for instance named `next_motif.gif`, the images are designed so that it is obvious that they have the same purpose, one of them being active, the other inactive), but the alt field is the same; the accesskey[4] attribute is set only in this case. In any case, there is some white space after the button. Note: in certain cases, a anchor has a target attribute. This can be `_alt` (this is a name not recognised by the standard; the effect is that the browser opens a new window); this can be `_top` or `_parent` (these names are defined by the standard); it can also be `mainraweb2004` (this is needed for links from the toc to the main frame). The buttons created by this procedure are in the main frame and point to the main frame; they have no target attribute.

```
<xsl:template name="make.icon">
  <xsl:param name="nom" />
  <xsl:param name="position" />
  <xsl:param name="accesskey" />
  <xsl:choose>
    <xsl:when test="$nom=''">
      <xsl:call-template name="icon.image">
        <xsl:with-param name="alt" select="$position" />
        <xsl:with-param name="src" select="concat($position,'_motif_gr')" />
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <a>
        <xsl:attribute name='href'>
           <xsl:call-template name="formaturl">
             <xsl:with-param name="base" select="$nom" />
           </xsl:call-template>
        </xsl:attribute>
        <xsl:if test="$accesskey!=''">
          <xsl:attribute name="accesskey">
            <xsl:value-of select="$accesskey" />
          </xsl:attribute>
        </xsl:if>
        <xsl:call-template name="icon.image">
          <xsl:with-param name="alt" select="$position" />
          <xsl:with-param name="src" select="concat($position,'_motif')" />
        </xsl:call-template>
```

[4] If the acceskey is, say N, pressing down the alt key together with the letter N has the same effect as clicking on the icon, at least with my browser.

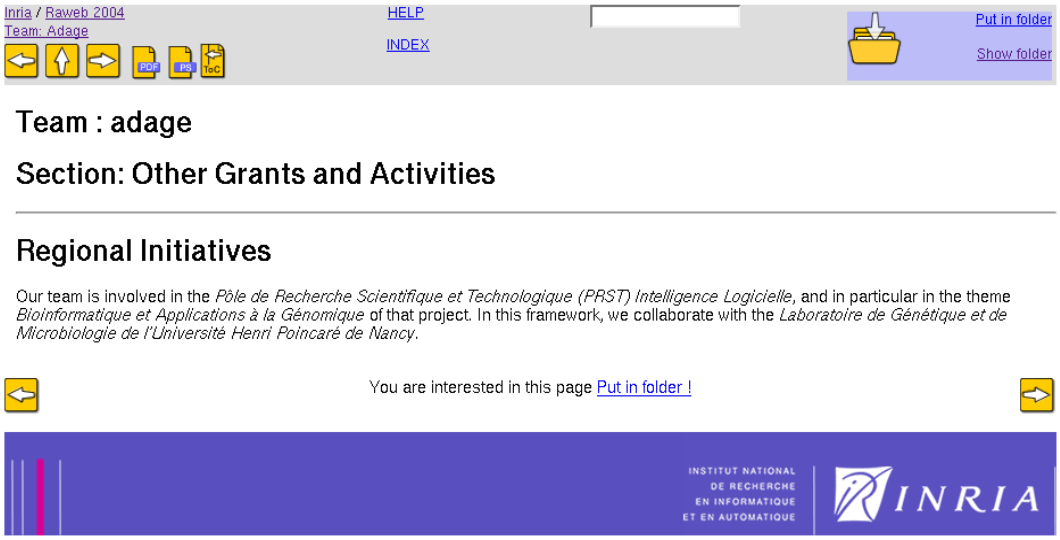


Figure 6.1: The layout of a sample page for a Team without topics.

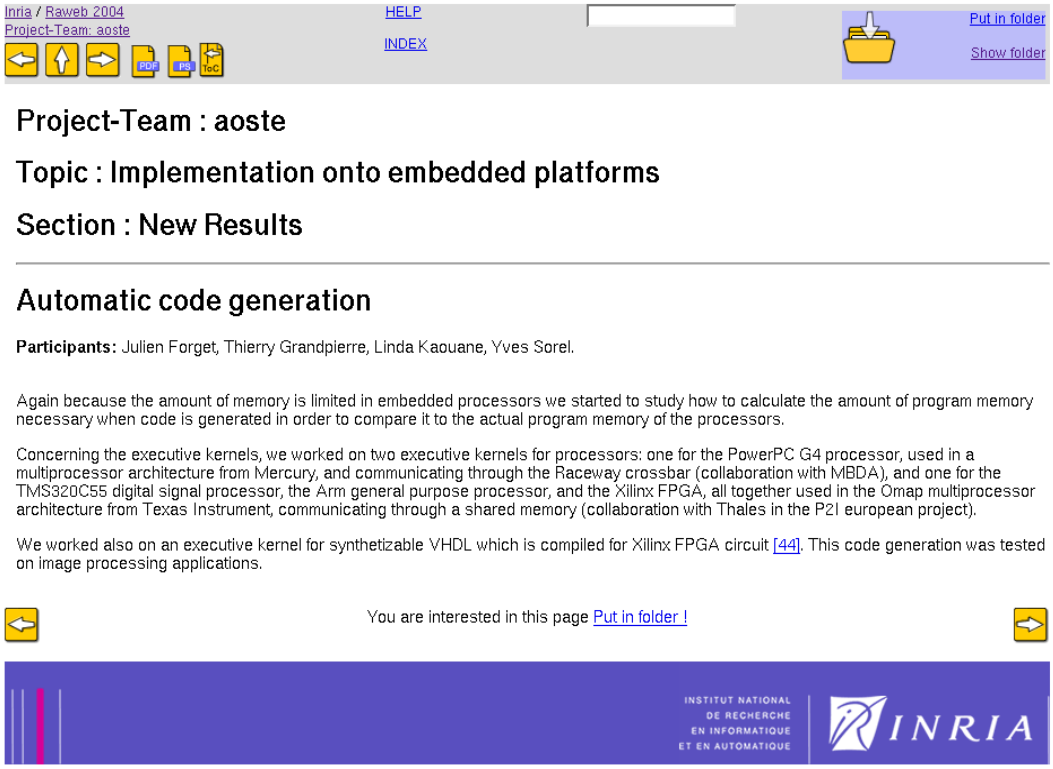


Figure 6.2: The layout of a sample page for a Project-Team with topics.

```
788         </a>
789       </xsl:otherwise>
790     </xsl:choose>
791     <xsl:text>
792
793     </xsl:text>
794   </xsl:template>
```

The next piece of code creates five buttons, and two other items, and puts them in a `<div>`. The first three are created by `make.icon`; this is something that takes three arguments; in order to make this document shorter, we have indicated only the value of the parameters, the names being `$nom`, `$position` and `$accesskey` in order. In the same fashion, `icon.image` is called with two parameters, named `$alt` and `$src`, we show only the value. The three quantities `$precedent`, `$suivant` and `$haut` are arguments to the procedure. They refer to the previous page, next page and top page, which may exist or not; if the page exists, the button is in an anchor, otherwise it is just an image. The following buttons are anchors to '../adage2004/adage.pdf' and '../adage2004/adage.ps.gz', assuming that `$year` contains 2004, and `$LeProjet` contains 'adage'. Since 2005, there is also an anchor to the XML file. These items are followed a link to the TOC and the javascript. Implementation details are given later.

The main Raweb location is http://www.inria.fr/rapportsactivite, this contains a subdirectory for each year, for instance RA2004. In this directory, we have some common files and directories (the css, the icons, etc.), and the teams, for instance adage2004. For a HTML page under adage2004, the next page can be foo.html or ../adage2004/foo.html, the style sheet is in ../raweb.css.

Notes for the 2006 version. This document is written before the RA2006 is put on the web, so that the style sheets are not yet definitive. Nevertheless, it seems that the main directory for a team does not contain the year. Hence the Pdf file is in '../adage/adage.pdf'. In the code shown here, you will see the 2004 anchor, and later on the actual code used in 2006.

```
795 <xsl:template name="page.icons">
796   <xsl:param name="precedent" />
797   <xsl:param name="suivant" />
798   <xsl:param name="haut" />
799   <div class="NavigationIcones">
800     <xsl:call-template name="make.icon">("$precedent", "'previous'", "'P'")
801     </xsl:call-template>
802     <xsl:call-template name="make.icon">("$haut","'up'","'U'")
803     </xsl:call-template>
804     <xsl:call-template name="make.icon">("$suivant","'next'" ,"'N'")
805     </xsl:call-template>
806 <!--    Links to ps, pdf and xml... see below -->
807     <a href="../{$LeProjet}{$year}/{$LeProjet}.pdf"> ...</a>
808     <xsl:text>  </xsl:text>
809     <a href="../{$LeProjet}{$year}/{$LeProjet}.ps.gz"> ... </a>
810     <a href="../{$LeProjet}{$year}/{$LeProjet}.xml"> ... </a>
811     <xsl:call-template name="jg.toclink" />
812     <a id="toclink">...</a>
813     <java ... />
814   </div>
815 </xsl:template>
```

In the code above, we have a comment, followed by two links to the PostScript, Pdf, and XML versions of the document. In each case, we have lines like the following. We call some template with two arguments, the team name and some text (which is in fact a button, created by `icon.image`). The template is defined in the static and dynamic versions.

```
816     <xsl:call-template name="url-pdf-file">
817       <xsl:with-param name="projet" select="$LeProjet" />
```

```
818       <xsl:with-param name="Text">
819          <xsl:call-template name="icon.image"> ("'PDF'","'pdf_motif'")
820          </xsl:call-template>
821       </xsl:with-param >
822     </xsl:call-template>
```

This is the template called in the above code, static version. All three templates `url-ps-file`, `url-pdf-file` and `url-xml-file` are similar. The effect is to create an anchor, depending on the name of the team, containing the text. As you can see, the file name is ../adage/adage.pdf.

```
823 <xsl:template name="url-pdf-file">
824   <xsl:param name="projet" />
825   <xsl:param name="Text" />
826   <a href="../{$LeProjet}/{$LeProjet}.pdf"><xsl:copy-of select="$Text" /></a>
827   </xsl:template>
```

A similar procedure is used to create a link to the Team's homepage. The anchor has a target attribute.

```
828 <xsl:template name="url-fiche-projet">
829   <xsl:param name="FicheProjetName" />
830   <xsl:param name="Text" />
831   <a href="http://www.inria.fr/recherche/equipes/{$FicheProjetName}.en.html" >
832     <xsl:call-template name="targetAttrib.alt" />
833     <xsl:copy-of select="$Text" />
834   </a>
835 </xsl:template>
```

The previous template is called twice. Here is the first occurence. The text is simply 'Project-Team Apics'.

```
836 <xsl:template name="jg.url-fiche-projet-A">
837   <xsl:call-template name="url-fiche-projet-A">
838     <xsl:with-param name="FicheProjetName" select="$FicheProjetName" />
839     <xsl:with-param name="Text">
840       <xsl:value-of select="$LeTypeProjet" />:
841       <xsl:value-of select="/raweb/identification/shortname" />
842     </xsl:with-param>
843   </xsl:call-template>
844 </xsl:template>
```

Here is the second occurence. The text is 'Presentation of the project'.

```
845 <xsl:template name="jg.url-fiche-projet-B">
846   <xsl:call-template name="url-fiche-projet">
847    <xsl:with-param name="FicheProjetName" select="$FicheProjetName" />
848    <xsl:with-param name="Text">
849      <xsl:choose>
850        <xsl:when test='/raweb/identification/@isproject="true"'>
851           Presentation of the project</xsl:when>
852        <xsl:otherwise>Presentation of the team</xsl:otherwise>
853      </xsl:choose>
854    </xsl:with-param>
855   </xsl:call-template>
856 </xsl:template>
```

The semantics of the Raweb says: a team name is formed of letters and digits; a name like `pop_art` is refused (this team did not exist when the rule was established). This rule was never changed, until January 2007 and version 2.9.4 (essentially because underscore characters are not always properly handled by LaTeX and Tralics). This implies that we must compute the real name (used by the procedure given above) and store it in a variable, say `$FicheProjetName`. Question: is there any reason why the template uses a parameter and not the global variable?

```
<xsl:variable name="FicheProjetName">
 <xsl:choose>
  <xsl:when test="string(/raweb/identification/@id)='popart'">pop_art</xsl:whn>
  <xsl:when test="string(/raweb/identification/@id)='led'">langue_et_dialogue</xsl:when>
  <xsl:when test="string(/raweb/identification/@id)='virtualplants'"
     >virtual_plants</xsl:when>
  <xsl:otherwise><xsl:value-of select="/raweb/identification/@id" /></xsl:otherwise>
 </xsl:choose>
</xsl:variable>
```

The page contains an anchor, whose id is 'toclink', and whose href is something as complicated as 'adage_tf.html?../adage2004/uid4.html': after the question mark, there is the address of the current page, and before it is the name of the page with the frames.

```
<xsl:template name="jg.toclink.old">
    <a id="toclink">
       <xsl:attribute name="href">
         <xsl:value-of select="$LeProjet"/>_tf.html?../<xsl:value-of
             select="$LeProjet"/><xsl:value-of select="$year"/>/<xsl:value-of
             select="@id"/>.html</xsl:attribute>
       <img align="bottom" border="0" alt="Access to tdm"
          src="../icons/contents_motif.gif" />
    </a>
</xsl:template>
```

The code changed in 2006. If the variable `$noframe` is zero, meaning that we have frames, the anchor has 'toclink' as id, otherwise it does not. The class of the anchor depends also on this variable (but in all cases the value is the same, so that the test is not shown here). The content of the alt attribute was changed: the French abreviation TDM was replaced by the English equivalent TOC. Finally, the value of the link is computed by some piece of code.

```
<xsl:template name="jg.toclink">
  <a>
    <xsl:if test="$noframe='0'">
      <xsl:attribute name="id">toclink</xsl:attribute>
    </xsl:if>
    <xsl:attribute name="class">toc</xsl:attribute>
    <xsl:attribute name="href">
       <xsl:call-template name="toclink" />
    </xsl:attribute>
    <img align="bottom" border="0" alt="Access to toc" >
        src="{$iconspath}/contents_motif.gif" />
  </a>
</xsl:template>
```

This constructs something like 'adage_tf.html?../adage/uid4.html', without the year. In the case of the main page, we use 'uid0' instead. The link to the TOC depends on whether topics are used or not. However, the code is the same: `toclink_Topic` is identical to `toclink_sansTopic`.

```
<xsl:template name="toclink_Topic">
  <xsl:value-of select="$LeProjet" />
  <xsl:text>_tf.html?../</xsl:text>
  <xsl:value-of select="$LeProjet" />
  <xsl:text>/</xsl:text>
  <xsl:choose>
    <xsl:when test="@id=$LeProjet">uid0</xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="@id" />
    </xsl:otherwise>
  </xsl:choose>
```

```
900   <xsl:text>.html</xsl:text>
901 </xsl:template>
```

Then comes a javascript (it is defined in lib.js). The raweb.css file gives 'display:none' as property for the element identified by the `toclink` id. As a consequence, the button is invisible. However, if the name of the frame is 'mainraweb2004', the code on line 908 sets the 'display' style of elements with id 'toclink' to 'inline', so that the button is visible.

Assume that the button is visible, and you click on it. This will load the 'adage_tf.html' file; its content is explained later; all that you have to know is that the browser contructs a page with two frames, on the left the TOC, on the right is the current page (what follows the question mark); the name of this second frame is 'mainraweb2004', as a consequence the button is invisible. There are two modifications in 2006. The first one is that the frame name is now in the variable `$MainwindowTarget`. The second is that thare are options `$noframe` (if set, there are no frames, hence no possibility to switch) and `$notoc` (if set, there is no table of contents). In these cases, the java script is not executed.

```
902     <xsl:choose>
903       <xsl:when test="$noframe='0' and $notoc='0'" >
904         <script type="text/javascript">
905           <xsl:comment>
906            var cible=this.location;
907            if (self.name != "<xsl:value-of select="$MainwindowTarget" />")
908             changestyle('toclink','inline');
909            //</xsl:comment>
910         </script>
911       </xsl:when>
912     </xsl:choose>
```

The bottom of the page contains only two buttons: previous and next. We have used the same conventions as above. The parameters `$precedent` and `$suivant` contain the locations of the previous and next pages. We insert another item, a javascript. These three items are each in a `<div>`, with an id attribute, the raweb.css file says that these should be flushed left or right, or centered. There is a `<br>` element before and after these three `<div>` elements. There was also an empty `<p>`, removed in 2006.

```
913 <xsl:template name="pagedown.icons">
914     <xsl:param name="precedent" />
915     <xsl:param name="suivant" />
916     <xsl:comment>FIN du corps du module</xsl:comment>
917     <br/>
918     <div id="tail_agauche">
919       <xsl:call-template name="make.icon">("$precedent","'previous'","'P'")
920       </xsl:call-template>
921     </div>
922     <div id="tail_adroite">
923       <xsl:call-template name="make.icon"> ("$suivant","'next'","'N'")
924       </xsl:call-template>
925     </div>
926     <div id="tail_aucentre"> <xsl:call-template name="classeurlink2" /> </div>
927     <br />
928 </xsl:template>
```

The bottom of the page contains, between the previous and next button, a javascript `<div>` that reads: "You are interested in this page Put in folder !".

```
929 <xsl:template name="classeurlink2">
930     <div class="folderLine">
931       <script type="text/javascript" src="{$javadir}/classeur/classeurInOut.js" />
932       <noscript><p>Using javascript allows access to folder</p></noscript>
```

```
933     </div>
934 </xsl:template>
```

The top of the page contains, on the right, a `<div>` element (whose color is a kind of blue, the raweb.css file says it is BCBCF9), with an image and a pointer to the help page for the folder. There is also a `<script>`, that allows you to put the current page in the folder (same action as on the bottom of the page), or to view and manipulate the folder.

```
935 <xsl:template name="classeurlink1">
936   <div class="folderButtons noscript">
937     <a id="folderIconRef" href="../classeur/aide.html">
938       <img id="folderIcon" src="{$iconspath}/folder.gif"
939            alt="icon of folder" border="0"/>
940     </a>
941     <script type="text/javascript" src="{$javadir}:classeur/classeurInOutShow.js" />
942   </div>
943   <noscript> Using JavaScript allows access to folder </noscript>
944 </xsl:template>
```

The middle part of the head of the page consists in two buttons vertically aligned (because of the `<br>`) in a `<small>` element[5]. If you click on them, you get the help, and the index. On the right of these, you will find the search form.

```
945 <xsl:template name="head-middle">
946  <div id="head_aucentre">
947   <xsl:call-template name="formRechercheExalead" />
948   <a href="{$helpdir}/aide.html" target="aide" onclick="displayHelp(this)">HELP</a>
949    <br /> <br />
950    <a href="../index/index.html">
951       <xsl:call-template name="targetAttrib.alt" />
952       INDEX
953    </a>
954  </div>
955 </xsl:template>
```

This constructs the search `<form>`. We do not indicate here all the attributes, and the hidden fields.

```
956 <xsl:template name="formRechercheExalead">
957   <form ...>
958     Search in Activity Report, year <xsl:value-of select="$year" />:<br />
959     <input name="_q" size="20" maxlength="800" value=""/>
960   </form>
961 </xsl:template>
```

Another form. Why are two forms required? The search engine was AltaVista until July 2005, and Exalead after that. The version of the style sheet we present here is dated 2006/09/21. Apparently, this form is used, rather then the preceding one. The difference is tiny (of course, the result of the search engine is completely different, but this has nothing to do with the HTML input).

```
962 <xsl:template name="formRecherche">
963   <form id="recherche" ... >
964     <input name="q1" size="20" maxlength="800" value=""/>
965   </form>
966 </xsl:template>
```

At the very bottom of the page, we have the Inria logo and a link to Inria's home page. The `<div>` and `<span>` elements have an id that is interpreted by the raweb.css style sheet.

[5] The 'small' was removed in 2006

```
<xsl:template name="bandeau_inria">
  <div id="bandeau">
    <a id="bandeau_logo" href="http://www.inria.fr">
       <span id="logotext" >Logo Inria</span>
    </a>
  </div>
</xsl:template>
```

This creates the top of the page. It takes four arguments, namely `$precedent`, `$haut`, `$suivant`, which are the names of the previous, up or next page, and `$couleur`. This last value is not a color, it is a symbolic name (it is 'premiere' or 'autre', French for 'first' and 'other') that will be added as class attribute to the main `<div>` element. This `<div>` element has three `<div>` children, that explain what should be put on the left, the middle, and the right of the page. We have already explained what is on the middle and the right. On the left, there are three lines, separated by `<br>`, using a `<small>` font. The first line contains something like "Inria / Raweb 2004", with two links, the second line contains "Team: Adage", with a link to the team's home page (note that 'Adage' comes from `<shortname>`, and the link points to 'adage', that comes from `$LeProjet`). The last line contains the navigation buttons, created by `page.icons` (arguments are obvious, they are not indicated here, but replaced by a question mark).

```
<xsl:template name="bandeau-sup">
  <xsl:param name="couleur"/>
  <xsl:param name="precedent" />
  <xsl:param name="haut" />
  <xsl:param name="suivant" />
  <div id="toplign">
    <xsl:attribute name="class"><xsl:value-of select="$couleur" /></xsl:attribute>
    <div id="head_agauche">
     <small>
      <a href="http://www.inria.fr">
        <xsl:call-template name="targetAttrib.top" />
        Inria
      </a> /
      <a href="{$indexPath}">Raweb
            <xsl:value-of select="$year" /></a>
      <br />
      <xsl:call-template name="jg.url-fiche-projet-A" />
     </small>
     <br />
     <xsl:call-template name="page.icons"> (?,?,?)
     </xsl:call-template>
    </div>
    <xsl:call-template name="head-middle" />
    <div id="head_adroite"> <xsl:call-template name="classeurlink1" /> </div>
  </div>
</xsl:template>
```

This piece of code constructs the start of a page. In order to make things easier to understand, we split the code in two parts. In the first part, we construct three variables. One contains the authors; this was used for the value `dc.creator`, but in the current version, there is one `dc.creator` per author, and the variable has become useless. The second variable contains the toplevel keywords, from `<keyword>`. The last variable contains the title, from `<bodyTitle>`, preceded by the section title (which is a bit more complicated to compute; this was added in 2006).

```
<xsl:template name="page.head">
  <xsl:param name="title" />
  <xsl:variable name="MTAUT" />
  <xsl:variable name="MTMCL">
```

```
1004        <xsl:for-each select="./keyword">
1005          <xsl:apply-templates /> <xsl:text>/</xsl:text>
1006        </xsl:for-each>
1007      </xsl:variable>
1008      <xsl:variable name="MTDES">
1009        <xsl:if test="name()='identification'">
1010          <xsl:value-of select="projectName"/>
1011        </xsl:if>
1012        <xsl:if test="name()='biblio'"> <xsl:text>Bibliography</xsl:text></xsl:if>
1013        <xsl:if test="name()='team'">  <xsl:text>Members</xsl:text></xsl:if>
1014        <xsl:if test="name(..)='presentation' or  name(..)='fondements' or
1015            name(..)='domaine' or name(..)='logiciels' or name(..)='resultats'
1016            or name(..)='contrats' or  name(..)='international'
1017            or name(..)='diffusion'">
1018          <xsl:value-of select="../bodyTitle"/><xsl:text> - </xsl:text>
1019        </xsl:if>
1020        <xsl:value-of select="./bodyTitle" />
1021      </xsl:variable>
```

Action is trivial. The `<title>` of any HTML page produced by the Adage team in 2004 has the form "Team-Adage", while `dc.title` contains the title of the module. Since 2006, a parameter `$title` was added, so that the title of the HTML page could be "Project-Team-Adage:Introduction"; this is strange in case the parameter is empty.

```
1022      <head>
1023        <title>
1024          <xsl:value-of select="$LeTypeProjet" />-<xsl:value-of
1025                select="/raweb/identification/shortname" />
1026           :<xsl:value-of select="$title" />
1027        </title>
1028        <link rel="Stylesheet" href="../raweb.css" type="text/css" />
1029        <meta name="description" content="{$MTDES}" />
1030        <meta name="dc.title" content="{$MTDES}" />
1031        <xsl:for-each select=".//person">
1032          <meta name="dc.creator">
1033             <xsl:attribute name="content">
1034                <xsl:value-of select="firstname"/>
1035                <xsl:text> </xsl:text>
1036                <xsl:value-of select="lastname" />
1037             </xsl:attribute>
1038          </meta>
1039        </xsl:for-each>
1040        <meta name="dc.subject" content="{$MTMCL}" />
1041        <meta name="dc.publisher" content="INRIA" />
1042        <meta name="dc.date" content="(SCHEME=ISO8601) 2004-01" />
1043        <meta name="dc.type" content="Report" />
1044        <meta name="dc.language" content="(SCHEME=ISO639-1) en" />
1045        <meta name="projet">
1046          <xsl:attribute  name="content">
1047             <xsl:value-of  select="/raweb/identification/shortname" />
1048          </xsl:attribute>
1049        </meta>
1050        <xsl:call-template name="classeurDecl"/>
1051        <xsl:call-template name="interrogationDecl"/>
1052        <script type="text/javascript" src="../lib.js"></script>
1053      </head>
1054    </xsl:template>
```

This declares the three javascripts.

```
<xsl:template name="interrogationDecl">
    <script type="text/javascript" src="{$javadir}/interro.js" />
</xsl:template>
<xsl:template name="classeurDecl">
    <script type="text/javascript"  src="{$javadir}/classeur/classeur.js" />
    <script type="text/javascript"  src="{$javadir}/lib.js" />
</xsl:template>
```

We generate one HTML page for each module, one for the composition of the team (that comes before the first module), and one for the bibliography that comes last. The first page (the title page) will contain the full table of contents. Since 2006, the code is split in two parts; two templates are used for each page, the first has already been described, and we show now the second one, whose objective is to produce the `<head>` and `<body>` of the `<html>` part of the page.

There is a possiblity to generate the Raweb without frames (`$noframe` is true). In general, one frame contains the text and the other one the TOC, thus, in the case of a single frame, there is no direct access to the TOC. For this reason if `$notoc` is false, a copy of the table of contents is added at the start of every page (except the main page that contains the full TOC).

```
<xsl:template name="jg.toc">
  <xsl:if test="$noframe='1' and $notoc='0'">
    <xsl:call-template name="tdm" />
  </xsl:if>
</xsl:template>
```

For the first page, the links to the top and previous pages are identical, they point to the title page (uid0.html), the link to the next page is to the first `<subsection>` (the name of the page is the id attribute, with a '.html' extension). In order to reduce the size of this document, we do not show the parameters to `bandeau-sup` and `pagedown.icons`, the value is obvious, it is replaced by a question mark. The `page.head` template takes a title parameter, not shown here, it is empty.

We have explained above how page headers and footers are created, let's explain here how the body is constructed. First, we extract all `<moreinfo>` elements that come from this element or the `<identification>` part, they are output in a `<blockquote>` element. After that, we insert the title in a `<h1>` heading. This is followed by all the `<participants>` elements; each such element has a category attribute, that gives a subtitle, a `<h3>` element. We convert underscores back to spaces. Each `<person>` of these participants is inserted, with a `<br>` as separator.

```
<xsl:template match="identification/team">
  <xsl:variable name="precedent" select="'uid0'" />
  <xsl:variable name="haut" select="'uid0'" />
  <xsl:variable name="suivant" select="/raweb/*/subsection[1]/@id" />
  <xsl:call-template name="page.head" />
  <body>
    <xsl:call-template name="jg.toc"/>
    <xsl:call-template name="bandeau-sup">
      <xsl:with-param name="couleur" select="'autre'" /> (?,?,?)
    </xsl:call-template>
    <div id="main">
      <xsl:for-each select="/raweb/identification/moreinfo">
        <blockquote> <xsl:apply-templates /> </blockquote>
      </xsl:for-each>
      <xsl:for-each select="./moreinfo">
          <blockquote> <xsl:apply-templates /> </blockquote>
      </xsl:for-each>
      <h1>Team</h1>
      <xsl:for-each select="participants">
        <h3> <xsl:value-of select="translate(@category, '_', ' ')" /> </h3>
```

```
1087          <xsl:for-each select="person">
1088             <xsl:apply-templates select="." />  <br />
1089            </xsl:for-each>
1090        </xsl:for-each>
1091        <br />
1092        <xsl:call-template name="pagedown.icons"> (?,?)
1093        </xsl:call-template>
1094        <xsl:call-template name="bandeau_inria"/>
1095      </div>
1096    </body>
1097  </xsl:template>
```

This constructs the page with the bibliography.

```
1098  <xsl:template match="biblio">
1099    <xsl:variable name="precedent"
1100         select="preceding-sibling::*[1]/subsection[position()=last()]/@id"/>
1101    <xsl:variable name="haut" select="'uid0'"/>
1102    <xsl:variable name="suivant" select="''"/>
1103       <xsl:call-template name="page.head"/>
1104       <body>
1105         <xsl:call-template name="jg.toc"/>
1106         <xsl:call-template name="bandeau-sup">(?,?,?)
1107            <xsl:with-param name="couleur" select="'autre'" />
1108         </xsl:call-template>
1109         <h1> Bibliography </h1>
1110         <xsl:call-template name="tri-par-publis"/>
1111         <xsl:call-template name="pagedown.icons"> (?,?)
1112         </xsl:call-template>
1113         <xsl:call-template name="bandeau_inria"/>
1114       </body>
1115  </xsl:template>
```

In the case of `<subsection>`, we have to distinguish between modules and normal sections. A module is a subsection whose parent is not a subsection, and a full HTML page must be created. In this case, we generate a page, like above (arguments of `bandeau-sup` and `pagedown.icons` are not shown, the non-trivial part is to find the previous and next pages, in `$precedent` and `$suivant`). The page body contains the following: it starts with a title (for instance "Team : adage" in a `<h1>`, followed by the title of the section (it could be "Section: Overall Objectives" in a `<h2>`, this depends on whether `$isTopic` is set), followed by a horizontal rule, a `<hr>`. This is followed by the title (value of `<bodyTitle>`) in a `<h3>` and the keywords[6]. Then comes the content of the subsection.

```
1116  <xsl:template match="subsection">
1117   <xsl:variable name="haut" select="'uid0'" />
1118   <xsl:variable name="precedent"><xsl:call-template name="precedent" /> </xsl:variable>
1119   <xsl:variable name="suivant"><xsl:call-template name="suivant" /></xsl:variable>
1120    <xsl:choose>
1121     <xsl:when test="not(parent::subsection)">
1122       <xsl:call-template name="page.head" >
1123         <xsl:with-param name="title">
1124           <xsl:value-of select="./bodyTitle" />
1125         </xsl:with-param>
1126       </xsl:call-template>
1127       <body>
1128         <xsl:call-template name="jg.toc"/>
1129         <xsl:call-template name="bandeau-sup">(?,?,?)
```

[6]The order of these two items changed between 2005 and 2006

```
          <xsl:with-param name="couleur" select="'autre'" />
        </xsl:call-template>
        <div id="main">
          <xsl:call-template name="use-id" />
          <h1>
            <xsl:value-of select="$LeTypeProjet" />
            <xsl:text> : </xsl:text>
            <xsl:value-of select="$LeProjet" />
          </h1>
          <xsl:choose>
            <xsl:when test="$isTopic='1'">
              <xsl:call-template name="section_title_Topic" />
            </xsl:when>
            <xsl:otherwise>
              <xsl:call-template name="section_title_sansTopic" />
            </xsl:otherwise>
          </xsl:choose>
          <hr />
          <xsl:apply-templates select="bodyTitle" mode="titre3"/>
          <xsl:call-template name="jg.keywords" />
          <xsl:apply-templates />
        <xsl:call-template name="pagedown.icons"> (?,?)
        </xsl:call-template>
        <xsl:call-template name="bandeau_inria"/>
        </div>
      </body>
    </xsl:when>
```

In the case where `<subsection>` is in a subsection, we output its title (value of `<bodyTitle>`) using `<h4>` or `<h5>`, depending on whether the parent is in a subsection, then the keywords, then the content.

```
    <xsl:otherwise>
      <xsl:call-template name="use-id" />
      <xsl:choose>
        <xsl:when test="(../parent::subsection)">
          <xsl:apply-templates select="bodyTitle" mode="titre5"/>
        </xsl:when>
        <xsl:otherwise>
           <xsl:apply-templates select="bodyTitle" mode="titre4"/>
        </xsl:otherwise>
      </xsl:choose>
      <xsl:call-template name="keywords-list" />
      <xsl:apply-templates />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This piece of code creates the title page, the TOC, and the frameset. In the case of the adage team, the files are adage.html, adage_tdm.html, and adage_tf.html in the adage2004 directory. Let's consider the title page first. It looks like a normal page, except that `bandeau_inria` (the banner with the logo) is before the text.

```
<xsl:template match="identification">
  <xsl:variable name="precedent" select="''" />
  <xsl:variable name="haut" select="'uid0'" />
  <xsl:variable name="suivant" select="team/@id" />
  <xsl:call-template name="page.head" />
  <body>
```

```
1178      <xsl:call-template name="bandeau-sup">
1179        <xsl:with-param name="couleur" select="'premiere'" /> (?,?,?)
1180      </xsl:call-template>
1181      <xsl:call-template name="bandeau_inria"/>
1182      <xsl:comment>DEBUT du corps du module</xsl:comment>
1183      <xsl:call-template name="jg.titlepage"/>
1184      <hr class="rose"/>
1185      <xsl:call-template name="table.matieres" />
1186      <xsl:call-template name="pagedown.icons"> (?,?)
1187      </xsl:call-template>
1188   </body>
1189 </xsl:document>
```

The title page is divided in two parts, the first part contains the identification of the team, it is followed by the TOC. We start with a `<h1>` element that contains the full name of the team, from `<projectName>`. This is followed by the short name, from `<shortname>`. It is followed by "2006 research project activity reports", a link to the UR (Research Unit), a link to the team, to the PostScript, Pdf and XML versions of the report.

```
1190 <xsl:template name="jg.titlepage"/>
1191   <h1 class="center"> <xsl:value-of select="projectName" /> </h1>
1192   <div class="entete">
1193     <div class="bigspace"><h2><xsl:value-of select="shortname" /></h2></div>
1194     <xsl:value-of select="$year" /> research project activity reports
1195     <xsl:call-template name="UR"/>
1196     <div class="bigspace">
1197       <xsl:call-template name="jg.find-theme"/>
1198     </div>
1199     <xsl:call-template name="jg.url-fiche-projet-B" />
1200     <xsl:call-template name="url-ps-file">
1201       <xsl:with-param name="projet" select="$LeProjet" />
1202       <xsl:with-param name="Text">PostScript</xsl:with-param>
1203     </xsl:call-template>
1204     <xsl:text>, </xsl:text>
1205     <xsl:call-template name="url-pdf-file">
1206       <xsl:with-param name="projet" select="$LeProjet" />
1207       <xsl:with-param name="Text">PDF</xsl:with-param>
1208     </xsl:call-template>
1209     <xsl:text> or </xsl:text>
1210     <xsl:call-template name="url-xml-file">
1211       <xsl:with-param name="projet" select="$LeProjet" />
1212       <xsl:with-param name="Text">XML</xsl:with-param>
1213     </xsl:call-template>
1214     <xsl:text> format</xsl:text>
1215   </div>
1216 </xsl:template>
```

This is the piece of code that inserts the theme, a bit arranged for clarity. It depends on whether `$xyleme` is true. The value of the variable is something like 'num' in the XML file produced by Tralics, converted to 'NUM' (all uppercase) in the new DTD. We convert it to 'Num' for the link[7].

```
1217 <xsl:template name="jg.find-theme"/>
1218   <xsl:variable name="path"
1219      select="http://www.inria.fr/recherche/equipes/listes/theme_"/>
1220   <xsl:variable name="themeurl">
1221     <xsl:choose>
```

[7] This is new in 2005; but theme_NUM.html and theme_Num.html are handled the same by the Web server.

```
1222        <xsl:when test="$xyleme='1'">
1223          <xsl:value-of select="$indexPath" />
1224          <xsl:text>?theme=</xsl:text/>
1225          <xsl:value-of select="substring(theme,1,3)" />
1226        </xsl:when>
1227        <xsl:otherwise>
1228          <xsl:value-of select="$path" />
1229          <xsl:value-of select="substring(theme,1,1)"/>
1230          <xsl:value-of select="translate(substring(theme,2,3),
1231               'ABCDEFGHIJKLMNOPQRSTUVWXYZ','abcdefghijklmnopqrstuvwxyz')" />
1232          <xsl:text>.en.html</xsl:text>
1233        </xsl:otherwise>
1234      </xsl:choose>
1235    </xsl:variable>
1236    Theme:
1237    <a href="{$themeurl}">
1238      <xsl:call-template name="targetAttrib.alt" />
1239      <xsl:value-of select="theme" />
1240    </a>
1241  </xsl:template>
```

Translation of <UR>. We look at the name attribute. The code is easy, perhaps a bit longish (we do not show the complete code here). We test the value against Rocquencourt, Rennes, Sophia, Lorraine, RhoneAlpes and Futurs.

```
1242  <xsl:template name="UR">
1243   <xsl:variable name="orga">http://www.inria.fr/inria/organigramme/fiche</xsl:variable>
1244   <div class="bigspace">
1245    <xsl:for-each select="UR">
1246      <i>
1247      <xsl:choose>
1248       <xsl:when test="@name='Rocquencourt'">
1249        <a href="{$orga}_ur-rocq.en.html">
1250          <xsl:call-template name="targetAttrib.alt" />
1251          <xsl:value-of select="@name"/>
1252        </a>
1253       </xsl:when>
1254        <!-- other cases are similar, not shown -->
1255       <xsl:otherwise>
1256         <a href="http://www.inria.fr/inria/organigramme" >
1257           <xsl:call-template name="targetAttrib.alt" />
1258             INRIA</a>
1259       </xsl:otherwise>
1260      </xsl:choose>
1261     </i>
1262     <xsl:if test="position()!=last()"> - </xsl:if>
1263    </xsl:for-each>
1264   </div>
1265  </xsl:template>
```

This contains the HTML page with the frameset. An interesting point is that this contains the table of contents, for the case where frames are refused. Note that this is an HTML document, not an XML one (the method attribute is different); more important, all other HTML files have XHTML1.0 as doctype, this one has HTML4.01. The document contains two frames; one of them points to an HTML page whose name is dynamically constructed. The idea is the following. Normally, the `location.search` variable is empty and the result of line 1282 is 'uid0.html'; however, in the case of an URL like line 869 or 891, that contains a question mark, the variable is not empty, what follows the question mark is used.

```
<xsl:template name="creer.frameset">
 <xsl:document href="{$Directory}/{./@id}_tf.html" method="html" encoding="iso-8859-1"
  doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"
  doctype-system="http://www.w3.org/TR/html4/loose.dtd">
  <html>
   <head>
     <title>
       <xsl:value-of select="$LeTypeProjet" />:
       <xsl:value-of select="$LeProjet" />
     </title>
     <meta name="Robots" content="noindex" />
     <link rel="stylesheet" href="../raweb.css" type="text/css" />
     <style type="text/css"> div.tdmdiv { width:100% } </style>
     <script type="text/javascript">
        contenuSRC = (location.search.substring(1))
          ?location.search.substring(1) : 'uid0.html';
        contenuSRC = unescape(contenuSRC);
        var writeFrame = '';
        writeFrame += '&lt;frameset COLS="235,*"&gt;';
        writeFrame += '&lt;frame src="<xsl:value-of select='$LeProjet' />_tdm.html"&gt;';
        writeFrame += '&lt;frame src="' + contenuSRC +
                      '" name="mainraweb06" SCROLLING="auto" &gt;';
        writeFrame += '&lt;\/frameset&gt;';
        document.write(writeFrame);
     </script>
   </head>
   <noscript>
     <h1 class="warning">Using javascript is better to read this Activity Report</h1>
     <xsl:call-template name="tdm" />
   </noscript>
  </html>
 </xsl:document>
</xsl:template>
```

6.1.3 Titles, keywords, persons

This outputs the current section title, the `<bodyTitle>` of the parent, using `<h2>` as heading.

```
<xsl:template name="section_title_sansTopic">
  <h2>Section: <xsl:value-of select="../bodyTitle" /></h2>
</xsl:template>
```

This outputs the current title, found in `<bodyTitle>`, using `<h2>`, `<h3>`, `<h4>`, or `<h5>` as heading.

```
<xsl:template match="bodyTitle" mode="titre2">
  <h2 class="titre2"> <xsl:apply-templates/></h2>
</xsl:template>
<xsl:template match="bodyTitle" mode="titre3">
  <h3 class="titre3"> <xsl:apply-templates/></h3>
</xsl:template>
<xsl:template match="bodyTitle" mode="titre4">
  <h4 class="titre4"><xsl:apply-templates/> </h4>
</xsl:template>
<xsl:template match="bodyTitle" mode="titre5">
  <h5 class="titre5"><xsl:apply-templates/> </h5>
</xsl:template>
```

In the case of topics, the layout is a bit different, and titles are handled by the routines that follow. At level one, the result is a `<h1>`. If the title is missing, it will be replaced by the title of the section (presentation, fondements, etc.). The result is empty if the parent has no topic. Note: Tralics refuses to create a section with an empty title. In some cases, for instance if this is the first module in a sequence of more than one modules, the title will be 'Introduction'. Otherwise, it will be '(Sans Titre)' and the style sheet is assumed to do something in this case.

```
<xsl:template match="(presentation|fondements|domaine|logiciels
  |resultats|contrats|international|diffusion)/subsection/bodyTitle"
    priority="1">
    <xsl:call-template name="use-id" />
    <xsl:if test="../subsection/@topic">
      <h1> <xsl:apply-templates /> </h1>
    </xsl:if>
</xsl:template>
```

A title is always output by the routines shown above, hence the default action is empty.

```
<xsl:template match="bodyTitle" />
```

This is how we typeset a title in the TOC. We use a `<li>` element and a link to the page. This fills the anchor (text and attributes). If the title is empty, the title of the parent is used instead.

```
<xsl:template name="item_title_or_not_title">
  <xsl:call-template name="targetAttrib.main" />
  <xsl:attribute name='href'>
    <xsl:call-template name="formaturl">
      <xsl:with-param name="base" select="@id" />
    </xsl:call-template>
  </xsl:attribute>
  <xsl:choose>
    <xsl:when test="bodyTitle">
      <xsl:value-of select="./bodyTitle" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="../bodyTitle" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Section titles are typeset twice: in the text, and in the TOC. When we are in the TOC, ids are not added. This is not very important for the HTML, because the TOC is a separated file. For the Pdf version it is necessary; it could be necessary if we decided to add a local TOC to each page (à la minitoc, this was done, for instance in 1996).

```
<xsl:template match="bodyTitle" mode="tocsection">
  <xsl:apply-templates mode="section" />
</xsl:template>
```

This rule says that keywords are handled via `keywords-list`, except if we are in a toplevel subsection that has subsections, case where `keywords-list2` is used instead.

```
<xsl:template name="jg.keywords">
   <xsl:choose>
      <xsl:when test="not(./subsection)">
          <xsl:call-template name="keywords-list" />
      </xsl:when>
      <xsl:otherwise>
          <xsl:call-template name="keywords-list2" />
      </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

We modified a bit the code that follows: there was a `<p>`, moved inside the template `listvir`[8]. In general, when a keyword is seen, we take all `<keyword>` elements from the subtree, and pass them to the template.

```
<xsl:template name="keywords-list">
  <xsl:if test=".//keyword">
     <xsl:call-template name="listvir">
       <xsl:with-param name='liste' select="(.//keyword)" />
       <xsl:with-param name='deco' select="string('keyword')" />
     </xsl:call-template>
  </xsl:if>
</xsl:template>
```

However, in the case of a toplevel subsection (a module) with subsections, the action is different: we take only the `<keyword>` elements of this subsection.

```
<xsl:template name="keywords-list2">
 <xsl:if test="./keyword">
    <xsl:call-template name="listvir">
      <xsl:with-param name='liste' select="(./keyword)" />
      <xsl:with-param name='deco' select="string('keyword')" />
    </xsl:call-template>
 </xsl:if>
</xsl:template>
```

This is the action when we have a list of keywords to convert. The test (is there a keyword on the tree?) should probably be replaced by a better one (is the list empty?). If `$xyleme` is true, each keyword has a link (code omitted).

```
<xsl:template name="listvir">
   <xsl:param name="liste" />
   <xsl:param name="deco" />
   <p>
     <xsl:if test=".//keyword"> <span clas="KW">Keywords</span> : </xsl:if>
     <xsl:for-each select="$liste" >
       <span class="{$deco}"><xsl:apply-templates /></span>
       <xsl:call-template name="separateur.objet" />
     </xsl:for-each >
   </p>
</xsl:template>
```

The previous routines takes all keywords, so that the default action is empty.

```
<xsl:template match="keyword">  </xsl:template>
```

In the presentation part, a `<person>` is handled by this code. We output the `<firstname>`, a space, the `<lastname>`, and the `<moreinfo>`, provided that this is not empty. Brackets are added. Since 2006, three new fields were added, only `<hdr>` produces something. Assume that our guy is called Jean Dupont, the result is then one of 'Jean Dupont [HdR]', 'Jean Dupont [CR, HdR]', 'Jean Dupont [CR]', 'Jean Dupont'. Each bracket is preceded and followed by space, 'CR' is the content of the `<moreinfo>` element, and 'HdR' is produced by the templates given below (we gave a name to `jg.hdr`, this makes the code easier to understand). We also removed a useless test (it tested that the element is non-empty, instead of testing `<moreinfo>`, but we know that `<moreinfo>`, when present, is non-empty).

```
<xsl:template match="person">
  <xsl:value-of select="./firstname" />
  <xsl:text> </xsl:text>
  <xsl:value-of select="./lastname" />
  <xsl:choose>
```

[8] This is a strange name. It has no obvious meaning

```
1385      <xsl:when test="moreinfo">
1386        <xsl:text> [ </xsl:text>
1387        <xsl:apply-templates select="./moreinfo/node()" />
1388        <xsl:apply-templates select="hdr" />
1389        <xsl:text> ] </xsl:text>
1390      </xsl:when>
1391      <xsl:otherwise>
1392        <xsl:call-templates select="jg.hdr" />
1393      </xsl:otherwise>
1394    </xsl:choose>
1395  </xsl:template>
```

Both templates shown here produce the string 'habilité(e)' which is non-sense in an English document. This will change in the final version of the RA2006, using the acronym 'HdR'. The first template is called when there is no `<moreinfo>` in the current `<pers>`, it outputs the funny string with the brackets if `<hdr>` is present; the second one is called if there is a `<moreinfo>`, it just outputs the funny string, preceded by a comma.

```
1396  <xsl:template name="jg.hdr">
1397    <xsl:if test="hdr">
1398      <xsl:text> [ habilité(e) ] </xsl:text>
1399    </xsl:if>
1400  </xsl:template>
1401  <xsl:template match="hdr">
1402    <xsl:text>, </xsl:text>
1403    <xsl:text> habilité(e) </xsl:text>
1404  </xsl:template>
```

In the other sections, a `<person>` is similarly handled, but the code is a bit different. If `$xyleme` is set, an anchor is added (this is the same as for keywords, code not shown here). There are no additional fields like `<hdr>` to consider here.

```
1405  <xsl:template name="xperson">
1406    <xsl:value-of select="./firstname"/>
1407    <xsl:text> </xsl:text>
1408    <xsl:value-of select="./lastname"/>
1409    <xsl:apply-templates select="moreinfo" mode="inpers"/>
1410  </xsl:template>
```

If you look carefully, you can see that spaces around square brackets are different here and in the previous case.

```
1411  <xsl:template match="moreinfo" mode ="inpers">
1412    <xsl:if test="not(normalize-space(string(.)) =’’)">
1413        <xsl:text> [</xsl:text><xsl:apply-templates/><xsl:text>]</xsl:text>
1414    </xsl:if>
1415  </xsl:template>
```

This takes a list as argument, it outputs an s if there are more than one element in the list.

```
1416  <xsl:template name="pluriel-p">
1417    <xsl:param name="liste" />
1418    <xsl:if test="count($liste)>1">s</xsl:if>
1419  </xsl:template>
```

In a `<subsection>`, the translation of `<participants>` is formed of every `<person>`, with comma as separator.

```
1420  <xsl:template match="subsection//participants">
1421    <p class="participants">
1422      <span class="part">Participant
1423           <xsl:call-template name="pluriel-p">
```

```
            <xsl:with-param name="liste" select="person" />
          </xsl:call-template>
    </span> :
    <xsl:for-each select="person">
      <xsl:call-template name="xperson" />
      <xsl:call-template name="separateur.objet" />
    </xsl:for-each>
  </p>
</xsl:template>
```

6.1.4 Other elements

Let's consider some trivial commands. We leave `<i>` unchanged.

```
<xsl:template match="i">
  <i><xsl:apply-templates /></i>
</xsl:template>
```

Ditto for `<b>`.

```
<xsl:template match="b">
 <b> <xsl:apply-templates /> </b>
</xsl:template>
```

Ditto for `<tt>`.

```
<xsl:template match="tt">
  <tt><xsl:apply-templates /> </tt>
</xsl:template>
```

Ditto for `<small>`.

```
<xsl:template match="small">
  <small> <xsl:apply-templates /> </small>
</xsl:template>
```

Ditto for `<big>`.

```
<xsl:template match="big">
 <big> <xsl:apply-templates /> </big>
</xsl:template>
```

Ditto for `<sup>`.

```
<xsl:template match="sup">
  <sup> <xsl:apply-templates /> </sup>
</xsl:template>
```

Ditto for `<sub>`.

```
<xsl:template match="sub">
  <sub> <xsl:apply-templates /> </sub>
</xsl:template>
```

This code is the same as above. This is used for evaluation of a title in the TOC.

```
<xsl:template match="sup" mode="section">
  <sup> <xsl:apply-templates /> </sup>
</xsl:template>
<xsl:template match="sub" mode="section">
  <sub> <xsl:apply-templates /> </sub>
</xsl:template>
```

The translation of `<em>` is a `<i>` in the case where the style attribute is 'UNDERLINE', is 'HIGHLIGHT' or is something else. We simplified a bit the code.

```
<xsl:template match="em">
    <i> <xsl:apply-templates /> </i>
</xsl:template>
```

This could be used by Xyleme.

```
<xsl:template match="highlight">
   <span class="xyHighlight">
     <xsl:apply-templates />
   </span>
</xsl:template>
```

The translation of `<span>` is the same element. We copy the class attribute, but not the other ones (why?).

```
<xsl:template match="span">
 <span class="{@class}"> <xsl:apply-templates /> </span>
</xsl:template>
```

We could do better.

```
<xsl:template match="center">
  <xsl:apply-templates />
</xsl:template>
```

A `<term>` is converted into a `<tt>`.

```
<xsl:template match="term">
 <tt> <xsl:apply-templates /> </tt>
</xsl:template>
```

A `<simplelist>` is converted into a `<ul>`.

```
<xsl:template match="simplelist">
  <ul> <xsl:apply-templates />  </ul>
</xsl:template>
```

A `<orderedlist>` is converted into a `<ol>`.

```
<xsl:template match="orderedlist">
  <ol> <xsl:apply-templates /> </ol>
</xsl:template>
```

A `<descriptionlist>` is converted into a `<dl>`.

```
<xsl:template match="descriptionlist">
  <dl> <xsl:apply-templates /> </dl>
</xsl:template>
```

Elements `<glosslist>` or `<descriptionlist>` should contain a `<label>`, a `<li>`, a `<label>`, a `<li>`, etc. These are converted into a `<dd>` or `<dd>` in the obvious way.

```
<xsl:template match="glosslist/label | descriptionlist/label">
  <dt> <xsl:apply-templates /> </dt>
</xsl:template>
<xsl:template match="glosslist/li | descriptionlist/li">
  <dd> <xsl:apply-templates /> </dd>
  </xsl:template>
```

A `<li>` is converted into a `<li>` in cases not shown above. If preceded by a `<label>`, the value of the label is added.

```
<xsl:template match="li">
  <li>
    <xsl:apply-templates select="@*" />
    <xsl:apply-templates mode="li" select="preceding-sibling::label[position()=1]" />
    <xsl:apply-templates />
```

```
1497    </li>
1498  </xsl:template>
```

Two rules are neeed: one that says that `<label>` should be ignored, and one that say that it should be translated (from inside the `<li>`, where the mode is correct).

```
1499  <xsl:template match="label" />
1500
1501  <xsl:template match="label" mode="li">
1502    <xsl:apply-templates />
1503  </xsl:template>
```

A `<glosslist>` is converted into a `<dl>`. It has a title, is put in a `<div>`, preceded and followed by a horizontal rule, a `<hr>` element.

```
1504  <xsl:template match="glosslist">
1505    <hr />
1506    <div id="glossaire" style="margin-left: 2em;margin-right: 2em;">
1507      <dl title="Glossary"> <xsl:apply-templates /> </dl>
1508     </div>
1509    <hr />
1510  </xsl:template>
```

The translation of `<moreinfo>` is a `<blockquote>`, unless it is in a `<pers>`, case where the code on lines 1387 or 1409 applies.

```
1511  <xsl:template match="moreinfo">
1512    <blockquote>  <xsl:apply-templates /> </blockquote>
1513  </xsl:template>
```

Action is empty here. Normally, there should be no `<moreinfo>` in `<raweb>`, it should be in `<identification>`.

```
1514  <xsl:template match="/raweb/moreinfo" priority="1"/>
```

The translation of `<simplemath>` is `<i>`. Such an element is generated by Tralics in the case of a trivial formula like `$x$`.

```
1515  <xsl:template match="simplemath">
1516    <i> <xsl:apply-templates /> </i>
1517  </xsl:template>
```

Math formulas are copied verbatim, with their context. We assume that either the browser understands MathML or that a postprocessor converts the math.

```
1518  <xsl:template match="m:math">
1519    <xsl:copy>
1520        <xsl:apply-templates mode="math" />
1521    </xsl:copy>
1522  </xsl:template>
1523
1524  <xsl:template mode="math" match="*|@*|text()">
1525   <xsl:copy>
1526      <xsl:apply-templates mode="math" select="*|@*|text()" />
1527   </xsl:copy>
1528  </xsl:template>
```

This is for the case where the math appears in a title in the TOC.

```
1529  <xsl:template match="m:math" mode="section">
1530    <m:math>
1531        <xsl:copy-of select="@*" />
1532        <xsl:apply-templates mode="math" />
1533     </m:math>
1534  </xsl:template>
```

Translation of `<formula>`. It depends on the type attribute. If this is not 'display', the result is a simple `<span>`, with attribute class = 'math'. If the type is 'display', the result is a `<div>`, with attribute class = 'mathdisplay', and align = 'center'. Moreover, if there is an id, an equation number is added. For this reason a `<table>` is created, with a single `<tr>` and two `<td>`, first the formula, centered, then its number, right aligned.

```
<xsl:template match="formula">
 <xsl:choose>
  <xsl:when test="@type = 'display' and @id">
   <div align="center" class="mathdisplay">
     <xsl:call-template name="use-id" />
     <table width='100%'>
      <tr valign='middle'>
       <td align='center'> <xsl:apply-templates /> </td>
       <td class="eqno" width="10" align="right">(<xsl:number
                                level="any" count="formula[@id]" />)</td>
      </tr>
     </table>
   </div>
  </xsl:when>
  <xsl:when test="@type = 'display'">
    <div align="center" class="mathdisplay"> <xsl:apply-templates /> </div>
  </xsl:when>
  <xsl:otherwise>
   <span class="math"> <xsl:apply-templates /> </span>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

The translation of `<footnote>Text</footnote>` is '(Text)', this is much more legible than a link to the text.

```
<xsl:template match='footnote'>
  <xsl:text>(</xsl:text><xsl:apply-templates /><xsl:text>)</xsl:text>
</xsl:template>
```

Because of the code above, a `<p>` in a footnote cannot be transformed into a paragraph.

```
<xsl:template match="footnote/p">
  <xsl:apply-templates />
</xsl:template>
```

The translation of `<p>` is a `<blockquote>` if the rend attribute is 'quoted'. Otherwise, it is a `<p>`. If rend is 'center' or 'centered', we add align = 'center'. If noindent is given[9], we add class = 'notaparagraph'.

```
<xsl:template match="p">
  <xsl:choose>
    <xsl:when test="@rend='quoted'">
        <blockquote> <xsl:apply-templates /> </blockquote>
    </xsl:when>
    <xsl:otherwise>
      <p>
        <xsl:choose>
          <xsl:when test="@rend='centered' or @rend='center'">
             <xsl:attribute name="align">center</xsl:attribute>
          </xsl:when>
          <xsl:when test="@noindent">
            <xsl:attribute name="class">notaparagraph</xsl:attribute>
```

[9] We should test the value.

```
          </xsl:when>
        </xsl:choose>
        <xsl:apply-templates />
      </p>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This is a bit strange. Currently[10] Tralics does not produce `<anchor>` elements. Note: the `<a>` element as shown here is empty, in reality, it contains a space. Why?

```
<xsl:template name="use-id">
  <xsl:if test="@id"> <a name="{@id}" />  </xsl:if>
  <xsl:if test="anchor/@id"><a name="{anchor/@id}" /> </xsl:if>
</xsl:template>
```

Translation of `<table>`. We call some template. The result will be in a `<div>`.

```
<xsl:template match="table">
 <div class="notinline" align='center' style="margin-top:20px">
    <xsl:call-template name="use-id" />
    <xsl:call-template name='generate-table' />
 </div>
</xsl:template>
```

If the table is inline, there is no reference to it (we can omit the id), and we do not produce a `<div>`.

```
<xsl:template match="table[@rend='inline']">
  <xsl:call-template name='generate-table' />
</xsl:template>
```

This is called when the table comes from an object. Note that the 'mode' is useless in the 'xsl:call-template'.

```
<xsl:template match="table" mode="object">
  <xsl:call-template name="use-id" />
  <xsl:call-template name='generate-table' mode="object" />
</xsl:template>
```

Translation of `<table>`. The result is a table, that contains the translation of the `<caption>`, followed by some `<tr>` elements. Each such elements is the translation of a `<tr>`, it is formed of the translation of all cells, the `<td>` and `<th>` sub-elements. The style attribute of each cell is copied, as well as the style of the row[11], in the case where the cell is empty, a height of three pixels is added. If an attribute rows or cols is present, it is transformed into rowspan or colspan. If a child of a cell is a `<ressource>`, the align attribute is set to center[12].

```
<xsl:template name="generate-table">
 <table>
  <xsl:apply-templates select="caption" />
  <xsl:for-each select="tr">
   <tr>
    <xsl:for-each select="td|th">
     <xsl:element name="{name()}">
      <xsl:attribute name="style">
        <xsl:if test="normalize-space(.) = ''">height:3px;</xsl:if>
        <xsl:value-of select="@style" />
        <xsl:value-of select="../@style" />
```

[10] Anchors were added in version 2.8.5, in order to make `\index` work.

[11] Question: what says the DTD about this style attribute?

[12] What if the initial XML document specifies a different horizontal alignment, something hidden in the style attribute?

```
      </xsl:attribute>
      <xsl:if test="@cols">
        <xsl:attribute name="colspan"><xsl:value-of select="@cols" /></xsl:attribute>
      </xsl:if>
      <xsl:if test="@rows">
        <xsl:attribute name="rowspan"><xsl:value-of select="@rows" /></xsl:attribute>
      </xsl:if>
      <xsl:if test="ressource">
         <xsl:attribute name="align">center</xsl:attribute>
      </xsl:if>
      <xsl:apply-templates />
     </xsl:element>
    </xsl:for-each>
   </tr>
  </xsl:for-each>
 </table>
</xsl:template>
```

A style attribute is always copied.

```
<xsl:template match="@style">
  <xsl:attribute name="style"><xsl:value-of select="." /></xsl:attribute>
</xsl:template>
```

A align attribute is converted to a style attribute.[13]

```
<xsl:template match="@align">
 <xsl:attribute name="style">text-align:<xsl-value-of select="." /></xsl:attribute>
</xsl:template>
```

The translation of a `<caption>` in a table is a `<caption>` element. It contains the table number (it should be the same as the one given by `calculateTableNumber`).

```
<xsl:template match="table/caption" >
  <caption align="bottom">
    <strong>Table
      <xsl:number count="table[not(ancestor::object)]" level="any" />
        <xsl:text>. </xsl:text>
    </strong>
    <xsl:apply-templates />
  </caption>
</xsl:template>
```

The translation of a `<caption>` in a table in a figure. The result is a simple `<caption>`. There should be no reference to it.

```
<xsl:template match="object//table/caption" priority="1">
   <caption> <xsl:apply-templates /> </caption>
</xsl:template>
```

Translation of `<img>`. We copy the attributes width, height, align, border, alt, and src. Note: the `<img>` element is not defined by the DTD. Thus, the author of the document should not create them; on the other hand, there is a post-processor that replaces some math formulas by images that match this usage. In the case where `$xyleme` is set and there is an attribute xylemeAttach, the src attribute is computed differently.

```
<xsl:template match="img">
 <img>
   <xsl:if test="@width">
     <xsl:attribute name="width"><xsl:value-of select="@width" /></xsl:attribute>
```

[13] What happens if both style and align are given? who wins? In fact, the question has no sense, because Tralics never emits a 'align', so that this template shoud never be called.

```
1650     </xsl:if>
1651     <xsl:if test="@height">
1652       <xsl:attribute name="height"><xsl:value-of select="@height" /></xsl:attribute>
1653     </xsl:if>
1654     <xsl:attribute name="align"><xsl:value-of select="@align" /></xsl:attribute>
1655     <xsl:attribute name="border"><xsl:value-of select="@border" /></xsl:attribute>
1656     <xsl:attribute name="alt"><xsl:value-of select="@alt" /></xsl:attribute>
1657     <xsl:attribute name="src">
1658       <xsl:choose>
1659         <xsl:when test="($xyleme='1') and @xylemeAttach">
1660           <xsl:value-of select="$imgpath" />
1661           <xsl:apply-templates select="@xylemeAttach" />
1662         </xsl:when>
1663         <xsl:otherwise>
1664           <xsl:value-of select="@src" />
1665         </xsl:otherwise>
1666       </xsl:choose>
1667     </xsl:attribute>
1668   </img>
1669 </xsl:template>
```

Given a string `$str`, of the form 'foo.bar.gee', this procedure returns what follows the last dot, namely 'gee'. It is a recursive procedure.

```
1670 <xsl:template name="get-extention" >
1671   <xsl:param name="str" />
1672   <xsl:choose>
1673     <xsl:when test="substring-after($str,'.')=''">
1674      <xsl:value-of select="$str" />
1675     </xsl:when>
1676     <xsl:otherwise>
1677       <xsl:call-template name="get-extention">
1678         <xsl:with-param name="str" select="substring-after($str,'.')" /> <!--$-->
1679       </xsl:call-template>
1680     </xsl:otherwise>
1681   </xsl:choose>
1682 </xsl:template>
```

Consider an image that has an attribute src with value 'toto.png' or aux with value 'titi.gif', and attribute xylemeAttach with value 'foo', this gives 'foo.png' or 'foo.gif' (we assume that only one extension will be found).

```
1683 <xsl:template match="@xylemeAttach">
1684    <xsl:value-of select="." />
1685    <xsl:text>.</xsl:text>
1686    <xsl:call-template name="get-extention">
1687       <xsl:with-param name="str" select="../@src|../@aux" />
1688    </xsl:call-template>
1689  </xsl:template>
```

Translation of `<ressource>`. The attribute media is 'WEB' by default. No rule applies otherwise.

```
1690 <xsl:template match="ressource[@media='WEB']">
1691   <xsl:call-template name='generate-graphics' />
1692   <xsl:apply-templates select="caption" />
1693 </xsl:template>
```

There is something strange here. If you specify both width and height, only the last attribute will be put in style. Perhaps, something like the code on lines 1607-1611 should be used instead. In the current version, a complicated procedure is used to get the name of the image.

```
<xsl:template name="generate-graphics">
  <img>
    <xsl:if test="@width">
     <xsl:attribute name="style">width:<xsl:value-of select="@width" /></xsl:attribute>
    </xsl:if>
    <xsl:if test="@height">
     <xsl:attribute name="style">height:<xsl:value-of select="@height" /></xsl:attribute>
    </xsl:if>
    <xsl:attribute name='alt'><xsl:value-of select="@xlink:href" /></xsl:attribute>
    <xsl:attribute name='src'>
      <xsl:value-of select="$imgpath" />
      <xsl:choose>
        <xsl:when test="($xyleme='1') and @xylemeAttach">
          <xsl:apply-templates select="@xylemeAttach" />
        </xsl:when>
        <xsl:when test="@png">
          <xsl:value-of select="$LeProjet"/>/<xsl:value-of select="@png" />
        </xsl:when>
        <xsl:when test="@gif">
          <xsl:value-of select="$LeProjet"/>/<xsl:value-of select="@gif" />
        </xsl:when>
        <xsl:when test="@jpg">
          <xsl:value-of select="$LeProjet"/>/<xsl:value-of select="@jpeg" />
        </xsl:when>
        <xsl:when test="@aux">
          <xsl:value-of select="$LeProjet"/>/<xsl:value-of select="@aux" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="$LeProjet"/><xsl:value-of select="@xlink:href"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:attribute>
  </img>
</xsl:template>
```

Translation of `<object>`. The result is a `<div>`, that contains a `<table>`. Each table in the object is converted into a single `<td>` in a `<tr>`.

```
<xsl:template match="object">
 <div align='center' style='margin-top:10px'>
    <a name="{./@id}"></a>
    <table title="{@title}" class="objectContainer">
       <xsl:call-template name="objectCaption" />
       <xsl:for-each select="table">
          <tr align="center">
            <td> <xsl:apply-templates select="." mode="object" /> </td>
          </tr>
       </xsl:for-each>
    </table>
  </div>
</xsl:template>
```

Translation of a `<caption>` of a `<object>`. It is like the caption of a table, but with name 'Figure' instead of 'Table'.

```
<xsl:template name="objectCaption">
  <caption align='bottom'>
      <strong>Figure
        <xsl:call-template name="calculateObjectNumber" />
```

```
1745          <xsl:text>. </xsl:text></strong>
1746        <xsl:apply-templates select="caption" />
1747    </caption>
1748  </xsl:template>
```

If the caption is in a resource in a cell, we output a `<br>`, and the value of the caption.

```
1749  <xsl:template match="td/ressource/caption" priority="1">
1750    <br />
1751    <xsl:apply-templates />
1752  </xsl:template>
1753
```

If the caption is in a ressource, we output the value of the caption.

```
1754  <xsl:template match="ressource/caption">
1755    <xsl:apply-templates />
1756  </xsl:template>
1757
```

If the caption matches none of the rules above, we output the value of the caption.

```
1758  <xsl:template match="caption">
1759    <xsl:apply-templates />
1760  </xsl:template>
```

These two elements exist so that we can transform them into TEX or LATEX in the Pdf result. In the HTML, it is better to use letters only.

```
1761  <xsl:template match="LaTeX">LaTeX</xsl:template>
1762  <xsl:template match="TeX">TeX</xsl:template>
```

This is a helper function. It adds a comma if the element is not the last.

```
1763  <xsl:template name="separateurED.objet">
1764    <xsl:if test="position() &lt; last()">, </xsl:if>
1765  </xsl:template>
```

This is a helper function. It adds a comma if the element is not the last, a dot otherwise.

```
1766  <xsl:template name="separateur.objet">
1767    <xsl:choose>
1768      <xsl:when test="position() &lt; last()">, </xsl:when>
1769      <xsl:otherwise>.</xsl:otherwise>
1770    </xsl:choose>
1771  </xsl:template>
```

6.1.5 References

The translation of `<ref>` is an `<a>`. The non trivial point is to compute the value (the reference is much easier). If the link does not start with a `#`, it is an external link, this will be handled later. Otherwise some variables are used. First, `$curid` is the target id (the link without the `#`), and `$curelement` is the target. In Tralics, this can be a section, a math formula, an image, a table, a bibliographic entry, a footnote, an item in a list[14]. In the Raweb, other elements might be referenced. We consider three variables: `$cursubsection`, `$curbiblio` and `$curidentification`, that contains ancestors of the target. These variables hold nodesets; their intersection is empty.

```
1772  <xsl:template match="ref">
1773    <xsl:choose>
1774      <xsl:when test="starts-with(@xlink:href, '#')">
1775        <xsl:variable name="curid"
1776            select="substring-after(@xlink:href, '#')" />
```

[14] Or an anchor, since version 2.8.5

```
1777          <xsl:variable name="curelement"
1778            select="/raweb//*[@id=$curid]" />
1779          <xsl:variable name="cursubsection"
1780            select="$curelement/ancestor-or-self::subsection" />
1781          <xsl:variable name="curbiblio"
1782            select="$curelement/ancestor-or-self::biblio" />
1783          <xsl:variable name="curidentification"
1784            select="$curelement/ancestor-or-self::identification" />
```

Case one. `$curidentification` is not empty. This assumes that only one target exists in the page. A safer solution would be to replace `$curid` by the id of the `<team>` element. Case two, the target is in a subsection. We consider the first subsection (in document order), its id gives the name of the HTML page, it suffices to add the local part. The content of the link is computed elsewhere. In the third case, the target is in the bibliography. Otherwise, let's hope that the target is a section, we use the first subsection of it.

```
1785      <xsl:choose>
1786       <xsl:when test="$curidentification">
1787         <a>
1788           <xsl:attribute name='href'>
1789             <xsl:call-template name="formaturl">
1790               <xsl:with-param name="base" select="$curid"/>
1791             </xsl:call-template>
1792           </xsl:attribute>
1793         </a>
1794       </xsl:when>
1795       <xsl:when test="$curbiblio">
1796         <xsl:apply-templates select="." mode="ref2biblio" >
1797           <xsl:with-param name="curelement" select="$curelement" />
1798         </xsl:apply-templates>
1799       </xsl:when>
1800       <xsl:when test="$cursubsection">
1801         <a href="./{$cursubsection[1]/@id}.html{@xlink:href}">
1802              <xsl:apply-templates mode="xref" select="$curelement" />
1803         </a>
1804       </xsl:when>
1805       <xsl:when test="$cursubsection"> <!-- vers la meme section -->
1806         <a>
1807           <xsl:attribute name='href'>
1808             <xsl:call-template name="formaturl">
1809               <xsl:with-param name="base" select="$cursubsection[1]/@id" />
1810             </xsl:call-template>
1811             <xsl:value-of select="@xlink:href" />
1812           </xsl:attribute>
1813           <xsl:attribute name="title">
1814             <xsl:value-of select="$cursubsection/bodyTitle" />
1815           </xsl:attribute>
1816           <xsl:apply-templates mode="xref" select="$curelement" />
1817         </a>
1818       </xsl:when>
1819       <xsl:otherwise>
1820        <a>
1821         <xsl:attribute name="title">
1822           <xsl:value-of select="$curelement/subsection[1]/subsection" />
1823         </xsl:attribute>
1824         <xsl:attribute name="href">
1825           <xsl:call-template name="formaturl">
```

```
1826              <xsl:with-param name="base" select="$curelement/subsection[1]/@id" />
1827            </xsl:call-template>
1828          </xsl:attribute>
1829          <xsl:apply-templates mode="xref" select="$curelement" />
1830        </a>
1831       </xsl:otherwise>
1832      </xsl:choose>
1833     </xsl:when>
```

In the case of an external reference, things are easier. The value of the `<ref>` element is assumed non-empty.[15]

```
1834    <xsl:otherwise>
1835     <a>
1836       <xsl:attribute name="href"><xsl:value-of select="@xlink:href" /></xsl:attribute>
1837       <xsl:call-template name="targetAttrib.alt" />
1838       <xsl:apply-templates />
1839     </a>
1840    </xsl:otherwise>
1841   </xsl:choose>
1842 </xsl:template>
```

For every element, like `<formula>`, that can be the target of a `<ref>`, we must compute its number. This is the value seen on the screen. In the case of a formula, this is easy, we just count all formulas with a number (i.e., with an id). Note: this behavior is different from what happens in the Pdf case. For this reason it would be safer to replace the code on line 1637 by a call to this template.[16]

```
1843 <xsl:template match="formula" mode="xref">
1844   <xsl:number level="any" count="formula[@id]" />
1845 </xsl:template>
```

In the case of a `<table>`, computation is non trivial. We use an intermediate command.

```
1846 <xsl:template match='table' mode="xref">
1847   <xsl:call-template name="calculateTableNumber" />
1848 </xsl:template>
```

This is how we count them: if the `<table>` is in an `<object>` it is not counted.

```
1849 <xsl:template name="calculateTableNumber">
1850    <xsl:number count="table[not(ancestor::object)]" level="any" />
1851 </xsl:template>
```

The same idea is used for `<object>` and `<ressource>`: we use a command.

```
1852 <xsl:template match="ressource" mode="xref">
1853   <xsl:call-template name="calculateRessourceNumber" />
1854 </xsl:template>
1855 <xsl:template match="object" mode="xref">
1856     <xsl:call-template name="calculateObjectNumber" />
1857 </xsl:template>
```

In order to understand this, remember that an `<object>` can contain a `<table>` that can contain some `<ressource>`. If an object contains more than one image, there is currently no way to refer to a specific image (Tralics allows you to reference the object, not the images). Hence, the number of a ressource is the number of its parent object.

```
1858 <xsl:template name="calculateObjectNumber">
1859   <xsl:number count="object" level="any" />
1860 </xsl:template>
```

[15] This sets the target attribute of the anchor. This behaviour is sometimes annoying.

[16] If you look at the code line 487, you can see that conversion from one DTD to the other does not add ids to formulas. Hence, if Tralics adds no useless ids, you will see the same equation number in the Pdf and the HTML.

```
1861
1862 <xsl:template name="calculateRessourceNumber">
1863   <xsl:apply-templates select="ancestor::object[1]" mode="xref" />
1864 </xsl:template>
```

This returns the number of the section in which the element is. Note that 'fondements' are numbered 3, even if 'presentation' is missing[17]. If you don't like this, then you should fill all sections[18].

```
1865 <xsl:template name="sec.num">
1866     <xsl:choose>
1867       <xsl:when test="ancestor-or-self::*[self::identification]"> 0</xsl:when>
1868       <xsl:when test="ancestor-or-self::*[self::presentation]">   1</xsl:when>
1869       <xsl:when test="ancestor-or-self::*[self::fondements]"> 2</xsl:when>
1870       <xsl:when test="ancestor-or-self::*[self::domaine]">  3</xsl:when>
1871       <xsl:when test="ancestor-or-self::*[self::logiciels]">  4</xsl:when>
1872       <xsl:when test="ancestor-or-self::*[self::resultats]">  5</xsl:when>
1873       <xsl:when test="ancestor-or-self::*[self::contrats]">   6</xsl:when>
1874       <xsl:when test="ancestor-or-self::*[self::international]"> 7</xsl:when>
1875       <xsl:when test="ancestor-or-self::*[self::diffusion]">  8</xsl:when>
1876       <xsl:when test="ancestor-or-self::*[self::biblio]">  9</xsl:when>
1877       <xsl:otherwise>*-</xsl:otherwise>
1878     </xsl:choose>
1879  </xsl:template>
```

The number associated to each section is given by the procedure above.

```
1880 <xsl:template match="identification|presentation|fondements|domaine|
1881       logiciels|resultats|contrats|international|diffusion|biblio"
1882     mode="xref">
1883     <xsl:call-template name="sec.num" />
1884 </xsl:template>
```

This computes the number of a subsection.

```
1885 <xsl:template match="subsection" mode="xref">
1886   <xsl:call-template name="calculateNumbersubsection" />
1887 </xsl:template>
```

We count the number of subsections, this is preceded by the section number. We removed here the variable `$numbersuffix`, replacing it by a simple `<xsl:text>` element.

```
1888 <xsl:template name="calculateNumbersubsection">
1889    <xsl:call-template name="sec.num" />
1890    <xsl:text>.</xsl:text>
1891    <xsl:number level="multiple" grouping-separator="."
1892      from="raweb" format="1.1" count="subsection" />
1893 </xsl:template>
```

In the case of a title or an anchor, we call a template (originally, there were two templates).

```
1894 <xsl:template match="bodyTitle|anchor" mode="xref">
1895     <xsl:call-template name="calculateNumber" />
1896 </xsl:template>
```

The number we compute in the generic case is the number of its subsection.

```
1897 <xsl:template name="calculateNumber">
1898    <xsl:call-template name="sec.num" />
1899    <xsl:text>.</xsl:text>
```

[17] The first section is numbered zero for no obvious reason, so that 'fondements' is numbered 3 in the Pdf filem and 2 in the HTML file.

[18] Why is there a space before the number? The LaTeX equivalent would be wrong. It seems that my brower discards these spaces.

```
1900     <xsl:number level="multiple" from="raweb" format="1.1" count="subsection" />
1901   </xsl:template>
```

That's the end of the file.

```
1902   </xsl:stylesheet>
```

```
1903   <xsl:template match="@id">
1904     <xsl:attribute name="id"><xsl:value-of select="." /></xsl:attribute>
1905   </xsl:template>
```

6.2 Dealing with topics

The following three templates do nothing in static mode, but add a class attribute to the current element. For simplicity, the calls to these templates are often omitted.

```
1906   <xsl:template name="isSamePage" />
1907   <xsl:template name="isPageTeam" />
1908   <xsl:template name="isPageBiblio"/>
```

There are two views: with and without topics. The difference resides in how pages are linked together, hence has an influence on the table of contents. We show here how to compute the next page.

```
1909   <xsl:template name="suivant">
1910     <xsl:choose>
1911       <xsl:when test="$isTopic='1'">
1912         <xsl:call-template name="suivant_avec_topic" />
1913       </xsl:when>
1914       <xsl:otherwise>
1915         <xsl:call-template name="suivant_sans_topic" />
1916       </xsl:otherwise>
1917     </xsl:choose>
1918   </xsl:template>
```

There are additional templates that follow the same scheme. The template `precedent` returns the previous page, `tdm` and `table.matieres` create the table of contents, `toclink` produces a link to the TOC.

```
1919   <xsl:template name="precedent">
1920     choose one of "precedent_avec_topic""precedent_sans_topic"
1921   </xsl:template>
1922
1923   <xsl:template name="noframe_p">
1924     choose one of "noframe_p_Topic" "noframe_p_sansTopic"
1925   </xsl:template>
1926
1927   <xsl:template name="tdm">
1928     choose one of "tdm_Topic" "tdm_sansTopic"
1929   </xsl:template>
1930
1931   <xsl:template name="table.matieres">
1932     choose one of "table.matieres_Topic" "table.matieres_sansTopic"
1933   </xsl:template>
1934
1935   <xsl:template name="toclink">
1936     choose one of "toclink_Topic" "toclink_sansTopic"
1937   </xsl:template>
```

In the static version, these produce nothing (and more than often we shall omit the call).

```
<xsl:template name="noframe_p_sansTopic" />
<xsl:template name="noframe_p_Topic/">
```

Now comes the delicate part: what is the previous or next page? Finding the previous module is rather easy: if there is a previous module in the section, we chose it; if there is a previous section we chose the last module of it; otherwise it is the front page.

```
<xsl:template name="precedent_sans_topic">
  <xsl:choose>
    <xsl:when test="preceding-sibling::subsection">
      <xsl:value-of select="preceding-sibling::subsection[1]/@id" />
    </xsl:when>
    <xsl:when test="../preceding-sibling::*[1]/subsection">
      <xsl:value-of select="../preceding-sibling::*[1]/subsection[position()=last()]/@id"/>
    </xsl:when>
    <xsl:otherwise>
       <xsl:value-of select="/raweb/identification/team/@id" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Finding the next module is easy too: if there is a next module in the section, we chose it; if there is a next section we chose the first module of it; otherwise it is the bibliography.

```
<xsl:template name="suivant_sans_topic">
  <xsl:choose>
    <xsl:when test="following-sibling::subsection">
      <xsl:value-of select="following-sibling::subsection[1]/@id" />
    </xsl:when>
    <xsl:when test="../following-sibling::*[1]/subsection">
      <xsl:value-of select="../following-sibling::*[1]/subsection[1]/@id" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>bibliography</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

The situation is a bit more complicated in the case of topics. Assume that S3M4 is module 4 in section 3 and T2S3M4 is module 4 in section 3 with topic 2. Topics are allowed only in sections 3 to 6. The ordering is the following: S1M1, S1M2, S2M1, S2M2, T1S3M1, T1S3M2, T1S4M1, T1S4M2, T1S5M1, T1S5M2, T1S6M1, T1S6M2, T2S3M1, T2S3M2, T2S4M1, T2S4M2, T2S5M1, T2S5M2, T2S6M1, T2S6M2, S7M1, S7M2, S8M1, S8M2, S9M1, S9M2, S10M1, S10M2. We assume that a module has a topic attribute if and only if it is in a section with topics. This attribute has a symbolic value, say A, B, C. The start of the document contains a list of `<topic>` elements, say `<tA>`, `<tB>`, `<tB>`, with id A, B and C, that define the order (A comes before B if `<tA>` is before `<tB>`).

This finds the previous module. There are two cases to consider: it depends on whether we are in the topic part, or non-topic part. We use two variables: `$var` is the current topic id and `$precedentTopic` is the previous topic (or is empty if there is no previous topic).

```
<xsl:template name="precedent_avec_topic">
  <xsl:variable name="var" select="./@topic" />
  <xsl:variable name="precedentTopic"
       select="/raweb/topic[@id=$var]/preceding-sibling::topic[1]" />
  <xsl:choose>
```

First assume that the module has a topic. The easy case is when there is a preceding sibling with the same topic. We chose the first one. Otherwise, we consider all modules that have as topic

the previous topic; we select the last one; if this fails, we select the last module of the presentation section.

```
    <xsl:when test="./@topic">
      <xsl:choose>
        <xsl:when test="preceding::subsection[@topic=$var]">
          <xsl:value-of select="preceding::subsection[@topic=$var][1]/@id" />
        </xsl:when>
        <xsl:when test="/raweb/descendant::subsection[@topic=($precedentTopic/@id)]">
          <xsl:variable name="precedingTopicId" select="$precedentTopic/@id" />
          <xsl:variable name="lastSubsection"
             select="/raweb/descendant::subsection[@topic=($precedentTopic/@id)][last()]" />
          <xsl:value-of select="$lastSubsection/@id" />
        </xsl:when>
        <xsl:otherwise>
           <xsl:value-of select="/raweb/presentation/subsection[last()]/@id" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
```

We consider now the case where the module has no topic. In the case where there is a preceding sibling in the same section without topic, we chose the first one. In the case where the section is the presentation, our module is located before the modules with topics, thus the preceding page is that with the composition. If there is a preceding module without topic, we use it. Otherwise, if there are topics, and if there is a module with a topic, we chose the last module of the last topic. Otherwise, we chose the page with the team.

```
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="parent::presentation and position()=1">
          <xsl:value-of select="/raweb/identification/team/@id" />
        </xsl:when>
        <xsl:when test="preceding-sibling::subsection[not(./@topic)]">
          <xsl:value-of select="preceding-sibling::subsection[not(./@topic)][1]/@id" />
        </xsl:when>
        <xsl:when test="../preceding-sibling::*[1]/subsection[not(./@topic)]">
          <xsl:value-of select="../preceding-sibling::*[1]/
                      subsection[not(./@topic)][last()]/@id" />
        </xsl:when>
        <xsl:when test="/raweb/topic and /raweb/descendant::subsection[@topic]">
          <xsl:variable name="lastTopic" select="//topic[last()]/@id" />
          <xsl:variable name="lastSubsection"
             select="/raweb/descendant::subsection[@topic=$lastTopic][last()]" />
          <xsl:value-of select="$lastSubsection/@id" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="/raweb/identification/team/@id" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This finds the next module. The algorithm is the same as above (to be precise: if X precedes Y, then Y follows X).

```
<xsl:template name="suivant">
  <xsl:variable name="var" select="./@topic" />
  <xsl:variable name="followingTopic"
```

```
      select="/raweb/topic[@id=$var]/following-sibling::topic[1]" />
  <xsl:choose>
```

There are two cases to consider. First assume that the module has a topic. The easy case is when there is a following sibling with a topic. We chose the first one. The next case is when there is a module with the following topic. We chose the first one. Then we consider the case where the current topic is the last one, and there is a module in one of the main sections (not presentation) that has no topic; the first one is selected. Otherwise, the next page is the bibliography.

```
    <xsl:when test="./@topic">
      <xsl:choose>
        <xsl:when test="following::subsection[@topic=$var]">
          <xsl:value-of select="following::subsection[@topic=$var][1]/@id" />
        </xsl:when>
        <xsl:when test="/raweb/descendant::subsection[@topic=$followingTopic/@id]">
          <xsl:value-of select="/raweb/descendant::subsection
                                [@topic=$followingTopic/@id][1]/@id" />
        </xsl:when>
        <xsl:when test="not($followingTopic) and
             /raweb/*[name()!='presentation']/subsection[not(@topic)]">
          <xsl:value-of select="/raweb/*[name()!='presentation']/
                                subsection[not(@topic)][1]/@id" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>bibliography</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
```

We consider here the case where the module has no topic. The easy case is when the section contains a following module without topic. Then comes the case where we are in the presentation section, it is the last module, and there is a module with a topic: in this case we chose the first module with the first topic. Otherwise we consider the case where there is a following module without topic, we chose the first one. Otherwise, we select the bibliography.

```
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="following-sibling::subsection[not(@topic)]">
          <xsl:value-of select="following-sibling::subsection[not(@topic)][1]/@id" />
        </xsl:when>
        <xsl:when test="parent::presentation and (position()=last())
               and /raweb/descendant::subsection[@topic]">
          <xsl:variable name="firstTopicId" select="/raweb/topic[1]/@id" />
          <xsl:value-of select="/raweb/descendant::subsection
                                          [@topic=$firstTopicId][1]/@id" />
        </xsl:when>
        <xsl:when test="../following-sibling::*[1]/subsection[not(@topic)]">

          <xsl:value-of select="../following-sibling::*[1]/subsection
                                       [not(@topic)][1]/@id" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>bibliography</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This is strange. We removed all tests concerning the `$type` parameter (which is never provided). If the `$base` is 'foo', this constructs ./foo.html.

```
<xsl:template name="formaturl">
  <xsl:param name="base" />
  <xsl:param name="type" />
  ./<xsl:value-of select="$base" />.html
</xsl:template>
```

In the case where topics are present, there are two possible views: with topics or without; you can switch from one view to the other by clicking somewhere in the frame with the TOC.

This piece of code is used in the case where the document has topics, and we generate the TOC without topics; you will see a line containing 'View by sections' and 'View by topics'. These pieces of text have different colors, the second one is associated to an anchor (that has a style attribute that says: no decoration, the change of color should be enough).

Note that the target is '_parent', so that the loading the aoste_tf.html file (or the same with 'Topics') will remove this page and the sibling frame, replacing them by two pages (TOC and main page).

```
<xsl:template name="acces.topic">
    <div id="bouton">
      <div id="clair">View by sections</div>
      <div id="fonce">
        <a style="text-decoration:none" target="_parent">
          <xsl:call-template name="lienVersTopics" />
          <div id="bouton_tdm_click">
            <font color="#FFFFFF">View by topics</font>
          </div>
        </a>
      </div>
    </div>
    <br />
</xsl:template>
```

This constructs the name ../aoste_Topics/aoste_tf.html of the file with topics.

```
<xsl:template name="lienVersTopics">
  <xsl:attribute name="href" >
    <xsl:value-of select="concat('../',$LeProjet,'_Topics/',$LeProjet,'_tf.html')"/>
  </xsl:attribute>
</xsl:template>
```

This piece of code is used in the case where the document has topics, and we generate the TOC with topics; you will see the same text as above; but colors are different and the link points to ../aoste/aoste_tf.html.

```
<xsl:template name="acces.topic_Topic">
  <br />
  <div id="bouton">
    <div id="fonce_T">
      <a href="../{$LeProjet}/{$LeProjet}_tf.html"
       style="text-decoration:none" target="_parent">
         <div id="bouton_tdm_click">
           <font color="#FFFFFF">View by sections</font>
         </div>
       </a>
     </div>
     <div id="clair_T">View by topics</div>
  </div>
</xsl:template>
```

Our next object is the table of contents. There are four cases to consider. The main page contains the TOC defined by `table.matieres`, and there is a frame containting the TOC (defined by `tdm`), they exist in two versions, with or without topics. Let's start with the frame.

This creates the full table of contents, that is on the front page. We have three parts: composition, main TOC, bibliography.

```
<xsl:template name="table.matieres_sansTopic">
  <ul class="tdm_frame">
    <xsl:call-template name="jg.tdm.members"/>
    <xsl:call-template name="table.section_sansTopic"/>
    <xsl:call-template name="table.bibliography" />
  </ul>
</xsl:template>
```

The case of topics is a bit more complicated. Instead of `table.section` that shows everything but the composition and the bibliography, we have three parts: the presentations, all modules that have a topics, then modules without topic. The second part is preceded by 'View by topics' and delimited by colored rules.

```
<xsl:template name="table.matieres_Topic">
  <div class="non-topic">
    <ul class="tdm_window">
      <xsl:call-template name="jg.tdm.members"/>
      <li> <xsl:call-template name="presentation_tdm_entry_Topic" /> </li>
    </ul>
  </div>
  <div class="topicIdent">
    <br />
    <hr class="topic_color" />
    View by topics
  </div>
  <xsl:call-template name="table.topic_Topic">
      <xsl:with-param name="class">tdm_window</xsl:with-param>
  </xsl:call-template>
  <hr class="topic_color" />
  <ul class="tdm_window">
   <xsl:call-template name="table.section_Topic" />
   <li>
     <xsl:call-template name="table.bibliography" />
   </li>
  </ul>
</xsl:template>
```

This is now the TOC frame. The structure is easy: we have, from top to bottom, the logo, the name of the team, the composition of the team, the presentation, the sections that might have a topic, and the bibliography.

```
<xsl:template name="tdm_sansTopic">
  <div>
    <xsl:call-template name="jg.tdm-common"/>
    <xsl:call-template name="jg.tdm.team.sans"/>
    <xsl:call-template name="presentation_tdm_entry_sansTopic" />
    <xsl:call-template name="table.tdm_sansTopic" />
    <xsl:call-template name="table.bibliography" />
  </div>
</xsl:template>
```

In the case of topics, the layout of the page with the frame is the same as the layout of the title page, but there are no rules.

```
<xsl:template name="tdm_Topic">
  <div>
    <xsl:call-template name="jg.tdm-common"/>
    <xsl:call-template name="jg.tdm.team.topic"/>
    <br />
    <xsl:call-template name="presentation_tdm_entry_Topic" />
    <xsl:call-template name="acces.topic_Topic" />
    <xsl:call-template name="table.topic_Topic">
      <xsl:with-param name="class">tdm_frame</xsl:with-param>
    </xsl:call-template>
    <xsl:call-template name="tdm.section_Topic" />
    <xsl:call-template name="table.bibliography" />
  </div>
</xsl:template>
```

The link to the composition of the team in the TOC is constructed via this template.

```
<xsl:template name="jg.team.link">
  <a href="{/raweb/identification/team/@id}.html">
     <xsl:call-template name="targetAttrib.main" />
     Members
  </a>
</xsl:template>
```

In fact, there are three variants of the previous template: in one case, the team is an item in a list, otherwise, it is in a `<div>` with class 'non-topic' or 'TdmEntry'.

```
<xsl:template name="jg.tdm.team.topic"/>
  <div class="non-topic">
    <xsl.call-template name="jg.team.link"/>
  </div>
</xsl:template>

<xsl:template name="jg.tdm.team.sans"/>
  <div class="TdmEntry">
    <xsl.call-template name="jg.team.link"/>
  </div>
</xsl:template>

<xsl:template name="jg.tdm.members">
  <li>
    <xsl.call-template name="jg.team.link"/>
  </li>
</xsl:template>
```

The frame TOC is in a `<div>` element that has a class attribute formed of 'tdmdiv' and optionally 'noframe'. It contains the Inria logo with a link to Inria's home page, and the name of the team.

```
<xsl:template name="jg.tdm-common"/>
  <xsl:attribute name="class">tdmdiv
    <xsl:if test="$noframe='1'">noframe</xsl:if>
  </xsl:attribute>
  <div class="logo">
    <a href="http://www.inria.fr">
      <xsl:call-template name="targetAttrib.main" />
      <img  border="0" src="../icons/logo-inria-.gif" alt="Inria" />
    </a>
  </div>
  <br />
```

```
  <div class="entete">
    <xsl:if test="/raweb/identification[@isproject]='true'">
      <xsl:text>Project </xsl:text>
    </xsl:if>
    <xsl:text>Team </xsl:text>
    <xsl:value-of select="/raweb/identification/shortname" />
  </div>
  <hr />
</xsl:template>
```

The bibliography is divided into three major parts, and since 2006, these appear in the TOC, unless they are empty.

```
<xsl:template name="table.bibliography">
  <div class="TdmEntry">Bibliography</div>
  <div class="TdmEntry">
    <ul>
      <xsl:if test="//biblio/biblStruct/note[@type='from']='refer'">
        <li>
          <a id="tdmbibentmajor" href="bibliography.html#Major" >
            <xsl:call-template name="targetAttrib.main" />
            Major publications
          </a>
        </li>
      </xsl:if>
      <xsl:if test="//biblio/biblStruct/note[@type='from']='year'">
        <li>
          <a id="tdmbibentyear" href="bibliography.html#year">
            <xsl:call-template name="targetAttrib.alt" />
            Year Publications
          </a>
        </li>
      </xsl:if>
      <xsl:if test="//biblio/biblStruct/note[@type='from']='foot'">
        <li>
           <a id="tdmbibentfoot" href="bibliography.html#Bibliography">
             <xsl:call-template name="targetAttrib.main" />
             References in notes
           </a>
        </li>
      </xsl:if>
    </ul>
  </div>
</xsl:template>
```

In order to make the code easier to understand we have introduced the following templates. It outputs the title, maybe in a `<a>` or a `<li>`. The void template is used when the title of a module is empty; in this case the title of the section containing the module is used instead.

```
<xsl:template name="jg.tdm.basic">
   <xsl:call-template name="item_title_or_not_title" />
</xsl:template>

<xsl:template name="jg.tdm.a">
  <a> <xsl:call-template name="item_title_or_not_title" /></a>
</xsl:template>

<xsl:template name="jg.tdm.a.li">
   <li><a><xsl:call-template name="item_title_or_not_title" /></a></li>
```

```
</xsl:template>

<xsl:template name="jg.tdm.void">
  <li>
    <a href="{@id}.html">
      <xsl:call-template name="targetAttrib.main" />
      <xsl:value-of select="/raweb/presentation/bodyTitle" />
    </a>
  </li>
</xsl:template>
```

Same as above, but `item_title_or_not_title` is not used. Instead, if the value is empty, some other quantity is used (shown here in `$varTitle`). Attributes of the anchor are computed by `table.tdm.xref`, and the title is output in 'tocsection' mode.

```
<xsl:template name="jg.tdm.sans.ali">
  <li>
    <a>
      <xsl:call-template name="targetAttrib.main" />
      <xsl:call-template name="table.tdm.xref" />
      <xsl:apply-templates mode="tocsection" select="bodyTitle" />
    </a>
  </li>
</xsl:template>

<xsl:template name="jg.tdm.sans.ali.default">
  <li>
    <a>
      <xsl:call-template name="targetAttrib.main" />
      <xsl:call-template name="table.tdm.xref" />
      <xsl:apply-templates mode="tocsection" select="{$varTitle}" />
    </a>
  </li>
</xsl:template>

<xsl:template name="jg.tdm.avec.ali">
  <li>
    <a name="{@id}" href="{@id}.html#{@id}">
      <xsl:call-template name="targetAttrib.main" />
      <xsl:apply-templates mode="tocsection" select="bodyTitle" />
    </a>
  </li>
</xsl:template>
```

This is not valid XSLT code, but you get the idea. In the case where the presentation section has a single module, its title is very often the same as the section, and we show it only once. The code is the same, with and without topics.

```
<xsl:template name="jg.tdm.special">
  <xsl:when test="count(/raweb/presentation/subsection)=1
         and /raweb/presentation/bodyTitle=
             /raweb/presentation/subsection/bodyTitle">
    <xsl:for-each select="/raweb/presentation/subsection">
      <xsl:call-template name="jg.tdm.a" />
    </xsl:for-each>
  </xsl:when>
</xsl:template>
```

This is the code for the presentation in the general case with topics. For each module, we print the section title and the module title.

```
<xsl:template name="jg.tdm.with-topics">
  <ul>
     <xsl:for-each select="/raweb/presentation/subsection">
       <xsl:value-of select="/raweb/presentation/bodyTitle" />
       <xsl:call-template name="jg.tdm.a.li" />
     </xsl:for-each>
  </ul>
</xsl:template>
```

The same without topics. We print the section title (if the module has a fake name) or the module title (otherwise).

```
<xsl:template name="jg.tdm.without-topics">
  <ul>
   <xsl:for-each select="/raweb/presentation/subsection">
    <xsl:choose>
      <xsl:when test="bodyTitle='(Sans Titre)'">
        <xsl:call-template name="jg.tdm.void" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:call-template name="jg.tdm.a.li" />
      </xsl:otherwise>
    </xsl:choose>
   </xsl:for-each>
  </ul>
</xsl:template>
```

A non-obvious point is: where is the 'presentention' section put in the TOC? In the case with topics, the following template is called, so that, if no module of the section has a topic, it will be output here, otherwise by `table.topic_Topic`. In the case without topics, it will be `presentation_tdm_entry_sansTopic` (framed case) `table.section_sansTopic` (otherwise). Full code of the presentation, case with topics.

```
<xsl:template name="presentation_tdm_entry_Topic">
  <xsl:if test="not(/raweb/presentation/subsection/@topic)">
    <div class="non-topic">
      <xsl:choose>
        <xsl:call-template name="jg.tdm.special" />
        <xsl:otherwise>
          <xsl:call-template name="jg.tdm.with-topics" />
        </xsl:otherwise>
      </xsl:choose>
    </div>
  </xsl:if>
</xsl:template>
```

Full code of the presentation, case without topics.

```
<xsl:template name="presentation_tdm_entry_sansTopic">
  <div class="TdmEntry">
    <xsl:choose>
      <xsl:call-template name="jg.tdm.special" />
      <xsl:otherwise>
        <xsl:value-of select="bodyTitle" />
        <xsl:call-template name="jg.tdm.without-topics" />
      </xsl:otherwise>
     </xsl:choose>
```

```
  </div>
</xsl:template>
```

Let's consider the following sections: `<fondements>`, `<domaine>`, `<logiciels>`, `<resultats>`, `<contrats>`, `<international>`, and `<diffusion>`, in the case of a frame, without topics. We start with sections that have a module with a topic, and then the other ones; note the `<div>` element; the CSS says that this element has a different color; it is preceded by a link (created by `access.topic`) that switches to the alternate view.

```
<xsl:template name="table.tdm_sansTopic">
  <xsl:if test="/raweb/child::*[self::fondements or ...
              or self::diffusion]//subsection[@topic]">
    <xsl:call-template name="acces.topic"/>
    <div class="topic">
      <xsl:call-template name="table.maybetopic"/>
    </div>
  </xsl:if>
  <xsl:call-template name="table.maybe-not-topic"/>
</xsl:template>
```

This is a loop over all sections, only those that have a module with a topic are chosen.

```
<xsl:template name="table.maybetopic"/>
  <xsl:for-each select="/raweb/child::*[self::fondements or ... or self::diffusion]">
     <xsl:if test=".//subsection[@topic]">
        <xsl:call-template name="table.tdm.entry_sansTopic" />
     </xsl:if>
  </xsl:for-each>
</xsl:template>
```

This is a loop over all sections, sections not selected above are selected here.

```
<xsl:template name="table.maybe-not-topic"/>
  <xsl:for-each select="/raweb/child::*[self::fondements or ... or self::diffusion]">
     <xsl:if test="not(.//subsection[@topic])">
       <xsl:call-template name="table.tdm.entry_sansTopic" />
     </xsl:if>
  </xsl:for-each>
</xsl:template>
```

For each section, we output its name, and the modules in it. We simplified the code, by adding a variable, instead of looking at some strange location, in the case when the title of the module is empty.

```
<xsl:template name="table.tdm.entry_sansTopic">
  <xsl:variable name="varTitle" select="bodyTitle" />
  <div class="TdmEntry">
    <xsl:value-of select="../bodyTitle" />
    <ul>
      <xsl:for-each select="subsection">
       <xsl:choose>
         <xsl:when test="./bodyTitle">
            <xsl:call-template name="jg.tdm.sans.ali" />
         </xsl:when>
         <xsl:otherwise>
            <xsl:call-template name="jg.tdm.sans.ali" />
         </xsl:otherwise>
       </xsl:choose>
      </xsl:for-each>
    </ul>
```

```
  </div>
</xsl:template>
```

Assume that the element has id 'uid4'. We shall add (to the anchor constructed below), two attributes, href with value 'uid4.html?B#tdmBuid4ent' and id with value 'tdmBuid4ent', where B has to be replaced by some newline character and white space; this seems a bit obscure (the file uid4.html does not contain the string 'uid4ent'). In the case `table.section`, the link will be 'uid4.html#uid', and this is a valid link.

```
<xsl:template name="table.tdm.xref">
   <xsl:attribute name="href">
     <xsl:value-of select="@id" />
     .html?
     <xsl:call-template name="noframe_p" />
     #tdm
     <xsl:value-of select="@id" />
     ent
   </xsl:attribute>
   <xsl:attribute name="id">
     tdm
     <xsl:value-of select="@id" />
     ent
   </xsl:attribute>
</xsl:template>
```

This outputs the name of the section, and all modules in it. There is a variant that uses `jg.tdm.avec.li` instead of `jg.tdm.a.li`, and another one that selects only modules having the same topic (which is in the variable `$num-topic`)

```
<xsl:template name="jg.tdm.modules">
  <xsl:value-of select="./bodyTitle" />
  <ul>
    <xsl:for-each select="subsection">
      <xsl:call-template name="jg.tdm.a.li" />
    </xsl:for-each>
  </ul>
</xsl:template>

<xsl:template name="jg.tdm.modules.avec">
  <!-- Same, but use jg.tdm.avec.ali -->
</xsl:template>

<xsl:template name="jg.tdm.modules.topic">
  <!-- Same, but select= "subsection[@topic=$num-topic]"-->
</xsl:template>
```

This inserts, in topics mode, all sections that have no topics (presentation is excluded here).

```
<xsl:template name="tdm.section_Topic">
  <xsl:for-each select="/raweb/child::*[ self::fondements or ...or self::diffusion]">
    <xsl:if test="not(child::subsection/@topic)">
      <xsl:call-template name="jg.tdm.modules" />
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

Same code as above, with a `<li>`. This is used in these case without frame (the previous code was used in the case of frames).

```
<xsl:template name="table.section_Topic">
  <xsl:for-each select="/raweb/child::*[ self::fondements or ...or self::diffusion]">
```

```
2408      <xsl:if test="not(child::subsection/@topic)">
2409        <li>
2410          <xsl:call-template name="jg.tdm.modules" />
2411        </li>
2412      </xsl:if>
2413    </xsl:for-each>
2414  </xsl:template>
```

This considers in the case without frames, without topics, all sections. If the section has a single module, with the same name, we insert the module, otherwise the section name, and all the modules.

```
2415  <xsl:template name="table.section_sansTopic">
2416    <xsl:for-each select="/raweb/child::*[self::presentation or ... or self::diffusion]">
2417      <xsl:variable name="varTitle" select="bodyTitle" />
2418      <xsl:choose>
2419        <xsl:when test="count(subsection)=1 and ./bodyTitle=subsection/bodyTitle">
2420          <xsl:for-each select="subsection">
2421            <xsl:call-template name="jg.tdm.avec.ali" />
2422          </xsl:for-each>
2423        </xsl:when>
2424        <xsl:otherwise>
2425          <li>
2426             <xsl:call-template name="jg.tdm.modules" />
2427          </li>
2428        </xsl:otherwise>
2429      </xsl:choose>
2430    </xsl:for-each>
2431  </xsl:template>
```

The structure of the TOC in the case of topics is: for each topic, for each section, for each module, insert an entry. This code takes as parameter `$class`, that will be used for the `<ul>`. The current topic name is in `$num-topic`.

```
2432  <xsl:template name="table.topic_Topic">
2433    <xsl:param name="class" />
2434    <div class="topic">
2435     <xsl:for-each select="/raweb/topic">
2436       <xsl:variable name="TOP" select="normalize-space(.)" />
2437       <xsl:variable name="num-topic" select="@id" />
2438       <h3 class="smallcap"> <xsl:apply-templates /> </h3>
2439       <ul>
2440          <xsl:attribute name="class">
2441            <xsl:value-of select="$class" />
2442          </xsl:attribute>
2443          <xsl:for-each select="/raweb/child::*[self::presentation or ... or self::diffusion]">
2444            <xsl:call-template name="table.topic.aux" />
2445          </xsl:for-each>
2446       </ul>
2447     </xsl:for-each>
2448    </div>
2449    <br />
2450  </xsl:template>
```

This template works on a section; if no modules has `$num-topic` as topic, nothing is done. Otherwise, we have two variables `$TOP` and `$SUB` containing the name of the topic, and the title of the module. If they are different, we output all relevant modules. Otherwise, the situation is unclear. Note: assume that the topic is A, its name (value of `$TOP`) is 'foo', and the test is true, let's say that modules B and C have A as topic attribute. Then `$SUB` is the concatenation of the

titles of B and C; more than often this will not be 'foo'. It might happen that a unique module has topic A, and that this module has the same title as the topic. In this case, you will see the title of the section, and a link to the first module[19]

```
<xsl:template name="table.topic.aux">
  <xsl:if test="subsection[@topic=$num-topic]">
    <xsl:variable name="SUB"
      select="normalize-space(subsection[@topic=$num-topic]/bodyTitle)" />
    <xsl:choose>
      <xsl:when test="$TOP=$SUB">
        <li>
           <a href="{subsection/@id}.html">
              <xsl:value-of select="bodyTitle" />
           </a>
        </li>
     </xsl:when>
     <xsl:otherwise>
       <li>
          <xsl:call-template name="jg.tdm.modules.topic" />
      </li>
      </xsl:otherwise>
  </xsl:choose>
 </xsl:if>
</xsl:template>
```

Are these used?

```
<xsl:template match="subsection/bodyTitle" mode="sansTopic" />
<xsl:template match="subsection/bodyTitle" mode="Topic" />
<xsl:template match="subsection/bodyTitle" />
```

This template is used when we construct the page associated to a module. If the module has no topic attribute, we just print the title of its parent (this is the section title). Otherwise, we produce 'Topic foo', 'Section bar', using a `<h2>` twice.

```
<xsl:template name="section_title_Topic">
    <xsl:choose>
      <xsl:when test="./@topic">
        <xsl:variable name="VarTop" select="@topic" />
        <h2>Topic :
        <xsl:value-of select="/raweb/topic[@id=$VarTop]" /></h2>
        <h2>Section :
        <xsl:value-of select="../bodyTitle" /></h2>
      </xsl:when>
      <xsl:otherwise>
        <h2>Section :
        <xsl:value-of select="../bodyTitle" /></h2>
      </xsl:otherwise>
    </xsl:choose>
 </xsl:template>
```

This is the end of the file.

```
</xsl:stylesheet>
```

[19] According to M. Kay, in XSLT2.0, if the style sheet starts with 'version=2.0', all id attributes of all modules are concatenated, and the result is a link to nowhere; if the style sheet has version=1.0, only the id is used. In any case, it would be better to check that there is a single module that matches, and make the link to it.

6.3 Converting the bibliography

This auxiliary file converts the bibliography from the old DTD to the new one.[20]

```
<xsl:transform
  xmlns:tei="http://www.tei-c.org/ns/1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
```

Usual header.

```
<xsl:output method='xml'
    doctype-public="http://www.inria.fr/xml/raweb/biblio.dtd"
    doctype-system='bibli0.dtd'
    indent='no'
    encoding='iso-8859-1'/>
```

We simplified the code, by assuming that the id is 'bibliography'. The transformation of <biblio> is an element of the same name. We consider all <citation> elements.

```
<xsl:template match="biblio">
  <xsl:element name="biblio">
    <xsl:attribute name="id">bibliography</xsl:attribute>
    <xsl:apply-templates select="citation"/>
  </xsl:element>
</xsl:template>
```

Translation of a <citation> element. The result is a <biblStruct>, depending on the type, it can have one of two alternatives (on page 228, we have the template that converts the entry to HTML, and a comment that says that we check the validity; nothing is done here, we assume that the first or second if-test is true). The last three <note> elements will be used to sort the entries, or for debug.

```
<xsl:template match="citation">
  <xsl:element name="biblStruct">
    <xsl:attribute name="id"><xsl:value-of select="./@id"/></xsl:attribute>
    <xsl:if test="@type='article' or @type='inbook' or @type='incollection'
           or @type='inproceedings' or @type='conference'">
        <xsl:call-template name="citation.analy"/>
    </xsl:if>
    <xsl:if test="@type='book' or @type='booklet' or @type='proceedings' or
        @type='phdthesis'  or @type='techreport' or @type='unpublished' or
        @type='misc' or @type='masterthesis' or @type='masterthesis' or
        @type='manual'">
      <xsl:call-template name="citation.mono"/>
    </xsl:if>
    <xsl:apply-templates select="bhowpublished"/>
    <xsl:element name="note">
      <xsl:attribute name="type">classification</xsl:attribute>
      <xsl:value-of select="./@type"/>
    </xsl:element>
    <xsl:element name="note">
      <xsl:attribute name="type">from</xsl:attribute>
      <xsl:value-of select="./@from"/>
    </xsl:element>
    <xsl:element name="note">
      <xsl:attribute name="type">userid</xsl:attribute>
      <xsl:value-of select="./@userid"/>
    </xsl:element>
```

[20] The bibliography part of this DTD comes frome the TEI. See http://www.tei-c.org/P4X/

```
  </xsl:element>
</xsl:template>
```

In order to make this document shorter, we have used the pseudo command “apply-many-templates” in the first version of this document. It happens that such a kind of shorthand exists in XPath 2.0, with the syntax shown here. The next three templates are specific to the TEI.

```
<xsl:template name="citation.mono">
  <xsl:element name="monogr">
    <xsl:apply-templates select ="btitle, bbooktitle, bseries, bjournal,
      bauteurs, bediteur, bnote, btype, bedition"/>
    <xsl:call-template name="imprint"/>
  </xsl:element>
</xsl:template>
```

This is a bit more complicated, because we have three parts `<analytic>`, `<monogr>` and `<imprint>`.

```
<xsl:template name="citation.analy">
  <xsl:element name="analytic">
    <xsl:apply-templates select="btitle, bauteurs"/>
  </xsl:element>
  <xsl:element name="monogr">
    <xsl:apply-templates select="bediteur, bbooktitle, bseries,
      bjournal, bnote, btype"/>
    <xsl:call-template name="imprint"/>
  </xsl:element>
</xsl:template>
```

This is used to indicate how the reference is published.

```
<xsl:template name="imprint">
  <xsl:element name="imprint">
    <xsl:apply-templates select="bvolume, bchapter, bnumber,
     bpublisher, bschool, borganization, binstitution"/>
    <xsl:call-template name="bdate"/>
    <xsl:apply-templates select="bpages,xref"/>
  </xsl:element>
</xsl:template>
```

Transformation of the `<btitle>`. The result is a `<title>` element, whose attribute depends on the type. We have not shown the value of the second test. Using ‘choose’ and ‘otherwise’ would make the code more robust.

```
<xsl:template match="btitle">
  <xsl:if test="../@type='article' or ../@type='inbook' or
       ../@type='incollection' or ../@type='inproceedings' or ../@type='conference'">
    <title level="a"> <xsl:apply-templates /> </title>
  </xsl:if>
  <xsl:if test="  ... ">
    <title level="m"> <xsl:apply-templates /> </title>
  </xsl:if>
</xsl:template>
```

Transformation of `<bauteurs>`. The result is a `<author>`, containing the same list of authors. We translate all the `<bpers>`, and replace `<etal>` by a person whose first name is ‘ETAL’.

```
<xsl:template match="bauteurs">
  <xsl:element name="author">
    <xsl:for-each select="bpers|etal">
      <xsl:choose>
        <xsl:when test="name()='etal'">
          <xsl:element name="persName">
```

```
            <xsl:element name="foreName"><xsl:text>ETAL</xsl:text></xsl:element>
          </xsl:element>
        </xsl:when>
        <xsl:otherwise> <xsl:call-template name="personne"/> </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:element>
</xsl:template>
```

Transformation of `<beditor>`. The result is a `<editor>`. What about Mister Etal?

```
<xsl:template match="bediteur">
  <xsl:element name="editor">
    <xsl:for-each select="bpers">
      <xsl:call-template name="personne"/>
    </xsl:for-each>
  </xsl:element>
</xsl:template>
```

Transformation of `<bpers>` in `<persName>`. The attributes nom and prenom are translated into `<foreName>` and `<surname>`.[21] The raweb specification says in 2006 to use full first names, whenever possible, in the bibliography; these are stored by Tralics in the prenomcomplet, copied in `<foreName>`, and the initials are in `<initial>`. Note: the actual code uses four `<xsl:element>` elements instead of literal elements.

```
<xsl:template name="personne">
   <persName>
      <foreName>  <xsl:value-of select="@prenomcomplet"/></foreName>>
      <surname>  <xsl:value-of select="@nom"/> </surname>
      <initial>  <xsl:value-of select="@prenom"/> </initial>
    </persName>
</xsl:template>
```

We replace `<bseries>` by `<title>`.

```
<xsl:template match="bseries">
  <title level="s"> <xsl:value-of select="."/>  </title>
</xsl:template>
```

Ditto for `<bjournal>`, with a different attribute value.

```
<xsl:template match="bjournal">
  <title level="j"> <xsl:value-of select="."/>  </title>
</xsl:template>
```

Ditto for `<booktitle>`. In some cases, we add the value of the `<baddress>`.

```
<xsl:template match="bbooktitle">
  <title level="m">
    <xsl:value-of select="."/>
    <xsl:if test="../baddress"> <xsl:text>, </xsl:text><xsl:value-of select="../baddress"/>
    </xsl:if>
  </title>
</xsl:template>
```

We replace `<bnote>` by `<note>`.

```
<xsl:template match="bnote">
  <note type="bnote"> <xsl:value-of select="."/> </note>
</xsl:template>
```

Ditto for `<btype>`, with a different attribute value.

[21] What about part and junior? Note that Tralics generates an empty part attribute, that is omitted from the element, but junior should be added to `<surname>`.

```
<xsl:template match="btype">
  <note type="typdoc"> <xsl:value-of select="."/> </note>
</xsl:template>
```

Ditto for `<bhowpublished>`, with a different attribute value.

```
<xsl:template match="bhowpublished">
 <note type="howpublished">  <xsl:value-of select="."/>  </note>
</xsl:template>
```

The transformation of `<bschool>` is `<publisher>`, with an `<orgName>` that contains the name of the school. It can be followed by the value of the `<baddress>` field of the entry.

```
<xsl:template match="bschool">
 <xsl:element name="publisher">
  <xsl:element name="orgName">
   <xsl:attribute name="type">school</xsl:attribute>
   <xsl:value-of select="."/>
  </xsl:element>
  <xsl:if test="../baddress"><xsl:apply-templates select="../baddress"/></xsl:if>
 </xsl:element>
</xsl:template>
```

Same idea here. But the implementation is different.

```
<xsl:template match="borganization">
  <xsl:element name="publisher">
    <xsl:element name="orgName">
      <xsl:attribute name="type">organisation</xsl:attribute>
      <xsl:value-of select="."/>
      <xsl:if  test="../baddress">
        <xsl:element name="address">
          <xsl:apply-templates select="../baddress"/>
        </xsl:element>
      </xsl:if>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

Same idea here.

```
<xsl:template match="binstitution">
  <xsl:element name="publisher">
    <xsl:element name="orgName">
      <xsl:attribute name="type">institution</xsl:attribute>
      <xsl:value-of select="."/>
      <xsl:if  test="../baddress">
        <xsl:element name="address">
          <xsl:apply-templates select="../baddress"/>
        </xsl:element>
      </xsl:if>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

Strange.

```
<xsl:template match="bedition">
  <xsl:element name="edition">
    <xsl:value-of select="./text()"/>
    <xsl:copy-of select="./*"/>
  </xsl:element>
</xsl:template>
```

Translation of a date into a <dateStruct>. We take the <bmonth> and <byear> and convert them into <month> and <year>. Nothing is done if both fields are missing.

```
<xsl:template name="bdate">
 <xsl:if test="string-length(bmonth|byear)>0">
  <xsl:element name="dateStruct">
   <xsl:if test="string-length(bmonth)>0">
    <xsl:element name="month"> <xsl:value-of select="bmonth"/> </xsl:element>
   </xsl:if>
   <xsl:if test="string-length(byear)>0">
     <xsl:element name="year"> <xsl:value-of select="byear"/> </xsl:element>
   </xsl:if>
  </xsl:element>
 </xsl:if>
</xsl:template>
```

Transformation of <bvolume> into a <biblScope>.

```
<xsl:template match="bvolume">
  <biblScope type="volume">  <xsl:value-of select="."/> </biblScope>
</xsl:template>
```

Transformation of <bchapter> into a <biblScope> with a different attribute value.

```
<xsl:template match="bchapter">
  <biblScope type="chapter">  <xsl:value-of select="."/> </biblScope>
</xsl:template>
```

Transformation of <bnumber> into a <biblScope> with a different attribute value.

```
<xsl:template match="bnumber">
  <biblScope type="number">  <xsl:value-of select="."/> </biblScope>
</xsl:template>
```

Transformation of <bpages> into a <biblScope> with a different attribute value.

```
<xsl:template match="bpages">
  <biblScope type="number">  <xsl:value-of select="."/> </biblScope>
</xsl:template>
```

Transformation of <bpublisher> into a <publisher> containing a <orgName>, and sometimes the address.

```
<xsl:template match="bpublisher">
  <xsl:element name="publisher">
    <xsl:element name="orgName">
      <xsl:value-of select="."/>
      <xsl:apply-templates select="../baddress"/>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

Translation of <baddress> into a <address> containing a <addrLine>.

```
<xsl:template match="baddress">
 <xsl:if test="not(../bbooktitle)">
   <xsl:element name="address">
      <xsl:element name="addrLine"><xsl:value-of select="."/></xsl:element>
   </xsl:element>
 </xsl:if>
</xsl:template>
```

Unused.

```
<xsl:template match="biblio/citation/xref">
 ...
</xsl:template>
```

This is the end of the file.

```
</xsl:transform>
```

6.4 Converting the bibliography into HTML

We explain here what is in the file biblio.xsl that converts the bibliography into HTML code. The input is the result of the translation (code shown in the section above) of what Tralics produces, using a TEI syntax. The full TEI is not yet implemented.

We declare the style sheet and all namespaces.

```
<xsl:stylesheet
  xmlns:tei="http://www.tei-c.org/ns/1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:m="http://www.w3.org/1998/Math/MathML"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:html="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="m html xlink tei">
```

We remove spaces in most elements.

```
<xsl:strip-space elements="biblStruct monogr analytic author editor
   title imprint address addrLine"/>
```

In order to make the following code a bit shorter, we replace '`note[@type='from']/text()`' with `$from` and '`note[@type='classification']/text()`' with `$type`. These two quantities are not defined by the TEI, but are used to sort the bibliography. We make a strong assumption: that `$from` is one of 'year' or 'refer' or 'foot'. Entries are sorted by putting 'refer' first and 'foot' last. We assign category 'd' and 'k' to these entries. The second assumption is that `$type` is one of 'article', 'book', 'booklet', 'inbook', 'incollection', 'inproceedings', 'manual', 'masterthesis', 'misc', 'phdthesis', 'proceedings', 'techreport', 'unpublished'. These values are taken from [7, page 763]. Note that BibTeX defines 'masterthesis' and 'mastersthesis' as being equivalent; the companion lists the name with an 's', Tralics outputs the other one. We accept also 'conference', this is the same as 'inproceedings', 'manuel' (like 'manual', not considered by Tralics) and 'coursenotes'. All these entries are classified[22]: first book, then Ph.D. theses, then articles in journals, then articles in conferences, then technical reports, then anything else. Their type is a letter between 'e' and 'j'.

The following code is invalid XSLT, because variables are forbidden in `<xsl:key>`, but the idea is there. The construction '`1 div X`' is a bit strange. There is an example in [4], that explains that it returns 1 in case X is the boolean value true, and infinity otherwise, so that the substring contains one or zero characters. In any case, this associates to each entry a value, that happens to be a single letter, because at most one substring produces a non-empty string.

```
<xsl:key name="bibliotypes" match="//biblStruct" use="
 concat(
    substring('d', 1 div ($from='refer')),
    substring('e', 1 div ($from='year' and ($type='book' or
             $type='booklet' or  $type='proceedings'))),
    substring('f', 1 div ($from='year' and ($type='phdthesis'))),
    substring('g', 1 div ($from='year' and ($type='article' or
            $type='inbook' or $type='incollection'))),
    substring('h', 1 div  ($from='year' and ($type='inproceedings' or  $type='conference'))),
    substring('i', 1 div ($from='year' and ($type='techreport' or
       $type='manuel' or $type='manual' or  $type='coursenotes'))),
    substring('j', 1 div ($from='year' and ($type='unpublished' or
```

[22] in the sense of 'sorted'; secret documents should not appear in the Raweb.

```
2722          $type='misc' or  $type='masterthesis'))),
2723       substring('k', 1 div ($from='foot'))
2724       )" />
```

Remember that the structure 'bibliotypes' is computed before any global variable is set. As a consequence, we can safely put in a variable the number of entries of each category.

```
2725 <xsl:variable name='d' select="count(key('bibliotypes', 'd'))"/>
2726 <xsl:variable name='e' select="count(key('bibliotypes', 'e'))"/>
2727 <xsl:variable name='f' select="count(key('bibliotypes', 'f'))"/>
2728 <xsl:variable name='g' select="count(key('bibliotypes', 'g'))"/>
2729 <xsl:variable name='h' select="count(key('bibliotypes', 'h'))"/>
2730 <xsl:variable name='i' select="count(key('bibliotypes', 'i'))"/>
2731 <xsl:variable name='j' select="count(key('bibliotypes', 'j'))"/>
2732 <xsl:variable name='k' select="count(key('bibliotypes', 'k'))"/>
```

In order to simplify the code that follows, we show here the source of one of the sections. It is a template with three arguments, a type, for instance 'f', a title, and the number of entries before the first of this type.

```
2733    <xsl:call-template name="tri">
2734      <xsl:with-param name="str" select="'f'"/>
2735      <xsl:with-param name="title" select="'some funny title'"/>
2736      <xsl:with-param name="countPrevious" select="$d+$e"/>
2737    </xsl:call-template>
```

The sections are in order from d to k, and the titles are respectively: 'Major publications by the team in recent years', 'Doctoral dissertations and Habilitation theses', 'Books and Monographs', 'Articles in referred journals and book chapters', 'Publications in Conferences and Workshops', 'Internal Reports', 'Miscellaneous', 'Bibliography in notes'. It was decided, in 2006, to make the origin of the publication more explicit; this means to add 'Year Publications' before entries of type e to j (all but d and k), and to use a smaller font for the subtitles. For this reason, the template shown above has an additional parameter, 'title2' that is the subtitle. We shall abreviate the previous template as *tri('f', $d+$e, 'Tit-f',")*, using the third title in the list; the last element in the list is the subtitle (one of 'title' or 'title2' is empty).

The variable `$except-publis` contains a list of sections that should be omitted. Hence the previous count is a bit more complicated that just computing the sum of `$d` and `$e`. For instance, this is the code that computes the first number for section f.

```
2738  <xsl:variable name="countPrevF">
2739    <xsl:choose>
2740      <xsl:when test="contains($except-publis,'e')">
2741         <xsl:value-of select="$countPrevE" />
2742      </xsl:when>
2743      <xsl:otherwise>
2744         <xsl:value-of select="$countPrevE + $e" />
2745      </xsl:otherwise>
2746    </xsl:choose>
2747  </xsl:variable>
```

In order to make the code shorter, we have abbreviated the previous template as *update-count(countPrevF, 'e', countPrevE)*. There is something strange in the code shown here: only section d is conditionally included.

```
2748 <xsl:template name="tri-par-publis">
2749   <xsl:param name="except-publis" />
2750   <div class="publis">
2751     <xsl:if test="not( contains($except-publis,'d') )">
2752       tri('d', '0', 'Tit-d', '')
2753     </xsl:if>
```

```
    update-count(countPrevE, d, 0)
    <a name="year"><h2>Year Publications</h2></a>
    tri('e', countprevE, '', 'Tit-e')
    update-count(countPrevF, e, countprevE)
    tri('f', countPrevF, '', 'Tit-f')
    update-count(countPrevG, f, countprevF)
    tri('g', countPrevG, '', 'Tit-g')
    update-count(countPrevH, g, countprevG)
    tri('h', countPrevH, '', 'Tit-h')
    update-count(countPrevI, h, countprevH)
    tri('i', countPrevI, '', 'Tit-i')
    update-count(countPrevJ, i, countprevI)
    tri('j', countPrevJ, '', 'Tit-j')
    update-count(countPrevK, k, countprevJ)
    tri('k', countPrevK, 'Tit-k', '')
  </div>
</xsl:template>
```

What the code does is obvious: we consider all entries that are of type `$str`. If the list is empty, nothing is done. Otherwise, the `$title` is output in a `<h2>`, or `$title2` is output in a `<h4>`. Note that this code can produce two anchors: 'Major' and 'References'. After that, we output all elements of the list, sorted by author's name. The template that outputs the entry takes a number: it is the index of the entry, the value of `$countPrevious` plus the index in the sorted list.

```
<xsl:template name="tri">
  <xsl:param name="str"/>
  <xsl:param name="title"/>
  <xsl:param name="title2"/>
  <xsl:param name="countPrevious"/>
  <xsl:if test="key('bibliotypes',$str)[1]">
    <xsl:if test="string-length($title)>0">
       <a name="{substring-before($title,' ')}">
         <h2><xsl:value-of select='$title'/></h2>
       </a>
    </xsl:if>
    <xsl:if test="string-length($title2)>0">
      <h4><xsl:value-of select='$title2'/></h4>
    </xsl:if>
    <dl>
    <xsl:for-each select="key('bibliotypes',$str)">
       <xsl:sort select="descendant::author[1]/persName[1]/surname/text()"/>
       <xsl:apply-templates select=".">
          <xsl:with-param name="pos">
            <xsl:value-of select="$countPrevious+position()"/>
          </xsl:with-param>
       </xsl:apply-templates>
    </xsl:for-each>
    </dl>
  </xsl:if>
</xsl:template>
```

This is used more than once, hence we introduced this template (this is not valid XSL code, because it uses an undeclared variable `$pos`, which is declared by the caller).

```
<xsl:template name="jp.pos">
    <xsl:text>[</xsl:text>
    <xsl:value-of select="$pos"/>
```

```
2800      <xsl:text>]</xsl:text>
2801  </xsl:template>
```

This template exists in two versions: the version given here terminates if the type is not in the list given above (code omitted). It converts a `<biblStruct>` into a `<dt>`, that contains an anchor, the number and key of the reference, and a `<dd>` that contains the reference. A non trivial point concerns punctuation. The idea is that there is a period after the list of authors, and at the end of the citation. Each item is preceded by a comma, except the author list (that is at the start of an entry) or the title (that comes after the authors).

```
2802  <xsl:template match="biblStruct">
2803    <xsl:param name="pos" select= "position()"/>
2804    <xsl:check-this-entry />
2805    <dt class="bib">
2806      <xsl:call-template name="jg.pos" />
2807      <xsl:text>[</xsl:text>
2808      <xsl:value-of select="note[@type='userid']"/>
2809      <xsl:text>]</xsl:text>
2810      <a name="{./@id}"></a>
2811    </dt>
2812    <dd> <xsl:apply-templates/> <xsl:text>.</xsl:text> </dd>
2813    <xsl:text>&#x0a;</xsl:text>
2814  </xsl:template>
```

Same code as above, with the following modifications; the user-id field is omitted (this is the cite key of the LaTeX source, it has nothing to do in the HTML file, but is useful for debug). On the other hand, if the `$xyleme` variable is set, a different anchor is used (the year string should be changed to 2006).

```
2815  <xsl:template match="biblStruct" mode="debug">
2816   <xsl:param name="pos" select="position()" />
2817   <dt class="bib">
2818    <xsl:choose>
2819      <xsl:when test="$xyleme='1'">
2820       <a name="{./@id}" class="ancre" title="référence en bibtex"
2821         href="../../2004/publiLatex.text?ref={./@id}&amp;projet={$projet}">
2822        <xsl:call-template name="jg.pos" />
2823       </a>
2824      </xsl:when>
2825      <xsl:otherwise>
2826       <a name="{./@id}" class="ancre">
2827       <xsl:call-template name="jg.pos" />
2828      </xsl:otherwise>
2829    </xsl:choose>
2830   </dt>
2831   <dd>  <xsl:apply-templates/><xsl:text>.</xsl:text> </dd>
2832  </xsl:template>
```

Most entries are trivial: a comma and the content. The case of `<edition>` is special: the LaTeX companion says that the value should be something like 'Second'. There is no such problem for `<orgName>` or `<addrLine>`.

```
2833  <xsl:template match="edition|orgName|addrLine">
2834    <xsl:text>, </xsl:text> <xsl:apply-templates/>
2835  </xsl:template>
```

Translation of `<biblScope>`. This depends on the `type` attribute. In the case 'volume', we output something like "vol. 25".

```
<xsl:template match="biblScope[@type='volume']">
  <xsl:text>, vol. </xsl:text> <xsl:apply-templates/>
</xsl:template>
```

In the case 'chapter', we output something like "chap. 25".

```
<xsl:template match="biblScope[@type='chapter']">
  <xsl:text>, chap. </xsl:text> <xsl:apply-templates/>
</xsl:template>
```

In the case 'number', we output something like "n° 25".

```
<xsl:template match="biblScope[@type='number']">
  <xsl:text>, n</xsl:text><sup>o</sup><xsl:text> </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
```

In the case of 'pages' we output something like "p. 10–30". If the text contains neither a dash nor an en-dash, we outpout "42 p".

```
<xsl:template match="biblScope[@type='pages'][text()!='']">
  <xsl:text>, </xsl:text>
  <xsl:choose>
    <xsl:when test="string-length(substring-before(., '-'))
                or string-length(substring-before(., '&#8211;'))>0">
       <xsl:text>p. </xsl:text>
       <xsl:apply-templates/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates/>
      <xsl:text> p</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Translation of `<note>`. This is trivial, but used only if the `type` attribute is 'bnote' (a BibTeX note), 'typdoc' (this could be the type of a report) or 'howpublished' (for a non-standard entry).

```
<xsl:template match="note[@type='howpublished' | @type='bnote' | @type='typdoc']">
    <xsl:text>, </xsl:text>
    <xsl:apply-templates/>
</xsl:template>
```

These notes are used for sorting, their printed value is empty.

```
<xsl:template match="note[@type='from' |@type='userid' | @type='classification']">
</xsl:template>
```

Translation of `<title>`. Is this really needed?

```
<xsl:template match="title[ancestor::biblio]">
   <xsl:text>"</xsl:text> <xsl:apply-templates/><xsl:text>"</xsl:text>
</xsl:template>
```

Translation of `<title>`. The attribute says that it is the name of a journal.

```
<xsl:template match="title[@level='j']">
  <xsl:text>, in: </xsl:text>
  <span class="journal"> <xsl:apply-templates/> </span>
</xsl:template>
```

Translation of `<title>`. The attribute says that it is the name of a series.

```
<xsl:template match="title[@level='s']">
  <xsl:text>, </xsl:text> <xsl:apply-templates/>
</xsl:template>
```

Translation of `<title>`. The attribute says that it is the title of an article, a book, or something like that. The result is a `<span>` with an attribute that says to use a different font.

```
<xsl:template match="title[@level='a']">
   <span class="journal"> <xsl:apply-templates/> </span>
</xsl:template>
```

Translation of `<title>`. The attribute says that it is something different than the cases considered above. The TEI says something like: If the title appears directly enclosed within an `<analytic>` element, the level, if given, must be 'a'; if it appears directly enclosed within a `<monogr>` element, level must be 'm', 'j', or 'u'; when `<title>` is directly enclosed by `<series>`, level must be 's'. As a consequence, if the test is true, we have two titles (for instance a book title, and the title of a part of the book).

```
<xsl:template match="title[@level='m']">
  <xsl:choose>
    <xsl:when test="string-length(../../analytic/title) > 0">
      <xsl:text>, in: </xsl:text>
      <span class="journal"><xsl:apply-templates/></span>
    </xsl:when>
    <xsl:when test="string-length(..//author) > 0
             and string-length(..//editor) > 0 ">
      <xsl:text>, </xsl:text>
      <span class="journal"> <xsl:apply-templates/> </span>
    </xsl:when>
    <xsl:otherwise>
      <span class="journal"> <xsl:apply-templates/>  </span>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Translation of `<author>`. We consider all `<persName>` in the list, typesetting them one after the other with a comma as separator, and a period at the end. There is a `<br>` at the end of the list. A special case is when the first name is ETAL. Otherwise, we output the forename (either initials in `<intial>` or whatever is in `<foreName>`) and the `<surname>`.

```
<xsl:template match="author">
 <xsl:for-each select="persName">
   <xsl:choose>
     <xsl:when test="foreName='ETAL'">
        <small><xsl:text>et al.</xsl:text></small>
     </xsl:when>
     <xsl:otherwise>
       <xsl:choose>
         <xsl:when test="initial">
           <xsl:apply-templates select="initial"/>
         </xsl:when >
         <xsl:otherwise>
           <xsl:apply-templates select="foreName"/>
         </xsl:otherwise>
       </xsl:choose>
        <xsl:value-of select="foreName"/>
        <xsl:text> </xsl:text>
        <xsl:apply-templates select="surname" />
        <xsl:call-template name="separateur.objet"/>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:for-each>
```

```
 <br/>
</xsl:template>
```

The translation of `<editor>` is similar. If an author and an editor are given, they are separated by a comma (remember that there is a line break after the authors). We add a `<br>` at the end in the case where no author is given. Editors are separated by commas, but at the end, we put a '(editor)' or '(editors)' remark.

```
<xsl:template match="editor">
 <xsl:if test="../..//author">
   <xsl:if test="ancestor-or-self/node()/title[@level='a']">
     <xsl:text>, </xsl:text>
   </xsl:if>
 </xsl:if>
 <xsl:for-each select="persName">
   <xsl:choose>
     <xsl:when test="foreName='ETAL'">
       <small><xsl:text>et al.</xsl:text></small>
     </xsl:when>
     <xsl:otherwise>
       <xsl:choose>
         <xsl:when test="initial">
           <xsl:apply-templates select="initial"/>
         </xsl:when >
         <xsl:otherwise>
           <xsl:apply-templates select="foreName"/>
         </xsl:otherwise>
       </xsl:choose>
       <xsl:text> </xsl:text>
       <xsl:apply-templates select="surname" />
       <xsl:call-template name="separateurED.objet"/>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:for-each>
 <xsl:text> (editor</xsl:text><xsl:call-template name="pluriel-p">
      <xsl:with-param name="liste" select="persName" /></xsl:call-template>
 <xsl:text>).</xsl:text>
 <xsl:if test="not(../../*/author)"><br/></xsl:if>
</xsl:template>
```

This is used in the case where a name (say Dupont) appears as an autor or editor in the bibliography. We put in in a with 'smallcaps' attribute. Moreover, if is true, and the author is member of the Team (there is a `<person>` with `<lastname>` Dupont), an anchor is created, with value root/Publications/2006/publications.html?name=Dupont&projet=apics.

```
<xsl:template match="surname">
  <span class="smallcap">
    <xsl:choose>
      <xsl:when test="$xyleme='1'
           and //person/lastname[text()=current()/text()]">
        <a>
          <xsl:attribute name="href">
            <xsl:value-of select="$rootUrl"/>
            /Publications/
            <xsl:value-of select="$year" />
            /publications.html?name=
            <xsl:apply-templates/>&amp;projet=
            <xsl:value-of select="$projet" />
          </xsl:attribute>
```

```
          <xsl:apply-templates />
        </a>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates />
      </xsl:otherwise>
    </xsl:choose>
  </span>
</xsl:template>
```

This seems to be useless.

```
<xsl:template match="bedition"> ? </xsl:template>
```

Case of a <dateStruct>. We output the <month> and the <year>.

```
<xsl:template match="dateStruct">
  <xsl:text>, </xsl:text>
  <xsl:value-of select="month"/><xsl:text> </xsl:text><xsl:value-of select="year"/>
</xsl:template>
```

Case of <publisher>. We output the two parts: <orgName> and optionally <address>.

```
<xsl:template match="publisher">
  <xsl:apply-templates select="orgName"/>
  <xsl:if test="./address"><xsl:apply-templates select="address"/> </xsl:if>
</xsl:template>
```

Case of <address>. No initial comma. This is commented out in the 2006 version.

```
<xsl:template match="address">
  <xsl:apply-templates/>
</xsl:template>
```

Case of a reference. We assume that it is always an external reference. Moreover, the program that converts from the old DTD to the new one never adds a url attribute[23]. The essential difference between the two cases is that a <br> is added before the link.

```
<xsl:template match="biblStruct//ref">
  <xsl:choose>
   <xsl:when test="./@url">
     <a>
       <xsl:attribute name="target">_alt</xsl:attribute>
       <xsl:attribute name="href"><xsl:value-of select="./@url"/></xsl:attribute>
       <xsl:apply-templates/>
     </a>
   </xsl:when>
   <xsl:otherwise>
      <xsl:text>, </xsl:text>
      <br/>
      <a>
        <xsl:attribute name="target">_alt</xsl:attribute>
        <xsl:attribute name="href"><xsl:value-of select="."/></xsl:attribute>
        <xsl:apply-templates/>
      </a>
   </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Case of <analytic>. We output <author>, <title>, <editor>, and <imprint>. If no author is given the editor is put before the title.

[23]The value of the link is in xlink. It happens that Tralics puts the same value in the link and in the attribute in most of the cases. But this is liable to change.

```
<xsl:template match="analytic">
  <xsl:apply-templates select="author"/>
  <xsl:choose>
    <xsl:when test="author">
      <xsl:apply-templates select="title"/>
      <xsl:apply-templates select="editor"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates select="editor"/>
      <xsl:apply-templates select="title"/>
    </xsl:otherwise>
  </xsl:choose>
 <xsl:apply-templates select="imprint"/>
</xsl:template>
```

Case of `<monogr>`. We output `<title>`, `<author>`, and `<editor>`, in some strange order, followed by `<note>`, `<edition>` and `<imprint>`.

```
<xsl:template match="monogr">
 <xsl:choose>
  <xsl:when test="../analytic">
    <xsl:apply-templates select="title[@level='m']"/>
    <xsl:apply-templates select="title[@level='j']"/>
    <xsl:apply-templates select="author"/>
    <xsl:apply-templates select="editor"/>
    <xsl:apply-templates select="title[@level='s']"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:apply-templates select="author"/>
    <xsl:apply-templates select="editor"/>
    <xsl:apply-templates select="title"/>
   </xsl:otherwise>
 </xsl:choose>
 <xsl:apply-templates select="note"/>
 <xsl:apply-templates select="edition"/>
 <xsl:apply-templates select="imprint"/>
</xsl:template>
```

Case of `<imprint>`. We output `<publisher>`, `<dateStruct>`, `<biblScope>`, and `<ref>`.

```
<xsl:template match="imprint">
  <xsl:apply-templates select="publisher"/>
  <xsl:apply-templates select="dateStruct"/>
  <xsl:apply-templates select="biblScope"/>
  <xsl:apply-templates select="ref"/>
</xsl:template>
```

Consider the following piece of code; call it *position-in-bib(W)*, where W can be a letter between d and k. Each `'d'` should be replaced by W, and `"0"` by the sum of `$d`, `$e`, etc, up to, but excluded `$W`.

```
  <xsl:when test="count($curelement|key('bibliotypes', 'd'))
                =count(key('bibliotypes', 'd'))">
    <xsl:call-template name="positionInBib">
      <xsl:with-param name="str" select="'d'" />
      <xsl:with-param name="current" select="$curelement" />
      <xsl:with-param name="countPrevious" select="0" />
    </xsl:call-template>
  </xsl:when>
```

Here the test compares two values, `count(A|B)` and `count(B)`. The test is true if A is in B (A is a node, B is a node-set, `count(A|B)` is the union of B and the set that contains only A). Such a construct is given as an example in [4] under the entry `<xsl:key>`. It happens that bibliography entries are sorted, by type, a type is a letter between d and k, and, for each type, by author. Consider the n-th item, let's call it X; question: what is n? Assume that the entry is of type m, and there are p entries of type less than m; if this entry is the q-th of type m, then n=p+q. All entries of type m can be found using `key`, with argument 'bibliotypes' and m. The test in the routine above is then: is `$curelement` of type 'e'? If so, we call the template with three arguments: `$str` that holds the type, `$current` the entry to test, and `$countPrevious` the quantity p. The variable `$d` holds the number of entries of type 'd', it is computed on line 2728. Thus, we obtain p by adding some of `$d`, `$e`, `$f`, `$g`, `$h`, `$i`, and `$j` (the quantity `$k` is not used). Computing q is not trivial: we have a function that computes the index of the current node Y in a list, it is the `position()` function. We want the position of X, we obtain it by concatenation of all *f(Y)*, where *f(Y)* is the position of Y if Y is X, empty otherwise. The nodes X and Y are the same if they have the same unique id, not if they have the same text. Normally, the number created here is the same as the one computed on line 2750 (the number associated to the entry), it is created for each reference to the bibliography.

```
3052 <xsl:template name="positionInBib">
3053   <xsl:param name="str" />
3054   <xsl:param name="countPrevious" />
3055   <xsl:param name="current" />
3056   <xsl:for-each select="key('bibliotypes',$str)">
3057     <xsl:sort select="descendant::author[1]/persName[1]/surname/text()" />
3058     <xsl:if test="generate-id(.)=generate-id($current)">
3059        <xsl:value-of select="$countPrevious+position()" />
3060     </xsl:if>
3061   </xsl:for-each>
3062 </xsl:template>

3063 <xsl:template match="ref" mode="ref2biblio">
3064     <xsl:param name="curelement" />
3065     <a>
3066       <xsl:attribute name='href'>
3067         <xsl:call-template name="formaturl">
3068           <xsl:with-param name="base" select="'bibliography'" />
3069         </xsl:call-template>
3070         <xsl:value-of select="@xlink:href" />
3071       </xsl:attribute>
3072       <xsl:text>[</xsl:text>
3073       <xsl:choose>
3074          position-in-bib(d)
3075          position-in-bib(e)
3076          position-in-bib(f)
3077          position-in-bib(g)
3078          position-in-bib(h)
3079          position-in-bib(i)
3080          position-in-bib(j)
3081          position-in-bib(k)
3082          <xsl:otherwise>
3083            <xsl:message>Ooops reference non trouvee</xsl:message>
3084          </xsl:otherwise>
3085       </xsl:choose>
3086       <xsl:text>]</xsl:text>
3087    </a>
```

```
</xsl:template>

```

This is the end of the file.

```
</xsl:stylesheet>
```

6.5 Helper style sheets

We describe here two style sheets, verif-aut.xsl and verif-membres.xsl. The purpose of these two style sheets is to produce small HTML files, containing the list of authors, cited in the bibliography, with their cite key, or the list of members of the team, with their affiliations; this allows authors to quickly check typos in their document.

We shall not show all details, like attributes of the `<xsl:stylesheet>` element, they are similar to those shown elsewhere in this document. The verif-aut.xsl file contains a rule for the top-level element, that says to consider only the bibliography.

```
<xsl:template match="/raweb">
  <xsl:apply-templates select="biblio" />
</xsl:template>
```

The result is a simple HTML file. It contains a title, and an anchor to the second file, and a list, whose content will be explained later.

```
<xsl:template match="biblio">
 <html>
   <head>
     <title>Verification des auteurs référencés dans la bibliographie </title>
     <style type="text/css">
       h1 {font-weight:bold;font-size:16pt;}
       dt {font-weight:bold;color:blue;}
     </style>
   </head>
   <body>
    <h1>Les auteurs référencés dans la bibliographie</h1>
    <a href="membres-{$projet}.html">Voir les membres de l'équipe</a> <!--$-->
     <dl>
...
     </dl>
   </body>
 </html>
</xsl:template>
```

This piece of code selects all `<persName>` elements; these are authors or editors. Consider such a person X, with first name A, last name B; we sort these persons, alphabetically, on the last name, then first name. We consider the bibliography entry Y that has X as author or editors (this is a `<biblStruct>`) and its cite key C (this is one of `<note>` children of Y), and output it. Quantities A and B are also printed, but only once.

```
 <xsl:for-each select="//persName">
   <xsl:sort select="surname"/>
   <xsl:sort select="foreName"/>
   <xsl:if test="current()[generate-id(.)
               =generate-id(key('authorslist',surname)[1])]">
    <dt>
      <xsl:apply-templates select="surname"/>
```

```
3119        <xsl:text>, </xsl:text> <xsl:value-of select="foreName"/>
3120       </dt>
3121     </xsl:if>
3122     <dd>
3123       <xsl:apply-templates
3124          select="ancestor::biblStruct/note[@type='userid']"/>
3125     </dd>
3126
3127  </xsl:for-each>
```

This piece of code constructs a 'key', an association list, that associates to each author or editor X its last name B. Said otherwise, *key('authorslist', B)* gives the list `<persName>` whose last name is B. If this list is L, then *generate-id(L[1])* is the unique id of the first X with last name B. Assume that there is only one person named B, and that its first name is always given identically; then the test shown above will output the name of the person before its first reference (otherwise, it will be put randomly because of the second sort instruction).

```
3128 <xsl:key name="authorslist"
3129      match="//biblio//persName"
3130      use="surname" />
```

Action is trivial here.

```
3131 <xsl:template match="note">
3132   <xsl:apply-templates />
3133 </xsl:template>
3134
3135 <xsl:template match="surname">
3136   <xsl:apply-templates />
3137 </xsl:template>
```

The second style sheet is similar. It constructs an HTML page, with a link to the first one.

```
3138 <xsl:template match="/raweb">
3139   <html>
3140     <head>
3141       <title>Verification des membres de l'équipe</title>
3142       <style type="text/css">
3143          h1 {font-weight:bold;font-size:16pt;color:blue;}
3144          table {border :2px solid blue;border-collapse:collapse;}
3145          tr {border :1px solid blue;padding:5px;}
3146          td {border :1px solid blue;padding:5px;}
3147         td.head {font-weight:bold;border :1px solid red;padding:5px;
3148                background-color:#ffe3d1;}
3149       </style>
3150     </head>
3151     <body>
3152       <h1>Les membres de l'équipe avec les catégories et affiliations</h1>
3153       <a href="auteurs-{$projet}.html">  <!-- $ emacs -->
3154          Les auteurs référencés dans la bibliographie
3155       </a>
3156 ...
3157     </body>
3158  </html>
3159 </xsl:template>
```

The firt item in the file is the number of people entitled to supervise PhD students. It is just the number of `<hdr>` elements.

```
<p>Nombre de personnes habilitées à diriger des recherches :
   <span style="font-weight:bold;">
     <xsl:value-of select="count(//hdr)"/>
   </span>
</p>
```

Then comes a table, with a header and one line for each team member.

```
<table>
 <tr>
   <td class="head">Nom</td>
   <td class="head">Affiliation</td>
   <td class="head">Catégorie profession</td>
   <td class="head">Habilité</td>
 </tr>
 <xsl:apply-templates select="//team/participants/person"/>
</table>
```

The important point here is to output the `<categoryPro>` and affiliation, since they do not appear in the normal HTML file.

```
<xsl:template match="person">
  <tr>
    <td>
      <xsl:apply-templates select="firstname"/>
      <xsl:text> </xsl:text>
      <xsl:apply-templates select="lastname"/></td>
    <td><xsl:apply-templates select="categoryPro"/></td>
    <td><xsl:apply-templates select="affiliation"/></td>
    <td><xsl:apply-templates select="hdr"/></td>
  </tr>
</xsl:template>

<xsl:template match="firstname|lastname|affiliation|profession">
  <xsl:apply-templates />
</xsl:template>
```

We consider now a last style sheet rawebindex.xsl that creates an index. It defines some variables, for instance, `$LeProjet`, containing the team name. The main template template here is the following. Note that the output method is plain text.

```
<xsl:template match="raweb">
  <xsl:document href="{$LeProjet}_mcl" method="text" encoding="iso-8859-1">
    <xsl:call-template name="keywords"/> <!-- $emacs -->
  </xsl:document>
</xsl:template>
```

A rule is applied to each section.

```
<xsl:template name="keywords">
  <xsl:apply-templates select="/raweb/composition"   mode="mkindex"/>
  <xsl:apply-templates select="/raweb/presentation"  mode="mkindex"/>
  <xsl:apply-templates select="/raweb/fondements"    mode="mkindex"/>
  <xsl:apply-templates select="/raweb/domaine"       mode="mkindex"/>
  <xsl:apply-templates select="/raweb/logiciels"     mode="mkindex"/>
```

```
3200    <xsl:apply-templates select="/raweb/resultats"     mode="mkindex"/>
3201    <xsl:apply-templates select="/raweb/contrats"      mode="mkindex"/>
3202    <xsl:apply-templates select="/raweb/international" mode="mkindex"/>
3203    <xsl:apply-templates select="/raweb/diffusion"     mode="mkindex"/>
3204  </xsl:template>
```

The code starts by computing a variable in a strange way; we simplified it a bit; it is the name of the current section (for instance, 'Scientific Foundation'). For each keyword, for instance 'Rational Approximation', we convert it to upper case. In order to make the code shorter, we have written 'S' instead of 'ancestor-or-self::subsection'. If we have a section A, with a subsection B, with a subsection C, with a keyword D, then 'S' contains C and D (in document order), but the last element on this ancestor-or-self axis is C. If the of this selement is uid14, and the team is Apics, this piece of code prints a line of the form *MCL:RATIONAL APPROXIMATION:apics/uid14.html:Team : Apics - Scientific Foundations - Identification and deconvolution*. Here the last item is the title of subsection B, it is omitted when equal to A (modulo case, in all three cases, the two strings given to `translate` have 26 characters.)

```
3205  <xsl:template match="composition|presentation|fondements|domaine|
3206         logiciels|resultats|contrats|international|diffusion"
3207        mode="mkindex" >
3208    <xsl:variable name="SECTION">
3209      <xsl:value-of select="bodyTitle" />
3210    </xsl:variable>
3211    <xsl:for-each select="subsection">
3212      <xsl:for-each select=".//keyword">
3213        <xsl:text>MCL:</xsl:text>
3214        <xsl:value-of select="normalize-space(translate(.,'q','Q'))"/>
3215        <xsl:text>:</xsl:text>
3216        <xsl:value-of select="$LeProjet"/>
3217        <xsl:text>/</xsl:text>
3218        <xsl:value-of select="S[position()=last()]/@id"/>
3219        <xsl:text>.html:Team : </xsl:text>
3220        <xsl:value-of select="$PROJET"/><xsl:text> - </xsl:text>
3221        <xsl:value-of select="$SECTION"/>
3222
3223        <xsl:variable name="toto" select="translate(S/bodyTitle,'Q','q')"/>
3224        <xsl:variable name="toti" select="translate($SECTION,'Q','q')"/>
3225        <xsl:if test="$toto!=$toti">
3226         <xsl:text> - </xsl:text>
3227         <xsl:apply-templates select="S[position()=last()]/bodyTitle"/>
3228        </xsl:if>
3229        <xsl:text>&#x0a;</xsl:text>
3230      </xsl:for-each>
3231    </xsl:for-each>
3232  </xsl:template>
```

6.6 The raweb CSS style sheet

We describe here the content of the file raweb.css, containing rendering information for the Raweb. Recall the syntax: we have a keyword K, followed by an open brace, a sequence of properties, and a closing brace. A property is a name, a colon, a value, and a semi-colon as a separator.

If the keyword K is a name, this is the name of the element. Here we say that the `<body>` element (this is the whole HTML page) should have a white background, black foreground, and the font should be Helvetica, or Arial, or some other sans-serif font.

```
body {
    background-color:#ffffff;
    color:#000000;
    font-family: Helvetica, Arial,sans-serif }
```

If the keyword K contains a dot (say 'foo.bar'), then it applies if the element `<foo>` has 'bar' as style. The following rule says that the `<body>` of the TOC has a different background color.

```
body.tdm {
    background-color:#DDDDDD;
    color:#000000;
    font-family: Helvetica, Arial,sans-serif }
```

These rules indicate the size of the headings, `<h1>`, `<h2>`, `<h3>`, `<h4>`, and `<h5>`.

```
h1 { font-size : 150% }
h2 { font-size : 140% }
h3 { font-size : 130% }
h4 { font-size : 110% ; font-style: italic }
h5 { font-size : 90% ;}
```

These lines (as well as following lines with keywords in capital letters) were in the original style sheet, created by `latex2html`; they are not used anymore. The idea is the following: Translation of `\huge` is a `<big>` element, with style 'huge'. A browser that understands CSS will use a 'larger' font, a browser that does not understand CSS will use the font associated to `<big>`. By default, Tralics knows only three sizes of fonts (normal, small and large).

```
SMALL.XTINY             { font-size : xx-small; }
SMALL.TINY              { font-size : x-small;  }
SMALL.SCRIPTSIZE        { font-size : smaller;  }
SMALL.FOOTNOTESIZE      { font-size : small ;   }
BIG.XLARGE              { font-size : large ;   }
BIG.XXLARGE             { font-size : x-large;  }
BIG.HUGE                { font-size : larger ;  }
BIG.XHUGE               { font-size : xx-large; }
```

If the keyword has the form '.bar' or '*.bar', it applies to every element with class 'bar'. The rules here explain how to locally change the font properties.

```
.smaller{ font-size : smaller   }
.small  { font-size : small    }
.large  { font-size : large    }
.bold   { font-weight : bold    }
.it     { font-style : italic    }
*.center     { text-align:center }
*.smallcap     { font-variant:small-caps; }
```

More properties for the `<span>` element. The last rule applies in the case where the lement is `<a>` in a `<span>` with style 'keyword'.

```
SPAN.textit             { font-style: italic  }
SPAN.textbf             { font-weight: bold  }
span.keyword            { font-style: italic  }
span.keyword a          { font-style: italic  }
```

These properties apply to the header of the page, containing the navigation buttons; the first page is white, other are light grey. The first page has a thin border, which has the same color as the big rule that separates the metadata and the TOC (the color is some kind of pink).

```
.premiere     { background-color:#ffffff ;
                color:#000000 ;
                border-width: 1px;
                border-color: #D60098;
                border-style:solid solid solid solid ;
                height: 70px;
                padding: 0px 5px 0px 5px
              }
.autre     {  background-color:#dddddd ; height: 70px;}
.rose {background-color: #D60098; height:3px}
```

Properties for math. The first three lines are from `latex2html`, not used anymore (if Tralics sees a font change in a math formula, it constructs a MathML element that will be converted into an image. If the formula needs an equation number, a table will be constructed, in order to get correct alignment. Cells in this table have no border.

```
.MATH    { font-family: "Century Schoolbook", serif; }
.MATH I  { font-family: "Century Schoolbook", serif; font-style: italic }
.BOLDMATH { font-family: "Century Schoolbook", serif; font-weight: bold }
div.mathdisplay table { border:none }
div.mathdisplay td    { border:none }
div.mathdisplay tr    { border:none }
```

Figures are generally centered via the use of tables; in these cases we set border and margin to zero. The with of the outer table is 80 percent of the whole page (why?)

```
table.objectContainer       {  border:none; width:80%; margin:0; }
table.objectContainer td    {  border:none; margin:0;  }
table.objectContainer tr    {  border:none; margin:0;  }
table.objectContainer table {  border:none; }
caption {font-size:80%; padding-bottom:10px;}
```

By default, we show border of tables.

```
td    { border:1px solid}
tr    { border:1px solid}
table { border:1px solid; empty-cells:show;  border-collapse:collapse; }
```

A `<ul>` in a `<dd>` is indented by one em (this is strange: what about other lists (dl or ol) inside items of other list, such as li). There are now two rules for `<dt>` elements in the bibliography (containing a cite key). This element is to be shown in a bold font. The next rule has the form 'foo.bar+gee', it applies to every 'gee' element preceded by a 'foo' with style bar. The rule has a negative margin-top, not shown here, because the cite key is overwritten by the author list. We have three rules for list items in the TOC frame.

```
dd ul { margin-top:1em;}
dt.bib  { display: inline; font-weight: bold; }
dt.bib + dd { vertical-align:top;  margin-bottom: 1em; }
ul.tdm_windows {  }
ul.tdm_frame {margin-top:0px;margin-bottom:10px;}
li.tdm_frame {margin-left:-10px;}
```

These two rules explain how 'Keywords' and 'Participants' should be printed. Currentlt, they are underlined.

```
*.KW {  text-decoration: underline }
*.part { text-decoration: underline }
```

These are unused.

```
.journal { font-style: italic }
*.xyHighlight { background-color: yellow;}
```

The following rules explain how to render the buttons that are on the top right part of the page. A rule of the form #foo applies to every element that has and id attribute with value 'foo'. A rule of the form #foo:bar applies if this element has status 'bar'; here the element is an anchor, and we specify how the color of the anchor should changed when visited, etc.

```
.folderButtons {
    position:absolute;
    background-color:#BCBCF9 ;
    top:14px;right:12px ;height: 60px ;width: 180px }
.folderLine    { color: #000000 }
.folderText {font-size : smaller; text-align:right; }
#folderIconRef         { border:0px ;float:left; }
#folderIconRef:link    { color:#dddddd; }
#folderIconRef:visited { color:#dddddd; }
```

Six rules that say where to put navigation buttons (left, right, center, or top, bottom).

```
#head_adroite    {  float:right; }
#head_agauche    {  float:left;  }
#head_aucentre   {  text-align:center ; width:70% ;font-size : smaller;}
#tail_agauche    {  float:left ; }
#tail_adroite    {  float:right; }
#tail_aucentre   {  text-align:center; width:100%; }
```

Four rules that appear in the TOC frame, make the distinction between topics and non-topics obvious. Call these C, F, CT and FT. In the case without topics, we will have C followed by T, otherwise FT followed by CT. The first element is left aligned, the second is right aligned. The C and CT have a lighter color than the F and FT.

```
#clair   {
    background-color:#BCBCF9; width:48% ; height:20px;
    float:left; border-style:solid solid none solid ;border-width:thin ;
    text-align:center;font-size : smaller;}
#fonce   {
   background-color:#594FBF; width:48%  ; height:20px;
   float:right; border-style:solid;border-width:thin ;
   text-align:center;font-size : smaller;}
#clair_T {
   background-color:#BCBCF9; width:48% ; height:20px;
   float:right; border-style:solid solid none solid ;border-width:thin ;
   text-align:center;font-size : smaller;}
#fonce_T {
   background-color:#594FBF; width:48% ; height:20px;
   float:left; border-style:solid;border-width:thin ;
   text-align:center;font-size : smaller;}
```

This is for the logo at the bottom of the page. The text has attribute 'display:none', so that you will not see it, unless the browser cannot show the image, and uses the text instead.

```
#bandeau {
  clear: both;
```

```
3332   width: 100%;
3333   height: 90px;
3334   background-color: #594FBF;
3335   background-image: url(icons/bandeau_g.gif);
3336   background-repeat: no-repeat;
3337   text-align:right;}
3338 #bandeau_logo {
3339    float: right;
3340    height: 90px;
3341    width: 329px;
3342    padding: 0px;
3343    background: url(icons/bandeau_d.gif);}
3344 #logotext {  display: none }
```

This is for the logo on the TOC frame.

```
3345 *.logo { background-color:#594FBF; color: #FFFFFF;  text-align:center; }
```

Other rules use for the TOC frame.

```
3346 *.topic_color  {  color: green; }
3347 div.bigspace   { margin-top:3em; margin-bottom:3em; }
3348 div.entete     { text-align:center; font-weight:bold; }
3349 div.tdmdiv     { position:fixed;top:0;left:0;clear:both;
3350   background-color:#dddddd;width:100%; overflow:scroll;max-height: 100% }
3351 div.noframe    { width:20%; }
3352 div.tdmdiv + div + div#main        { margin-left:21%; }
3353 div.tdmdiv + #toplign              { margin-left:20%; width:80%; }
3354 *.non-topic  { margin-bottom:-1em; padding-top:0em;
3355     padding-right:1em;  padding-bottom:0em;}
3356 *.topic      { border-width:thin ; border-color:black;
3357                border-style:none solid solid solid; }
3358 *.topicIdent { color:green; text-align:right ; }
3359 div.topic {background-color:#BCBCF9; color: #FFFFFF; padding: 5px 5px 0px 0px;}
3360 table.topic  { background-color:#BCBCF9;  color: #FFFFFF;}
```

More rules.

```
3361 #bouton  { padding:0em  }
3362 .NavigationIcones {margin:4px 0px 0px 0px; }
3363 #recherche { float:right }
3364 #toclink {  display:none }
3365 div.aucentre p   {  text-align:center; }
3366 p { }
```

These are unused.

```
3367 #corpsdroit a:hover div {  color: #fff; background-color: #369; text-decoration: none;}
3368 #bandeau_titre { height: 90px; border : 1px; }
3369 #annuel { float: left; margin-left:150px; height: 90px; width: 329px;
3370    padding: 0px; background: url(icons/pochette.png); }
3371 *.tdmActPage { background-color:red;  }
3372 div.boutonf  { font-size : smaller; color : #FFFFFF; background-color:#BCBCF9; }
```

Chapter 7

Converting XML to XSL/Format

In this chapter, we consider conversion of XML to XSL/Format, using the initial Raweb DTD, and we shall explain the difference with the new DTD. The files described in this chapter are part of the Tralics distribution (those for the new DTD are in the Raweb package). All these files are adapted from the TEI distribution; the copyright notice is the same as those given in the previous chapter.

7.1 The rrrafo3.xsl file

We start with a small file containing some commands that deal with fonts. In the original version of Tralics, the translation of `{\tiny etc}` was a `<hi>` element, with an attribute rend = 'small'. In the current version, Tralics understands more than three sizes (small, large, and normal), and can indicate these in an element, rather than an attribute. Thus, the translation can be `<font-small4>`. In this file, we have a big template, that interprets all values of the rend attribute, and a lot of trivial, small templates, that interpret elements like `<small4>`.

This interprets the rend attribute. The effect is to add attributes to the current element, in general `<fo:inline>`. In the initial version, the template took two arguments used in case of unknown specification (the default is to use bold font weight). Note: Perhaps, we should replace the value of 'small' by 9pt instead of 8pt.

```
<xsl:template name="rend">
  <xsl:choose>
    <xsl:when test="@rend='overline'">
      <xsl:attribute name="text-decoration">overline</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='underline'">
      <xsl:attribute name="text-decoration">underline</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='oldstyle'">
      <xsl:attribute name="font-family">Concrete</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='ital'">
      <xsl:attribute name="font-style">italic</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='emph'">
      <xsl:attribute name="font-style">italic</xsl:attribute>
    </xsl:when>
    <xsl:when test="@rend='sub'">
      <xsl:attribute name="vertical-align">sub</xsl:attribute>
```

```
20      </xsl:when>
21      <xsl:when test="@rend='sup'">
22        <xsl:attribute name="vertical-align">super</xsl:attribute>
23      </xsl:when>
24      <xsl:when test="@rend='it'">
25        <xsl:attribute name="font-style">italic</xsl:attribute>
26      </xsl:when>
27      <xsl:when test="@rend='slanted'">
28        <xsl:attribute name="font-style">italic</xsl:attribute>
29      </xsl:when>
30      <xsl:when test="@rend='sc'">
31        <xsl:attribute name="font-variant">small-caps</xsl:attribute>
32      </xsl:when>
33      <xsl:when test="@rend='tt'">
34        <xsl:attribute name="font-family">Computer-Modern-Typewriter</xsl:attribute>
35      </xsl:when>
36      <xsl:when test="@rend='sansserif'">
37        <xsl:attribute name="font-family">sansserif</xsl:attribute>
38      </xsl:when>
39      <xsl:when test="@rend='bold'">
40        <xsl:attribute name="font-weight">bold</xsl:attribute>
41      </xsl:when>
42      <xsl:when test="@rend='ul'">
43        <xsl:attribute name="text-decoration">ul</xsl:attribute>
44      </xsl:when>
45      <xsl:when test="@rend='caps'">
46        <xsl:attribute name="text-decoration">caps</xsl:attribute>
47      </xsl:when>
48      <xsl:when test="@rend='hl'">
49        <xsl:attribute name="text-decoration">hl</xsl:attribute>
50      </xsl:when>
51      <xsl:when test="@rend='so'">
52        <xsl:attribute name="text-decoration">so</xsl:attribute>
53      </xsl:when>
54      <xsl:when test="@rend='st'">
55        <xsl:attribute name="text-decoration">st</xsl:attribute>
56      </xsl:when>
57      <xsl:when test="@rend='small'">
58        <xsl:attribute name="font-size">8pt</xsl:attribute>
59      </xsl:when>
60      <xsl:when test="@rend='small1'">
61        <xsl:attribute name="font-size">9pt</xsl:attribute>
62      </xsl:when>
63      <xsl:when test="@rend='small2'">
64        <xsl:attribute name="font-size">8pt</xsl:attribute>
65      </xsl:when>
66      <xsl:when test="@rend='small3'">
67        <xsl:attribute name="font-size">7pt</xsl:attribute>
68      </xsl:when>
69      <xsl:when test="@rend='small4'">
70        <xsl:attribute name="font-size">5pt</xsl:attribute>
71      </xsl:when>
72      <xsl:when test="@rend='large'">
73        <xsl:attribute name="font-size">12pt</xsl:attribute>
74      </xsl:when>
75      <xsl:when test="@rend='large1'">
```

```
76        <xsl:attribute name="font-size">12pt</xsl:attribute>
77      </xsl:when>
78      <xsl:when test="@rend='large2'">
79        <xsl:attribute name="font-size">14pt</xsl:attribute>
80      </xsl:when>
81      <xsl:when test="@rend='large3'">
82        <xsl:attribute name="font-size">17pt</xsl:attribute>
83      </xsl:when>
84      <xsl:when test="@rend='large4'">
85        <xsl:attribute name="font-size">20pt</xsl:attribute>
86      </xsl:when>
87      <xsl:when test="@rend='large5'">
88        <xsl:attribute name="font-size">25pt</xsl:attribute>
89      </xsl:when>
90      <xsl:otherwise>
91        <xsl:attribute name="font-weight">bold</xsl:attribute>
92      </xsl:otherwise>
93    </xsl:choose>
94  </xsl:template>
```

Replacement code for the rend attribute defined above. In all cases, we provide a long and a short name. We produce an `<fo:inline>` element, with some attributes. We start with two templates associated to `<small>` and `<large>`. The attribute we set is font-size.

```
95  <xsl:template match='small|font-small'>
96    <fo:inline font-size='8pt'> <xsl:apply-templates/> </fo:inline>
97  </xsl:template>
```

Case large:

```
98  <xsl:template match='large|font-large'>
99    <fo:inline font-size='12pt'> <xsl:apply-templates/> </fo:inline>
100 </xsl:template>
```

We continue with the ten font size commands of LaTeX. This corresponds to `\tiny`.

```
101 <xsl:template match='small4|font-small4'>
102   <fo:inline font-size='5pt'> <xsl:apply-templates/> </fo:inline>
103 </xsl:template>
```

This corresponds to `\scriptsize`.

```
104 <xsl:template match='small3|font-small3'>
105   <fo:inline font-size='7pt'> <xsl:apply-templates/> </fo:inline>
106 </xsl:template>
```

This corresponds to `\footnotesize`.

```
107 <xsl:template match='small2|font_small2'>
108   <fo:inline font-size='8pt'> <xsl:apply-templates/> </fo:inline>
109 </xsl:template>
```

This corresponds to `\small`.

```
110 <xsl:template match='small1|font-small1'>
111   <fo:inline font-size='9pt'> <xsl:apply-templates/> </fo:inline>
112 </xsl:template>
```

This corresponds to `\normalsize`.

```
113 <xsl:template match='normalsize|font-normalsize'>
114   <fo:inline font-size='10pt'> <xsl:apply-templates/> </fo:inline>
115 </xsl:template>
```

This corresponds to `\large`.

```
<xsl:template match='large1|font-large1'>
  <fo:inline font-size='12pt'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

This corresponds to \Large.

```
<xsl:template match='large2|font-large2'>
  <fo:inline font-size='14.4pt'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

This corresponds to \LARGE.

```
<xsl:template match='large3|font-large3'>
  <fo:inline font-size='17.28pt'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

This corresponds to \huge.

```
<xsl:template match='large4|font-large4'>
  <fo:inline font-size='20.74pt'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

This corresponds to \Huge.

```
<xsl:template match='large5|font-large5'>
  <fo:inline font-size='24.88pt'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Now the four shapes. The attribute is font-style or font-variant. This corresponds to \itshape.

```
<xsl:template match='it|font-italic-shape'>
  <fo:inline font-style='italic'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

This corresponds to \slshape.

```
<xsl:template match='slanted|font-slanted-shape'>
  <fo:inline font-style='oblique'> <xsl:apply-templates/>  </fo:inline>
</xsl:template>
```

This corresponds to \scshape.

```
<xsl:template match='sc|font-small-caps-shape'>
  <fo:inline font-variant='small-caps'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

This corresponds to \upshape.

```
<xsl:template match='upright|font-upright-shape'>
  <fo:inline font-variant='normal'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Now the three families. The attribute is font-variant or font-family. We provide two different tt families. This corresponds to \rmfamily.

```
<xsl:template match='roman|font-roman-family'>
  <fo:inline font-variant='normal'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

This corresponds to \sfamily.

```
<xsl:template match='sansserif|font-sansserif-family'>
  <fo:inline font-family='sansserif'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

This corresponds to \ttfamily (first variant).

```
<xsl:template match='computer-modern-tt'>
  <fo:inline font-family='Computer-Modern-Typewriter'>
    <xsl:apply-templates/>
```

```
  </fo:inline>
</xsl:template>
```

This corresponds to `\ttfamily` (second variant).

```
<xsl:template match='tt|font-typewriter-family'>
  <fo:inline font-family='monospace'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Now the two series. The attribute is font-weight. This corresponds to `\mdseries`.

```
<xsl:template match='medium|font-medium-series'>
  <fo:inline font-weight='medium'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

This corresponds to `\bfseries`.

```
<xsl:template match='bold|font-bold-series'>
  <fo:inline font-weight='bold'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Superscript, subscript in text fonts. The attribute is vertical-align.

```
<xsl:template match='sup|font-super'>
  <fo:inline vertical-align='super'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Case of a subscript (corresponds to a non-standard LaTeX command).

```
<xsl:template match='sub|font-sub'>
  <fo:inline vertical-align='sub'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Overline, underline in text fonts. The attribute is text-decoration.

```
<xsl:template match='underline|font-underline'>
  <fo:inline text-decoration='underline'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Case of overline.

```
<xsl:template match='overline|font-overline'>
  <fo:inline text-decoration='overline'> <xsl:apply-templates/> /fo:inline>
</xsl:template>
```

Five elements defined by the soul package. Only the long variant is considered here. The result is a text-decoration.

```
<xsl:template match='font-ul'>
  <fo:inline text-decoration='ul'> <xsl:apply-templates/>  </fo:inline>
</xsl:template>
<xsl:template match='font-caps'>
  <fo:inline text-decoration='caps'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
<xsl:template match='font-hl'>
  <fo:inline text-decoration='hl'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
<xsl:template match='font-so'>
  <fo:inline text-decoration='so'>  <xsl:apply-templates/>  </fo:inline>
</xsl:template>
<xsl:template match='font-st'>
  <fo:inline text-decoration='st'> <xsl:apply-templates/> </fo:inline>
</xsl:template>
```

Translation of `<note>`. We simplified a bit the code. In the TEI, there is an attribute place, whose value can be 'foot' if the note should appear on the foot of the page, it can be 'margin', 'left', 'right' if the note appears in the margin (or more specifically in the left or right margin), but

implementation is incomplete, if can be 'display' or 'divtop' if the note should appear in the text as a block; otherwise it will be inlined (with parentheses around).

In the case of the Raweb, we consider only the case where the note is a footnote. We first compute the number, and construct an inline element E, containing this number in a small font, vertically aligned as a superscript. We construct an element B, a `<fo:footnote-body>` containing this B followed by the content of the note. The result is then `<fo:footnote>` element, containing E and B.

```
<xsl:template match="note">
  <xsl:variable name="FootID">
     <xsl:call-template name="calculateFootnoteNumber"/>
  </xsl:variable>
  <fo:footnote>
    <fo:inline font-size="{$footnotenumSize}" vertical-align="super">
      <xsl:value-of select="$FootID"/>
    </fo:inline>
    <fo:footnote-body>
      <fo:block end-indent="0pt" start-indent="0pt"
             text-indent="{$parIndent}" font-size="{$footnoteSize}">
        <fo:inline font-size="{$footnotenumSize}" vertical-align="super">
          <xsl:value-of select="$FootID"/>
        </fo:inline>
        <xsl:apply-templates/>
      </fo:block>
    </fo:footnote-body>
  </fo:footnote>
</xsl:template>
```

This is the end of the file.

```
</xsl:stylesheet>
```

In the new DTD, there is no `<hi>` element anymore. Instead, element `<b>` selects a bold font weight, element `<i>` selects an italic font style, element `<tt>` selects type writer font family, elements `<sup>` and `<sub>` select some vertical alignment, elements `<small>` and `<big>` select a font size, while `<span>` selects a small caps font variant.

```
<xsl:template match="b">
  <fo:inline font-weight="bold"> <xsl:apply-templates /> </fo:inline>
</xsl:template>

<xsl:template match="i">
  <fo:inline font-style="italic"> <xsl:apply-templates /> </fo:inline>
</xsl:template>

<xsl:template match="tt">
  <fo:inline font-family='Computer-Modern-Typewriter'>
    <xsl:apply-templates />  </fo:inline>
</xsl:template>

<xsl:template match="sup">
  <fo:inline vertical-align="super"> <xsl:apply-templates /> </fo:inline>
</xsl:template>

<xsl:template match="sub">
  <fo:inline vertical-align="sub">  <xsl:apply-templates /> </fo:inline>
</xsl:template>

<xsl:template match="small">
```

```
  <fo:inline font-size="8pt"> <xsl:apply-templates /> </fo:inline>
</xsl:template>

<xsl:template match="span">
  <fo:inline font-variant='small-caps'> <xsl:apply-templates /> </fo:inline>
</xsl:template>

<xsl:template match="big">
  <fo:inline font-size="12pt"> <xsl:apply-templates /> </fo:inline>
</xsl:template>
```

Finally, the style of a is interpreted by this trivial procedure. It adds an attribute pair to the current element. The default behavior is bold font weight.

```
<xsl:template name="style">
  <xsl:param name="defaultvalue"/>
  <xsl:param name="defaultstyle"/>
  <xsl:choose>
    <xsl:when test="@style='UNDERLINE'">
      <xsl:attribute name="text-decoration">underline</xsl:attribute>
    </xsl:when>
    <xsl:when test="@style='overline'">
      <xsl:attribute name="text-decoration">overline</xsl:attribute>
    </xsl:when>
    <xsl:when test="@style='sansserif'">
      <xsl:attribute name="font-family">sansserif</xsl:attribute>
    </xsl:when>
   <xsl:when test="@style='slanted'">
      <xsl:attribute name="font-style">oblique</xsl:attribute>
    </xsl:when>
    <xsl:when test="@style='HIGHLIGHT'">
      <xsl:attribute name="font-style">cursive</xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
      <xsl:attribute name="{$defaultstyle}">
        <xsl:value-of select="$defaultvalue"/>
      </xsl:attribute>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

7.2 The rawebfo file

This is the main file, its name is raweb3fo.xsl. It starts like this.

```
<xsl:stylesheet
  xmlns:fotex="http://www.tug.org/fotex"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:m="http://www.w3.org/1998/Math/MathML"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

We overwrite some parameters defined in the TEI file.

```
<xsl:param name="linkColor">red</xsl:param>
<xsl:param name="pageMarginRight">80pt</xsl:param>
<xsl:param name="pdfBookmarks"></xsl:param>
```

We include these two files. The content of the first one will be shown at the end of the chapter (section 7.17), the other was the purpose of the preceding section.

```
<xsl:include href="raweb3-param.xsl"/>
<xsl:include href="rrrafo3.xsl"/>
```

There are some elements for which white space is ignored.

```
<xsl:strip-space elements="cell bediteur bauteurs citation UR"/>
<xsl:output indent="no"/>
```

We do not use table specifications. Thus this code is a simplification of the original.

```
<xsl:variable name="top" select="/"/>
<xsl:variable name="tableSpecs">
  <Info></Info>
</xsl:variable>
```

In the case of unknown elements, we put the name in a comment, and we handle the content.

```
<xsl:template match="*">
  <xsl:comment><xsl:text>PASS THROUGH </xsl:text>
     <xsl:value-of select="name()"/>
  </xsl:comment>
  <xsl:apply-templates/>
</xsl:template>
```

7.3 Page definitions

The first object in the XSL/Format file is the definition of the page layout. In our case, half of the definitions given here are not used, but are taken from the TEI code. Maybe one day somebody wants an index in two columns, in this case, page masters with index 2 can be used. Before 2006, there is no index mechanism offered by Tralics, and the `\index` command is forbidden in the Raweb.

The `<fo:layout-master-set>` element contains first some `<fo:simple-page-master>` elements. All these use the value of `$pageMarginTop` for the top margin, `$pageMarginBottom` for the bottom margin, `$pageMarginLeft` for the left margin, `$pageMarginRight` for the right margin, `$pageWidth` for the width of the page, and `$pageHeight` for the height of the page (values omitted but for the first occurrence). In any case, we define three regions, `<fo:region-body>`, `<fo:region-before>`, and `<fo:region-after>`. These last two regions have always the same extent, stored in variables `$regionBeforeExtent` and `$regionAfterExtent`.

We start with simple1: no headings, one column, all pages have the same look. The regions before and after have no name.

```
<xsl:template name="setupPagemasters">
  <fo:layout-master-set>
     <fo:simple-page-master master-name="simple1"
        page-width="{$pageWidth}"
        page-height="{$pageHeight}"
        margin-top="{$pageMarginTop}"
        margin-bottom="{$pageMarginBottom}"
        margin-left="{$pageMarginLeft}"
        margin-right="{$pageMarginRight}">
        <fo:region-body
                margin-bottom="{$bodyMarginBottom}"
```

```
301             margin-top="{$bodyMarginTop}"/>
302       <fo:region-before extent="{$regionBeforeExtent}"/>
303       <fo:region-after extent="{$regionAfterExtent}"/>
304     </fo:simple-page-master>
```

This is for left-hand/even pages in twosided mode, single column.

```
305     <fo:simple-page-master master-name="left1" >
306       <!-- Attributes as above -->
307       <fo:region-body
308             margin-bottom="{$bodyMarginBottom}"
309             margin-top="{$bodyMarginTop}"/>
310       <fo:region-before
311             region-name="xsl-region-before-left"
312             extent="{$regionBeforeExtent}"/>
313       <fo:region-after
314             region-name="xsl-region-after-left"
315             extent="{$regionAfterExtent}"/>
316     </fo:simple-page-master>
```

Case of right-hand/odd pages in twosided mode, single column.

```
317     <fo:simple-page-master  master-name="right1" >
318       <!-- Attributes as above -->
319       <fo:region-body
320             margin-bottom="{$bodyMarginBottom}"
321             margin-top="{$bodyMarginTop}"/>
322       <fo:region-before
323             region-name="xsl-region-before-right"
324             extent="{$regionBeforeExtent}"/>
325       <fo:region-after
326             region-name="xsl-region-after-right"
327             extent="{$regionAfterExtent}"/>
328     </fo:simple-page-master>
```

Special case of first page in either mode, single column.

```
329     <fo:simple-page-master master-name="first1">
330       <!-- Attributes as above -->
331       <fo:region-body
332             margin-bottom="{$bodyMarginBottom}"
333             margin-top="{$bodyMarginTop}"/>
334       <fo:region-before
335             region-name="xsl-region-before-first"
336             extent="{$regionBeforeExtent}"/>
337       <fo:region-after
338             region-name="xsl-region-after-first"
339             extent="{$regionAfterExtent}"/>
340     </fo:simple-page-master>
```

Case of a blank page. No headings here.

```
341     <fo:simple-page-master master-name="blank1">
342       <!-- Attributes as above -->
343       <fo:region-body
344             margin-bottom="{$bodyMarginBottom}"
345             margin-top="{$bodyMarginTop}"/>
346       <fo:region-before
347             region-name="DummyRegion"
348             extent="{$regionBeforeExtent}"/>
349       <fo:region-after
350             region-name="DummyRegion"
```

```
351                 extent="{$regionAfterExtent}"/>
352       </fo:simple-page-master>
```

For pages in one-side mode, 2 columns per page. We use the value of `$columnCount` for the number of columns.

```
353       <fo:simple-page-master master-name="simple2">
354          <!-- Attributes as above -->
355          <fo:region-body
356                  column-count="{$columnCount}"
357                  margin-bottom="{$bodyMarginBottom}"
358                  margin-top="{$bodyMarginTop}"/>
359          <fo:region-before extent="{$regionBeforeExtent}"/>
360          <fo:region-after extent="{$regionAfterExtent}"/>
361        </fo:simple-page-master>
```

For left-hand/even pages in twosided mode, 2 columns per page.

```
362       <fo:simple-page-master master-name="left2">
363          <!-- Attributes as above -->
364          <fo:region-body
365                  column-count="{$columnCount}"
366                  margin-bottom="{$bodyMarginBottom}"
367                  margin-top="{$bodyMarginTop}"/>
368          <fo:region-before
369                  region-name="xsl-region-before-left"
370                  extent="{$regionBeforeExtent}"/>
371          <fo:region-after
372                  region-name="xsl-region-after-left"
373                  extent="{$regionAfterExtent}"/>
374        </fo:simple-page-master>
```

For right-hand/odd pages in twosided mode, 2 columns per page.

```
375        <fo:simple-page-master master-name="right2">
376          <!-- Attributes as above -->
377          <fo:region-body
378                  column-count="{$columnCount}"
379                  margin-bottom="{$bodyMarginBottom}"
380                  margin-top="{$bodyMarginTop}"/>
381          <fo:region-before
382                  region-name="xsl-region-before-right"
383                  extent="{$regionBeforeExtent}"/>
384          <fo:region-after
385                  region-name="xsl-region-after-right"
386                  extent="{$regionAfterExtent}"/>
387       </fo:simple-page-master>
```

Special case of first page in either mode, two columns per page.

```
388        <fo:simple-page-master master-name="first2">
389          <!-- Attributes as above -->
390          <fo:region-body
391                  column-count="{$columnCount}"
392                  margin-bottom="{$bodyMarginBottom}"
393                  margin-top="{$bodyMarginTop}"/>
394          <fo:region-before
395                  region-name="xsl-region-before-first"
396                  extent="{$regionBeforeExtent}"/>
397          <fo:region-after
398                  region-name="xsl-region-after-first"
```

```
399               extent="{$regionAfterExtent}"/>
400     </fo:simple-page-master>
```

We define now some `<fo:page-sequence-master>` elements.
They contain a `<fo:repeatable-page-master-alternatives>` element. These contain some `<fo:conditional-page-master-reference>`. We start with setup for double-sided, 1 column, no first page.

```
401     <fo:page-sequence-master master-name="twoside1nofirst">
402      <fo:repeatable-page-master-alternatives>
403        <fo:conditional-page-master-reference
404               master-reference="right1"
405               odd-or-even="odd"/>
406        <fo:conditional-page-master-reference
407               master-reference="left1"
408               odd-or-even="even"/>
409      </fo:repeatable-page-master-alternatives>
410     </fo:page-sequence-master>
```

Setup for double-sided, 1 column. Note that this is the only one that defines a page master for blank pages.

```
411     <fo:page-sequence-master master-name="twoside1">
412      <fo:repeatable-page-master-alternatives>
413        <fo:conditional-page-master-reference
414               master-reference="first1"
415               page-position="first"/>
416        <fo:conditional-page-master-reference
417               master-reference="right1"
418               odd-or-even="odd"/>
419        <fo:conditional-page-master-reference
420              master-reference="left1"
421              odd-or-even="even"/>
422        <fo:conditional-page-master-reference
423               master-reference="blank1"
424               blank-or-not-blank="blank"/>
425      </fo:repeatable-page-master-alternatives>
426     </fo:page-sequence-master>
```

Setup for single-sided, 1 column.

```
427     <fo:page-sequence-master  master-name="oneside1">
428      <fo:repeatable-page-master-alternatives>
429        <fo:conditional-page-master-reference
430               master-reference="first1"
431               page-position="first"/>
432        <fo:conditional-page-master-reference master-reference="simple1"/>
433      </fo:repeatable-page-master-alternatives>
434     </fo:page-sequence-master>
```

Setup for double-sided, 2 columns.

```
435     <fo:page-sequence-master master-name="twoside2">
436      <fo:repeatable-page-master-alternatives>
437        <fo:conditional-page-master-reference
438               master-reference="first2"
439               page-position="first"/>
440        <fo:conditional-page-master-reference
441               master-reference="right2"
442               odd-or-even="odd"/>
443        <fo:conditional-page-master-reference
444               master-reference="left2"
```

```
                odd-or-even="even"/>
      </fo:repeatable-page-master-alternatives>
     </fo:page-sequence-master>
```

Setup for single-sided, 2 columns.

```
     <fo:page-sequence-master  master-name="oneside2">
      <fo:repeatable-page-master-alternatives>
        <fo:conditional-page-master-reference
                master-reference="first2"
                page-position="first"/>
        <fo:conditional-page-master-reference master-reference="simple2" />
      </fo:repeatable-page-master-alternatives>
     </fo:page-sequence-master>
    <xsl:call-template name="hookDefinepagemasters"/>
```

That was a long template! The XSLT processor replaces all the space characters and newline characters by a single space. The resulting element is printed on a single line; it has over 5000 characters. This is much larger than the 500 that appear in the TEX source; in fact, in the case of the apics Team, TEX used 15174 input buffer positions.

```
    </fo:layout-master-set>
</xsl:template>
```

We define here two `<fo:static-content>` elements, for left and right pages. They contain a `<fo:block>` and a second block (is this really needed?) The inner block says text-indent = '0pt', because headings should not be indented. We set border-after-style to solid. In the original version, this did not work, and there was a `<pagestylehrule>` instead. On one end we have page numbers, on the other we have the name of the team or INRIA.

```
<xsl:template name="myheaders">
 <fo:static-content flow-name="xsl-region-before-right">
  <fo:block  text-align="justify" font-size="{$bodySize}">
    <fo:block border-after-style="solid" text-indent="0pt">
       <fo:inline font-style="italic"><xsl:value-of select="$PRID"/></fo:inline>
       <fo:leader rule-thickness="0pt"/>
       <fo:inline> <fo:page-number/> </fo:inline>
    </fo:block>
  </fo:block>
 </fo:static-content>

 <fo:static-content flow-name="xsl-region-before-left">
  <fo:block text-align="justify" font-size="{$bodySize}">
    <fo:block border-after-style="solid" text-indent="0pt">
       <fo:inline>  <fo:page-number/> </fo:inline>
       <fo:leader rule-thickness="0pt"/>
       <fo:inline font-style="italic">
              Activity Report INRIA <xsl:value-of select="$year"/>
       </fo:inline>
    </fo:block>
  </fo:block>
 </fo:static-content>
```

We use empty footers, and the first page has no header.

```
 <fo:static-content flow-name="xsl-region-before-first"/>
 <fo:static-content flow-name="xsl-region-after-right"/>
 <fo:static-content flow-name="xsl-region-after-left"/>
```

```
 <fo:static-content flow-name="xsl-region-after-first"/>
</xsl:template>
```

The main text is in a page sequence, defined like this. Note that the original file had master-name = 'twoside1nofirst'. This is useless, but raises the following question: should the first page of text (in general, the fifth page) have a page header, or should it be considered as a first page. The flow-name attribute is useless (used only for static content).

```
<xsl:template name="maintext">
  <fo:page-sequence
        format="1"
        text-align="justify"
        hyphenate="true"
        language="english"
        initial-page-number="1"
        master-reference="twoside1"
      >
      <fo:flow
         flow-name="xsl-region-body"
         font-family="{$bodyFont}"
         font-size="{$bodySize}">
        <xsl:call-template name="raweb.body"/>
      </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

The title page is another page sequence, defined like this. The content will be given later; in the new DTD, this is called <identification>.

```
<xsl:template match="accueil">
  <fo:page-sequence
        format="1"
        text-align="justify"
        hyphenate="true"
        language="english"
        initial-page-number="1"
        master-reference="twoside1"
        force-page-count="end-on-even"
      >
    <fo:flow font-style="italic" font-family="{$bodyFont}">
      <xsl:call-template  name="accueil.body"/>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

The TOC is similar.

```
<xsl:template name="myTOC">
  <fo:page-sequence
        format="1"
        text-align="justify"
        hyphenate="true"
        language="english"
        initial-page-number="1"
        master-reference="twoside1"
        force-page-count="end-on-even"
```

```
      >
    <fo:flow flow-name="xsl-region-body">
      <fo:block>
        <xsl:call-template name="toc.body"/>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

7.4 The text

The document element is `<raweb>`. The first thing to do is construct the `<fo:layout-master-set>`, then all `<fo:static-content>`. After that, we have three parts: the title page, the table of contents, and the main text. Each of these parts start on a right page (an odd one, but the first page in each section is numbered one). Using `<cleardoublepage>` is a big hack.[1]

```
<xsl:template match="raweb">
 <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <xsl:call-template name="setupPagemasters"/>
   <xsl:call-template name="myheaders"/>
   <xsl:apply-templates select="accueil"/>
   <xsl:call-template name="myTOC"/>
   <xsl:call-template name="maintext"/>
 </fo:root>
</xsl:template>
```

The `maintext` template selects a page sequence and a flow (see above), then calls this template. The idea is trivial: we consider one section after the other.

```
<xsl:template name="raweb.body">
  <xsl:apply-templates select="composition"/>
  <xsl:apply-templates select="presentation"/>
  <xsl:apply-templates select="fondements"/>
  <xsl:apply-templates select="domaine"/>
  <xsl:apply-templates select="logiciels"/>
  <xsl:apply-templates select="resultats"/>
  <xsl:apply-templates select="contrats"/>
  <xsl:apply-templates select="international"/>
  <xsl:apply-templates select="diffusion"/>
  <xsl:apply-templates select="biblio"/>
</xsl:template>
```

This computes the start of a section. The result is a `<fo:block>`, containing the number and title of the section; these are found in attributes. This can call two templates that can parameterize the code. Note: there is a problem with the bookmarks. For this reason, we do not show the code.

```
<xsl:template name="secNumberedHeading">
  <fo:block keep-with-next.within-page="always">
    <xsl:variable name="divid">
       xsl:call-template name="idLabel"/>
    </xsl:variable>
    <xsl:attribute name="id"> <xsl:value-of select="$divid"/> </xsl:attribute>
```

[1] It is removed in the 2005 version shown here.

```
562     <xsl:attribute name="text-align">start</xsl:attribute>
563     <xsl:attribute name="font-family">
564       <xsl:value-of select="$divFont"/>
565     </xsl:attribute>
566     <xsl:call-template name="setupDiv0"/>
567     <xsl:call-template name="blockStartHook"/>
568     <xsl:value-of select="@numero"/>
569     <xsl:text>. </xsl:text>
570     <xsl:value-of select="@titre"/>
571     <xsl:if test="$pdfBookmarks='true'">
572       <fotex:bookmark ... />
573     </xsl:if>
574   </fo:block>
575 </xsl:template>
```

The code changed a bit in the case of the new DTD. This is because the title of the section is now in `<bodyTitle>`, unless the section is `<identification>`, `<team>`, `<biblio>`, case where 'Identification', 'Team' or 'Bibliography' is used. Moreover, the numero attribute is not used. Instead, the effective section number is used instead, via `calculateNumberSection`. These lines are a replacement for lines 568 to 571.

```
576     <xsl:call-template name="calculateNumberSection"/>
577     <xsl:if test="parent::raweb">
578       <xsl:text> </xsl:text>
579     </xsl:if>
580     <xsl:choose>
581       <xsl:when test="name()='identification'">Identification</xsl:when>
582       <xsl:when test="name()='team'"><xsl:text> </xsl:text> Team</xsl:when>
583       <xsl:when test="name()='biblio'">Bibliography</xsl:when>
584       <xsl:otherwise>
585         <xsl:value-of select="bodyTitle"/>
586       </xsl:otherwise>
587     </xsl:choose>
```

This returns an id. Question? do we really need to replace underscores by hyphens?

```
588 <xsl:template name="idLabel">
589  <xsl:choose>
590    <xsl:when test="@id"><xsl:value-of select="translate(@id,'_','-')"/> </xsl:when>
591    <xsl:otherwise><xsl:value-of select="generate-id()"/></xsl:otherwise>
592  </xsl:choose>
593 </xsl:template>
```

This is for a subsection. The title is in `<head>` (and `<bodyTitle>` in the new DTD). The number has to be computed.

```
594 <xsl:template name="NumberedHeading">
595   <xsl:param name="level"/>
596   <fo:block keep-with-next.within-page="always">
597     <xsl:variable name="divid">
598        <xsl:call-template name="idLabel"/>
599     </xsl:variable>
600     <xsl:attribute name="id">
601       <xsl:value-of select="$divid"/>
602     </xsl:attribute>
603     <xsl:attribute name="text-align">start</xsl:attribute>
```

```
604     <xsl:attribute name="font-family">
605        <xsl:value-of select="$divFont"/>
606     </xsl:attribute>
607     <xsl:choose>
608      <xsl:when test="$level=0"><xsl:call-template name="setupDiv0"/></xsl:when>
609      <xsl:when test="$level=1"><xsl:call-template name="setupDiv1"/></xsl:when>
610      <xsl:when test="$level=2"><xsl:call-template name="setupDiv2"/></xsl:when>
611      <xsl:when test="$level=3"><xsl:call-template name="setupDiv3"/></xsl:when>
612      <xsl:when test="$level=4"><xsl:call-template name="setupDiv4"/></xsl:when>
613     </xsl:choose>
614     <xsl:call-template name="blockStartHook"/>
615     <xsl:variable name="Number">
616       <xsl:if test="$numberHeadings and $numberDepth &gt; $level">
617         <xsl:call-template name="calculateNumber">
618           <xsl:with-param name="numbersuffix" select="$headingNumberSuffix"/>
619         </xsl:call-template>
620       </xsl:if>
621     </xsl:variable>
622     <xsl:value-of select="$Number"/>
623     <xsl:apply-templates mode="section" select="head"/>
624     <xsl:if test="$pdfBookmarks='true'"> <fotex:toto/>
625       <fotex:bookmark   ... />
626     </xsl:if>
627   </fo:block>
628 </xsl:template>
```

The translation of the `<composition>` section is trivial. We add the toplevel moreinfo before the text.

```
629 <xsl:template match="composition">
630    <xsl:call-template name="secNumberedHeading"/>
631    <xsl:apply-templates select="/raweb/moreinfo"/>
632    <xsl:apply-templates/>
633 </xsl:template>
```

In the new DTD, the previous piece of code is to be replaced by this one. Note that the toplevel moreinfo is a child of `<identification>`, and the (optional) moreinfo that is a child of `<team>` is lost.

```
634 <xsl:template match="team">
635  <xsl:call-template name="secNumberedHeading"/>
636  <xsl:if test="../moreinfo"><xsl:apply-templates select="../moreinfo"/></xsl:if>
637  <xsl:apply-templates select="participants"/>
638 </xsl:template>
```

Other sections. We emit a title, then the content of the element. In the new DTD, the final apply-templates selects all nodes but `<bodyTitle>`.

```
639 <xsl:template
640     match="presentation|fondements|domaine|logiciels|resultats|contrats
641       |international|diffusion">
642    <xsl:call-template name="secNumberedHeading"/>
643    <xsl:apply-templates/>
644 </xsl:template>
```

Modules. This is a subsection. The title is in `<head>`. In the case of dummy titles, nothing is printed.

```
<xsl:template match="module">
 <xsl:if test="./head!='(Sans Titre)'">
  <xsl:call-template name="NumberedHeading">
    <xsl:with-param name="level" select="'1'"/>
   </xsl:call-template>
 </xsl:if>
 <xsl:apply-templates/>
</xsl:template>
```

A section is equivalent to `<div0>`, a module to `<div1>`. We consider here divisions of level 2, 3 and 4.

```
<xsl:template match="div2|div3|div4">
  <xsl:call-template name="NumberedHeading">
    <xsl:with-param name="level">
       <xsl:value-of select="substring-after(name(),'div')"/>
    </xsl:with-param>
  </xsl:call-template>
  <xsl:apply-templates/>
</xsl:template>
```

In the case of the new DTD, the two previous templates are replaced by a single one (for the element `<subsection>`). Of course, the `$level` has to be computed differently: it is the number of ancestors (or self) of type `<subsection>`. In the case of a section and subsection, the apply-templates is applied to all nodes whose name is not `<bodyTitle>`. Finally, in the case of a `<module>`, keywords are inserted via `keywords-list`, and for other divisions, they are inserted via `keywords-list2`.

7.5 The table of contents

The table of contents is formed of a a title, followed by all sections and subsections, translated in a special mode.

```
<xsl:template name="toc.body">
       <fo:block font-size="14pt" text-align="center" font-weight="bold"
           space-after="20pt">
         <xsl:text>Table of contents</xsl:text>
       </fo:block>
       <xsl:for-each select="//composition|//presentation|//fondements|
             //domaine|//logiciels|//resultats|//contrats|//international|
              //diffusion|//biblio|//module|//div2|//div3">
          <xsl:apply-templates mode="xtoc" select="(.)"/>
       </xsl:for-each>
</xsl:template>
```

Translation of a section in the TOC. The result is a bold line, containing the section number (computed), its title (from the attribute in the old DTD, computed in the new one), the page number, and a link to it.

```
<xsl:template mode="xtoc" match="composition|presentation|
    fondements|domaine|logiciels|resultats|contrats|international|
     diffusion|biblio">
  <xsl:variable name="tocindent">
    <xsl:value-of select="$div0Tocindent"/>
  </xsl:variable>
```

```
678   <fo:block>
679     <xsl:attribute name="font-weight">bold</xsl:attribute>
680     <xsl:attribute name="text-indent">
681       <xsl:value-of select="$tocindent"/>
682     </xsl:attribute>
683     <xsl:call-template name="sec.num"/>
684     <xsl:text> </xsl:text>
685     <fo:inline>
686       <xsl:value-of select="@titre"/>
687     </fo:inline>
688      <fo:leader leader-pattern="dots" />
689     <fo:inline color="{$linkColor}">
690       <xsl:variable name="pagref">
691         <xsl:call-template name="idLabel"/>
692       </xsl:variable>
693      <fo:basic-link internal-destination="{$pagref}">
694       <fo:page-number-citation ref-id="{$pagref}"/>
695      </fo:basic-link>
696     </fo:inline>
697   </fo:block>
698 </xsl:template>
```

A module in the TOC. If the title (from the `<head>` element) is empty, we do nothing, otherwise call some template that adds a line to the TOC. The parameter of the template is 1 (a module is equivalent to a `<div1>`).

```
699 <xsl:template mode="xtoc" match="module">
700  <xsl:if test="head!='(Sans Titre)'">
701    <xsl:call-template name="tocheading">
702     <xsl:with-param name="level">
703         <xsl:value-of select="1"/></xsl:with-param>
704    </xsl:call-template>
705  </xsl:if>
706 </xsl:template>
```

A division (`<div2>`, `<div3>`, or `<div4>`) in the TOC. Same as for a module, but the parameter of the template, the number, has to be computed. Moreover, an entry is inserted in the TOC even when the title is empty. For the new DTD, this template and the preceding one were replaced by a single template for `<subsection>`, the value of level parameter being the number of ancestors (or self) named `<subsection>`.

```
707 <xsl:template mode="xtoc" match="div2|div3|div4">
708    <xsl:call-template name="tocheading">
709     <xsl:with-param name="level">
710         <xsl:value-of select="substring-after(name(),'div')"/></xsl:with-param>
711    </xsl:call-template>
712 </xsl:template>
```

A division gives an entry in the TOC as follows. The result is a `<fo:block>`, with some indentation that depends on the level, plus a number, and a title (unless empty), leaders, then the page number, and a link.

```
713 <xsl:template name="tocheading">
714   <xsl:param name="level"/>
715   <xsl:variable name="tocindent">
716     <xsl:choose>
717       <xsl:when test="$level='0'">
```

```
718           <xsl:value-of select="$div0Tocindent"/></xsl:when>
719         <xsl:when test="$level='1'">
720           <xsl:value-of select="$div1Tocindent"/></xsl:when>
721         <xsl:when test="$level='2'">
722           <xsl:value-of select="$div2Tocindent"/></xsl:when>
723         <xsl:when test="$level='3'">
724           <xsl:value-of select="$div3Tocindent"/></xsl:when>
725         <xsl:when test="$level='4'">
726           <xsl:value-of select="$div4Tocindent"/></xsl:when>
727         <xsl:otherwise><xsl:value-of select="$div1Tocindent"/></xsl:otherwise>
728       </xsl:choose>
729     </xsl:variable>
730     <fo:block>
731       <xsl:attribute name="text-indent">
732         <xsl:value-of select="$tocindent"/>
733       </xsl:attribute>
734       <xsl:variable name="Number">
735         <xsl:if test="$numberHeadings and $numberDepth &gt; $level">
736           <xsl:call-template name="calculateNumber">
737             <xsl:with-param name="numbersuffix" select="$tocNumberSuffix"/>
738           </xsl:call-template>
739         </xsl:if>
740       </xsl:variable>
741       <xsl:value-of select="$Number"/>
742       <xsl:text> </xsl:text>
743       <xsl:if test="head!='(Sans Titre)'">
744         <fo:inline>
745           <xsl:apply-templates mode="tocsection" select="head"/>
746         </fo:inline>
747       </xsl:if>
748       <fo:leader rule-thickness="0pt"/>
749       <fo:inline color="{$linkColor}">
750         <xsl:variable name="pagref">
751           <xsl:call-template name="idLabel"/>
752         </xsl:variable>
753        <fo:basic-link internal-destination="{$pagref}">
754         <fo:page-number-citation ref-id="{$pagref}"/>
755        </fo:basic-link>
756       </fo:inline>
757     </fo:block>
758 </xsl:template>
```

A `<head>` in the TOC: the ID disappears (as well as all other attributes). Moreover, the content is evaluated in 'section' mode. All elements disappear (only text remains), unless specified otherwise.

```
759 <xsl:template match="head" mode="tocsection">
760   <xsl:apply-templates mode="section"/>
761 </xsl:template>
```

We interpret math elements in a title as usual (the title could be "Encoding π in π_{pa}", guess what happens without the math?)

```
762 <xsl:template match="m:math" mode="section">
763  <m:math>
764   <xsl:copy-of select="@*"/>
765   <xsl:apply-templates mode="math"/>
```

```
 </m:math>
</xsl:template>
```

A `<head>` in a section: the ID is in the result (why do we replace underscores by dashes?).

```
<xsl:template match="head" mode="section">
 <fo:inline>
  <xsl:if test=".!='(Sans Titre)'">
   <xsl:if test="@id">
    <xsl:attribute name="id">
     <xsl:value-of select="translate(@id,'_','-')"/>
    </xsl:attribute>
   </xsl:if>
   <xsl:apply-templates mode="section"/>
  </xsl:if>
 </fo:inline>
</xsl:template>
```

In the new DTD, lines 744–746 were replaced by the following, and the previous template was renamed `bodyTitle`. There is one problem: if the title has an ID, it will be inserted in the main text, and in the TOC, this is completely wrong (we are lucky, because a section has an ID, and the title of the section has none).

```
   <fo:inline space-start="20pt">
      <xsl:apply-templates mode="section" select="bodyTitle"/>
   </fo:inline>
```

7.6 The bibliography

Translation of the bibliography. There are eight optional parts. We consider them one after the other.

```
<xsl:template match="biblio">
   <xsl:call-template name="secNumberedHeading"/>
   <xsl:call-template name="biblioA"/>
   <xsl:call-template name='biblioBA'/>
   <xsl:call-template name='biblioBC'/>
   <xsl:call-template name='biblioBD'/>
   <xsl:call-template name='biblioBE'/>
   <xsl:call-template name='biblioBH'/>
   <xsl:call-template name='biblioBJ'/>
   <xsl:call-template name="biblioC"/>
</xsl:template>
```

This outputs the title of a subsection, for the bibliography. This title is in the `$name` parameter, the default value is never used.

```
<xsl:template name="biblioname">
  <xsl:param name="name">Unknown bibliography section</xsl:param>
  <fo:block font-weight='bold' font-size="14pt" space-before="5pt"
       keep-with-next='always'>
    <xsl:value-of select="$name"/>
  </fo:block>
</xsl:template>
```

Bibliography, part one. We select all entries that have a from attribute whose value is 'refer'. The title of the section is "Major publications by the team in recent years". If no entry matches, the translation is empty.

```
<xsl:template name="biblioA">
  <xsl:if test="citation[@from ='refer']">
    <xsl:call-template name="biblioname">
      <xsl:with-param name="name">  ... </xsl:with-param>
    </xsl:call-template>
    <xsl:for-each select="citation[@from ='refer']">
       <xsl:apply-templates select="(.)"/>
    </xsl:for-each>
  </xsl:if>
</xsl:template>
```

Bibliography, last part. We select all entries that have a from attribute whose value is 'foot'. The title of the section is "Bibliography in notes".

```
<xsl:template name="biblioC">
  <xsl:if test="citation[@from ='foot']">
   ...
  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is book or booklet or proceedings. The title of the section is "Books and Monographs".

```
<xsl:template name='biblioBA'>
 <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is phdthesis. The title of the section is "Doctoral dissertations and "Habilitation" theses".

```
<xsl:template name='biblioBC'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is article or inbook or incollection. The title of the section is "Articles in referred journals and book chapters".

```
<xsl:template name='biblioBD'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is inproceedings or conference. The title of the section is "Publications in Conferences and Workshops".

```
<xsl:template name='biblioBE'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is manual or techreport or coursenotes. The title of the section is "Internal Reports".

```
<xsl:template name='biblioBH'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

We select all entries that have a from attribute whose value is 'year' and type is unpublished or misc or masterthesis or mastersthesis. The title of the section is "Miscellaneous".

```
<xsl:template name='biblioBJ'>
  <xsl:if test="..."> ...  </xsl:if>
</xsl:template>
```

A `<citation>` produces a `<fo:block>`, with the same id. It contains the value of the key, in brackets, then the content of the element.

```
<xsl:template match="citation">
  <fo:block space-before="15pt" text-indent="-2em">
     <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
     [<xsl:value-of select="@key"/>]
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

This produces a comma between elements, a period at the end. A similar template rule that produces a comma, but no period, named `separateurED.objet` also exists.

```
<xsl:template name="separateur.objet">
 <xsl:choose>
  <xsl:when test="position()!=last()">, </xsl:when>
  <xsl:when test="position()=last()">.</xsl:when>
 </xsl:choose>
</xsl:template>
```

This is old code, not used anymore.

```
<xsl:template name="separateur.objet.spec"> ? </xsl:template>
```

The translation of `<bvolume>foo</bvolume>` is 'Bar foo', where 'Bar' is the name attribute of the element (it could be 'volume' or 'vol.'; it is defined by the DTD).

```
<xsl:template match='bvolume|bnumber|bpages|bchapter|bseries|bedition'>
  <xsl:value-of select="@bname"/>
  <xsl:text> </xsl:text><xsl:apply-templates/>
  <xsl:call-template name="separateur.objet"/>
</xsl:template>
```

The translation of `<btitle>` is the title in italics.

```
<xsl:template match='btitle'>
  <fo:inline font-style='italic'><xsl:apply-templates/>. </fo:inline>
</xsl:template>
```

The translation of `<bjournal>foo</bjournal>` is "in « foo »", or something like that.

```
<xsl:template match='bjournal|bbooktitle'>
  <xsl:text>in « </xsl:text><xsl:apply-templates/>
       <xsl:text> »</xsl:text>
  <xsl:call-template name="separateur.objet"/>
</xsl:template>
```

Translation of most bibliographic elements is trivial.

```
<xsl:template match='byear|bmonth|btype|bschool|bpublisher|
   bnote|borganization|binstitution|baddress|bhowpublished'>
  <xsl:apply-templates/>
  <xsl:call-template name="separateur.objet"/>
</xsl:template>
```

Translation of a `<bdoi>` element. This is a link to 'http://dx.doi.org/xxx'.

```
<xsl:template match='bdoi'>
  <xsl:value-of select="@bname"/><xsl:text> </xsl:text>
```

```
868   <fo:basic-link color="{$linkColor}">
869    <xsl:attribute
870      name="external-destination">http://dx.doi.org/<xsl:value-of select="."/>
871    </xsl:attribute>
872    <xsl:apply-templates/>
873  </fo:basic-link>
874   <xsl:call-template name="separateur.objet"/>
875 </xsl:template>
```

Transformation of `<cit>`. The result is a link to a bibliographical item.

```
876 <xsl:template match="cit">
877  <fo:basic-link color="{$linkColor}">
878     <xsl:attribute name="internal-destination">
879       <xsl:value-of select="ref/@target"/>
880     </xsl:attribute>
881     <xsl:text>[</xsl:text>
882     <xsl:value-of select="id(ref/@target)/@key"/>
883     <xsl:text>]</xsl:text>
884   </fo:basic-link>
885 </xsl:template>
```

The bibliography has completely changed in the new DTD. You can get an idea of the new code by looking at the HTML version, given in the previous chapter, and replacing all HTML formatting commands, using the templates above as example. We first construct a key, named 'bibliotypes', that associates to each bibliography entry its type, a letter between d and k, then for each letter, we put in the variable (for instance `$d`) the number of entries of this type. Then comes the following template, it is similar to 'tri-par-publis' in section 6.4 (details omitted). The template `biblioname` is no more used, since titles are shown here.

```
886 <xsl:template match="biblio">
887     <xsl:call-template name="secNumberedHeading"/>
888     <xsl:call-template name="tri"/> d
889     <fo:block font-weight='bold' font-size="14pt" space-before="5pt"
890        keep-with-next='always'>Year Publications</fo:block>
891     <xsl:call-template name="tri"/> e
892     <xsl:call-template name="tri"/> f
893     <xsl:call-template name="tri"/> g
894     <xsl:call-template name="tri"/> h
895     <xsl:call-template name="tri"/> i
896     <xsl:call-template name="tri"/> j
897     <xsl:call-template name="tri"/> k
898 </xsl:template>
```

The following template takes four arguments, `$str`, which is a letter, between d and k, `$title` and `$title2`, two titles, and `$countPrevious` the number of items already put in the bibliography. It outputs all entries, of type `$str`, in some order (the first sort is useless, of course).

```
899 <xsl:template name="tri">
900   <xsl:param name="str"/>
901   <xsl:param name="title"/>
902   <xsl:param name="title2"/>
903   <xsl:param name="countPrevious"/>
904   <xsl:if test="key('bibliotypes',$str)[1]">
905     <xsl:call-template "bib-title"/>
906     <xsl:for-each select="key('bibliotypes',$str)">
```

```
907         <xsl:sort select="concat(
908           substring('a', 1 div (note[@type='from']/text()='refer')),
909           substring('b', 1 div (note[@type='from']/text()='year')),
910           substring('c', 1 div (note[@type='from']/text()='foot')))"/>
911         <xsl:sort select="descendant::author[1]/persName[1]/surname/text()"/>
912         <xsl:apply-templates select=".">
913           <xsl:with-param name="pos">
914             <xsl:value-of select="$countPrevious+position()"/></xsl:with-param>
915         </xsl:apply-templates>
916       </xsl:for-each>
917     </xsl:if>
918 </xsl:template>
```

This outputs one of the two titles (in 14pt for the main title, 12pt for the subtitle).

```
919 <xsl:template name="bib-title"/>
920   <xsl:if test="string-length($title)>0">
921     <fo:block font-weight='bold' font-size="14pt" space-before="5pt"
922        keep-with-next='always'>
923       <xsl:value-of select='$title'/>
924     </fo:block>
925   </xsl:if>
926   <xsl:if test="string-length($title2)>0">
927     <fo:block font-weight='bold' font-size="12pt" space-before="5pt"
928        text-indent="1em" keep-with-next='always'>
929       <xsl:value-of select='$title2'/>
930     </fo:block>
931   </xsl:if>
932 </xsl:template>
```

This outputs the entry, in a block shifted left by two ems, starting with the number in brackets, followed by a period, with a new line as separator.

```
933 <xsl:template match="biblStruct">
934   <xsl:param name="pos" />
935   <fo:block space-before="15pt" text-indent="-2em">
936     <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
937     <xsl:text>[</xsl:text>
938     <xsl:value-of select="$pos"/>
939     <xsl:text>] </xsl:text>
940     <xsl:apply-templates/>
941     <xsl:text>.</xsl:text>
942   </fo:block>
943   <xsl:text>&#x0a;</xsl:text>
944 </xsl:template>
```

Trivial templates, for `<edition>`, `<note>` of type 'bnote', 'typedoc', 'howpublished', `<orgName>`, `<addrLine>`, or `<title>` of level 's' (corresponding to series in BibTEX). The main difference with the code shown above is that we do not call `separateur.objet`, but each item inserts a comma before it (except the first item, see below).

```
945 <xsl:template match="edition | note[@type='bnote' | note[@type='howpublished']
946      | title[@level='s'] | note[@type='typdoc'] | orgName | addrLine">
947   <xsl:text>, </xsl:text>
948   <xsl:apply-templates/>
949 </xsl:template>
```

These notes are used for sorting, hence produce no text.

```
<xsl:template match="note[@type='from']" />
<xsl:template match="note[@type='userid']" />
<xsl:template match="note[@type='classification']"/>
```

These are trivial.

```
<xsl:template match="imprint | address">
  <xsl:apply-templates/>
</xsl:template>
```

The case of `<edition>` is a bit more complicated. As already mentioned, the LaTeX companion says that the value should be something like 'Second', and the typeset text should be 'second edition'[2].

```
<xsl:template match="bedition">
  <xsl:text>, </xsl:text>
  <xsl:value-of select="@type"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
```

Other trivial rules for `<title>`. Level 'j' corresponds to the BibTeX field journal, while level 'a' corresponds to the title (for type inbook, article, incollection, inproceedings and conference). In one case we output something like 'in "foo"', using straight double quotes instead of guillemets as was the case of the old DTD; in the other case, we output the value, using italic font style, without an initial comma, because the title follows the author list, which is terminated by a period.

```
<xsl:template match="title[@level='j']">
  <xsl:text>, in "</xsl:text>
  <xsl:apply-templates/>
  <xsl:text>"</xsl:text>
</xsl:template>

<xsl:template match="title[@level='a']">
  <fo:inline font-style='italic'> <xsl:apply-templates/></fo:inline>
</xsl:template>
```

A BibTeX title gives a `<title>` of level 'a' in cases shown above, and level 'm' otherwise; as explained in the previous chapter, if `<title>` is below `<analytic>`, then the type should be 'a', hence the test is false for this title; as a consequence, if the test is true, there are two titles, and we are proceeding with the second one, handling it the same as level 'j', otherwise, it is the first, handling it as level 'a'.

```
<xsl:template match="title[@level='m']">
  <xsl:choose>
    <xsl:when test="string-length(../../analytic/title) > 0">
      <xsl:text>, in "</xsl:text>
      <xsl:apply-templates/>
      <xsl:text>"</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <fo:inline font-style='italic'><xsl:apply-templates/></fo:inline>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

[2]This implies that the first character has to be lower-casified

Translation of `<biblScope>`. Depending on the type attribute, this gives ‘vol. 17’, or ‘chap. 25’, or ‘n° 33’, preceded by a comma, as usual.

```
<xsl:template match="biblScope[@type='volume']">
  <xsl:text>, vol. </xsl:text>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="biblScope[@type='chapter']">
  <xsl:text>, chap. </xsl:text>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="biblScope[@type='number']">
  <xsl:text>, n</xsl:text>
  <fo:inline vertical-align="super">o</fo:inline>
  <xsl:text> </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
```

Comment from the preview chapter: *In the case of ‘pages’ we output something like ‘p. 10–30’. If the text contains neither a dash nor an en-dash, we output ‘42 p’.* As you can see, the last sentence is wrong in the Pdf case.

```
<xsl:template match="biblScope[@type='pages'][text()!='']">
  <xsl:text>, </xsl:text>
  <xsl:if test="string-length(substring-before(., '-'))
    or string-length(substring-before(., '&#8211;'))>0">
   <xsl:text>p. </xsl:text>
  </xsl:if>
  <xsl:apply-templates/>
</xsl:template>
```

In the case of `<dateStruct>`, we output the value of `<year>` and `<month>`. Note: is a simple copy enough here?

```
<xsl:template match="dateStruct">
  <xsl:text>, </xsl:text>
  <xsl:value-of select="month"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="year"/>
</xsl:template>
```

In the case of `<publisher>`, we output the two children, `<orgName>` and `<address>`.

```
<xsl:template match="publisher">
  <xsl:apply-templates select="orgName"/>
  <xsl:if test="./address">
    <xsl:apply-templates select="address"/>
  </xsl:if>
</xsl:template>
```

This is wrong and unused.

```
<xsl:template match='bdoi'>
  ...
</xsl:template>
```

Case of a reference in the bibliography. The destination of the link is the value of the url attribute, or the value of the element is the attribute is absent.

```
<xsl:template match="biblio//ref">
  <xsl:choose>
    <xsl:when test="./@url">
      <xsl:text>, </xsl:text>
      <fo:basic-link color="{$linkColor}">
        <xsl:attribute name="external-destination">
          <xsl:value-of select="./@url"/>
        </xsl:attribute>
        <xsl:apply-templates/>
      </fo:basic-link>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>, </xsl:text>
      <fo:basic-link color="{$linkColor}">
        <xsl:attribute name="external-destination">
          <xsl:value-of select="."/>
        </xsl:attribute>
        <xsl:apply-templates/>
      </fo:basic-link>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Same code as in the HTML version, see section 6.4.

```
<xsl:template match="analytic">
 ...
</xsl:template>

<xsl:template match="monogr">
...
</xsl:template>
```

7.7 People

There are two kinds of list of people, in the text and in the bibliography, and a person is represented differently. This a bit annyoing; we could imagine the case where we have a global list at the start of the document and references to the list; it would then be easier to associate to each member of the team its publication list. We start with the bibliography.

Translation of <bauteurs>. This is a list of <bpers>, with an optional <etal> (this one seems wrongly translated; it has a nom but no prenom). For each author, we output the first name, the particle, the last name.[3]

```
<xsl:template match ="bauteurs">
  <fo:inline font-variant='small-caps'>
     <xsl:for-each select="bpers|etal">
        <xsl:value-of select="@prenom"/>
        <xsl:text> </xsl:text>
        <xsl:if test="@part">
```

[3] What about junior?

```
            <xsl:value-of select="@part"/>
            <xsl:text> </xsl:text>
          </xsl:if>
          <xsl:value-of select="@nom"/>
          <xsl:call-template name="separateur.objet"/>
       </xsl:for-each>
    </fo:inline>
    <xsl:text> </xsl:text>
</xsl:template>
```

Same idea. A comma is put after each editor. After that, 'editor' or 'editors' is added at the end.

```
<xsl:template match ="bediteur">
  <fo:inline font-variant='small-caps'>
     <xsl:for-each select="bpers|etal">
        <xsl:value-of select="@prenom"/>
        <xsl:text> </xsl:text>
        <xsl:if test="@part">
           <xsl:text> </xsl:text>
           <xsl:value-of select="@part"/>
        </xsl:if>
        <xsl:value-of select="@nom"/>
        <xsl:text>, </xsl:text>
      </xsl:for-each>
    </fo:inline>
    <xsl:choose>
     <xsl:when test="count(bpers|etal) != 1">
        <xsl:text>editors</xsl:text>
     </xsl:when>
     <xsl:otherwise>
       <xsl:text>editor</xsl:text>
     </xsl:otherwise>
    </xsl:choose>
  <xsl:call-template name="separateur.objet"/>
</xsl:template>
```

Translation of an `<author>` (a list of authors) in the new DTD bibliography. Authors are separated by commas, with a period at the end. Initials are used rather than full first name if available.

```
<xsl:template match="author">
  <fo:inline font-variant='small-caps'>
    <xsl:for-each select="persName">
      <xsl:choose>
        <xsl:when test="surname='ETAL'">
           <xsl:text>et al.</xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:call-template name="jg.pers"/>
          <xsl:call-template name="separateur.objet"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </fo:inline>
```

```
  <xsl:text> </xsl:text>
</xsl:template>
```

Case of `<editor>`, the list of editors. We add a comma in front if there are authors (otherwise, the element is output just after the title). Nothing special is done in the 'ETAL' case.

```
<xsl:template match="editor">
  <xsl:if test="../..//author">   <xsl:text>, </xsl:text></xsl:if>
  <fo:inline font-variant='small-caps'>
    <xsl:for-each select="persName">
      <xsl:call-template name="jg.pers"/>
      <xsl:call-template name="separateurED.objet"/>
    </xsl:for-each>
  </fo:inline>
  <xsl:text> (editor</xsl:text>
  <xsl:call-template name="pluriel-p">
    <xsl:with-param name="liste" select="persName" />
  </xsl:call-template>
  <xsl:text>). </xsl:text>
</xsl:template>
```

Common code for authors or editors.

```
<xsl:template name=jgpers">
  <xsl:choose>
    <xsl:when test="initial"><xsl:apply-templates select="initial"/></xsl:when >
    <xsl:otherwise> <xsl:apply-templates select="foreName"/> </xsl:otherwise>
  </xsl:choose>
  <xsl:text> </xsl:text>
  <xsl:value-of select="surname"/>
</xsl:template>
```

This outputs an S in case the list has more than one element.

```
<xsl:template name="pluriel-p">
   <xsl:param name="liste" />
   <xsl:if test="count($liste)>1">s</xsl:if>
</xsl:template>
```

Translation of `<catperso>`. This is like a subsection, with a title (in `<head>`), we select all the `<pers>` children.

```
<xsl:template match="catperso">
  <fo:block space-before="3pt">
    <fo:block font-weight='bold' text-indent="-15pt">
      <xsl:value-of select="head"/>
    </fo:block>
    <xsl:for-each select="pers">
        <fo:block> <xsl:call-template name="pers"/></fo:block>
    </xsl:for-each>
  </fo:block>
</xsl:template>
```

This replaces the previous template in the case of the new DTD. Changes are obvious.

```
<xsl:template match="team/participants">
  <fo:block space-before="3pt">
    <fo:block font-weight='bold' text-indent="-15pt">
        <xsl:value-of select="translate(@category, '_', ' ')"/>
```

```
    </fo:block>
    <xsl:for-each select="person">
        <fo:block> <xsl:apply-templates select="."/></fo:block>
    </xsl:for-each>
  </fo:block>
</xsl:template>
```

Like above, but the result is inline: we do not put `<pers>` in a block, but use commas as separators.

```
<xsl:template match="participants|participant|participante|participantes">
 <fo:block space-after.optimum="4pt">
   <fo:inline>
     <xsl:attribute name="font-weight">bold</xsl:attribute>
       <xsl:value-of select="@titre"/>
   </fo:inline>
   <fo:inline>
    <xsl:for-each select="pers">
       <xsl:call-template name="pers"/>
       <xsl:call-template name="separateur.objet"/>
    </xsl:for-each>
   </fo:inline>
 </fo:block>
</xsl:template>
```

There are some differences in the new DTD. Instead of four elements, we have a single one, named `<participants>`, and the title will be 'Participant' in the case where is a single `<person>`, 'Participants' if there are more than one. Two consecutive `<participants>` are implicitly merged, the persons in the second one are handled with the first. In some cases, the implementation is wrong; the case of `<refperson>` is sometimes considered here (but omitted in the HTML case).

```
<xsl:template match="participants">
  <xsl:if test="not(preceding-sibling::participants) and not(parent::team)">
    <fo:block space-after.optimum="4pt">
      <fo:inline>
        <xsl:attribute name="font-weight">bold</xsl:attribute>
        <xsl:text>Participant</xsl:text>
          <xsl:call-template name="pluriel-p">
            <xsl:with-param name="liste" select="person" />
          </xsl:call-template>:
      </fo:inline>
      <fo:inline>
        <xsl:for-each select="person | following-sibling::participants/person
             | refperson">
          <xsl:apply-templates select="." mode="section"/>
          <xsl:call-template name="separateur.objet"/>
        </xsl:for-each>
      </fo:inline>
    </fo:block>
  </xsl:if>
</xsl:template>
```

Transformation of `<pers>`. This is easy: we have two attributes, first name and last name. Unless empty, the content is put in brackets.

```
<xsl:template name="pers">
    <xsl:value-of select="./@prenom"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="./@nom"/>
    <xsl:if test="not(normalize-space(string(.)) =’’)">
       <xsl:text> [</xsl:text>
        <xsl:apply-templates/>
       <xsl:text>]</xsl:text>
    </xsl:if>
</xsl:template>
```

In the case of the new DTD, we have a `<firstname>` and a `<lastname>`, instead of these two attributes, and `<moreinfo>` instead of a content. These are the only changes in the case of a `<refperson>` or a `<person>` outside the team composition[4].

```
<xsl:template match="person">...</xsl:template>
<xsl:template match="person" mode="section">...</xsl:template>
```

A `<refperson>` is just a reference to a `<person>` defined by its href attribute. In this code, we check that the reference is indeed a person.

```
<xsl:template match="refperson" mode="section">
  <xsl:variable name ="curid" select="@ref"/>
  <xsl:variable name ="curelement" select="/raweb//*[@id=$curid]"/>
  <xsl:if test="$curelement[name()=’person’]">
     <xsl:apply-templates select="$curelement" mode="section"/>
  </xsl:if>
</xsl:template>
```

7.8 References

The translation of `<xref>` (external reference) is a link, that points to the value of the url attribute. It has some color...

```
<xsl:template match="xref">
 <fo:basic-link color="{$linkColor}">
   <xsl:attribute name="external-destination">
     <xsl:value-of select="@url"/>
   </xsl:attribute>
   <xsl:apply-templates/>
 </fo:basic-link>
</xsl:template>
```

The same code is used in the case when the link is a `<citation>` element; but we add a separator (comma or period) after it.

```
<xsl:template match="citation/xref">
 <fo:basic-link color="{$linkColor}">
   <xsl:attribute name="external-destination">
     <xsl:value-of select="@url"/>
   </xsl:attribute>
   <xsl:apply-templates/>
 </fo:basic-link>
```

[4] In the `<team>` element, the `<hdr>` element is taken into account, same method as in the HTML case, see previous chapter.

```
 <xsl:call-template name="separateur.objet"/>
</xsl:template>
```

Translation of an internal link. There is an attribute target that says to what it points. The `<ref>` element is empty, the link is not: the value of the link element is obtained by evaluating the target in the 'xref' mode. This should give a number (or a sequence like 12.3.4 for a subsection). In the case of a table, figure, math expression, this should be the number of the table, figure, etc.; in the case of a division, it should be the number associated to the title.

```
<xsl:template match="ref">
 <fo:basic-link color="{$linkColor}">
  <xsl:attribute name="internal-destination">
    <xsl:value-of select="translate(@target,'_','-')"/>
  </xsl:attribute>
  <xsl:apply-templates mode="xref" select="id(@target)" />
  <xsl:apply-templates/>
 </fo:basic-link>
</xsl:template>
```

The case of the new DTD, is much more complicated, since we have to compute the number of the reference. The same algorithm is used as in the HTML case, see previous chapter, section 6.4. We have a template `positionInBib` (same as line 3054 in the previous chapter), and we define *position-in-bib(W)* in the same way. Then the code looks like this:

```
<xsl:template name="ref-in-bib">
  <fo:basic-link color="{$linkColor}">
    <xsl:attribute name="internal-destination">
      <xsl:value-of select="$curid"/>
    </xsl:attribute>
    <xsl:choose>
         position-in-bib(d)
         position-in-bib(e)
         position-in-bib(f)
         position-in-bib(g)
         position-in-bib(h)
         position-in-bib(i)
         position-in-bib(j)
         position-in-bib(k)
     </xsl:choose>
  </fo:basic-link>
  <xsl:text>]</xsl:text>
</xsl:template>
```

This is called when the attribute xlink:href does not start with a sharp sign; it can be an external reference (otherwise, the document is not valid, and the code is wrong, not shown here).

```
<xsl:template name="external-ref">
 <xsl:choose>
   <xsl:when test="@location='extern'">
     <fo:basic-link color="{$linkColor}">
       <xsl:attribute name="external-destination">
          <xsl:value-of select="@xlink:href"/>
       </xsl:attribute>
       <xsl:apply-templates/>
     </fo:basic-link>
   </xsl:when>
```

```
    <xsl:otherwise> ... </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This is called when the attribute xlink:href has the form '#foo', with 'foo' in `$curid`, and the target `$curelement` is `<identification>` or a child. In this case, we replace it by `<team>`. The idea is to evaluate the target in pre-ref, xref and post-ref modes, and put the value obtained in xref mode in the link.

```
<xsl:template name="ref-to-ident">
  <xsl:apply-templates select="/raweb/identification/team" mode="pre-ref" />
  <fo:basic-link color="{$linkColor}" internal-destination="{$curid}">
    <xsl:apply-templates select="/raweb/identification/team" mode="xref"/>
  </fo:basic-link>
  <xsl:apply-templates select="/raweb/identification/team" mode="post-ref" />
</xsl:template>
```

This is called when the attribute xlink:href has the form '#foo', with 'foo' in `$curid`, and the target is `$curelement`. Note that empty pre-ref and post-ref are defined for `<table>`, `<formula>`, `<subsection>`, `<biblio>`, `<presentation>`, `<fondements>`, `<domaine>`, `<logiciels>`, `<object>`, `<diffusion>`, `<resultats>`, `<contrats>`, `<identification>`, `<international>`, `<team>`, `<li>`. In all other cases default will be used (let's hope this case does not occur).

```
<xsl:template name="internal-ref">
  <xsl:apply-templates select="$curelement" mode="pre-ref" />
  <fo:basic-link color="{$linkColor}" internal-destination="{$curid}">
    <xsl:apply-templates select="$curelement" mode="xref"/>
  </fo:basic-link>
  <xsl:apply-templates select="$curelement" mode="post-ref" />
</xsl:template>
```

This is now the code for a reference in the new DTD.

```
<xsl:template match="ref">
  <xsl:choose>
    <xsl:when test="starts-with(@xlink:href, '#')">
      <xsl:variable name="curid" select="substring-after(@xlink:href, '#')"/>
      <xsl:variable name="curelement" select="/raweb//*[@id=$curid]"/>
      <xsl:variable name="curlabel"
          select="$curelement/ancestor-or-self::subsection" />
     <xsl:variable name="curlabelidentification"
          select="$curelement/ancestor-or-self::identification"/>
     <xsl:choose>
       <xsl:when test="$curlabelidentification">
         <xsl:call-template name="ref-to-ident"/>
       </xsl:when>
       <xsl:when test="@location='biblio'">
          <xsl:call-template name="ref-in-bib"/>
       </xsl:when>
       <xsl:when test="@location='intern'">
         <xsl:call-template name="external-ref"/>
       </xsl:when>
     </xsl:choose>
   </xsl:when>
   <xsl:otherwise>
      <xsl:call-template name="external-ref"/>
```

```
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

In the case of a section, computing the number is trivial: we take it from the DTD.

```
<xsl:template
     match="biblio|presentation|fondements|domaine|logiciels|resultats|
       contrats|composition|international|diffusion" mode = 'xref'>
  <xsl:value-of select = "@numero"/>
</xsl:template>
```

In the case of a subsection, computing the number is non trivial. The first component is the section number, followed by a dot.

```
<xsl:template name="sec.num">
    <xsl:value-of select =
          "ancestor-or-self::*[self::composition or self::presentation
           or self::fondements or self::domaine or self::logiciels
           or self::resultats or self::contrats or self::international
           or self::diffusion or self::biblio]/@numero"/>
  <xsl:text>.</xsl:text>
</xsl:template>
```

In the original DTD, the section number was an attribute of the section, hence was easy to compute. This is the replacement code:

```
  <xsl:when test="ancestor-or-self::*[self::identification]">1</xsl:when>
  <xsl:when test="ancestor-or-self::*[self::presentation]">2</xsl:when>
  <xsl:when test="ancestor-or-self::*[self::fondements]">3</xsl:when>
  <xsl:when test="ancestor-or-self::*[self::domaine]">4</xsl:when>
  <xsl:when test="ancestor-or-self::*[self::logiciels]">5</xsl:when>
  <xsl:when test="ancestor-or-self::*[self::resultats]">6</xsl:when>
  <xsl:when test="ancestor-or-self::*[self::contrats]">7</xsl:when>
  <xsl:when test="ancestor-or-self::*[self::international]">8</xsl:when>
  <xsl:when test="ancestor-or-self::*[self::diffusion]">9</xsl:when>
  <xsl:when test="ancestor-or-self::*[self::biblio]">10</xsl:when>
```

Some people find it annoying that section 7 is followed by section 9, in case where section 'International' is missing. Thus, the correct number is used; for instance, if current section is 'domaine', its number is the number of 'identification' plus 'presentation', plus 'fondements' plus one. Full code nor shown, but obvious.

```
<xsl:template name="calculateNumberSection">
  <xsl:choose>
   <xsl:when test="ancestor-or-self::*[self::identification]">1</xsl:when>
   <xsl:when test="ancestor-or-self::*[self::presentation]">
     <xsl:value-of select="count(/raweb/identification)+1"/>
   </xsl:when>
   <xsl:when test="ancestor-or-self::*[self::fondements]">
     <xsl:value-of select="count(/raweb/identification)
                  +count(/raweb/presentation)+1"/>
   </xsl:when>
   <xsl:when test="ancestor-or-self::*[self::domaine]">
     <xsl:value-of select="count(/raweb/identification)
           +count(/raweb/presentation)+count(/raweb/fondements)+1"/>
   </xsl:when>
   ...
```

```
    <xsl:when test="ancestor-or-self::*[self::biblio]">
       ...
    </xsl:when>
    <xsl:otherwise>Section</xsl:otherwise>
   </xsl:choose>
   <xsl:text>.</xsl:text>
</xsl:template>

<xsl:template match="biblio|presentation|fondements|domaine|
      logiciels|resultats|contrats|identification|international|diffusion|team"
    mode = 'xref'>
   <xsl:call-template name="calculateNumberSection"/>
</xsl:template>
```

This computes the number associated to a division (module, div2, div3, and div4) by counting these things. It is used twice: when the division is typeset (more exactly, when its title is typeset), and when the division appears in a link.

```
<xsl:template name="calculateNumber">
    <xsl:param name="numbersuffix"/>
    <xsl:call-template name="sec.num"/>
    <xsl:number level="multiple" from="raweb"
        count="module|div2|div3|div4"/>
    <xsl:value-of select="$numbersuffix"/>
</xsl:template>
```

This is the code for the new DTD, a little bit simpler.

```
<xsl:template name="calculateNumber">
   <xsl:param name="numbersuffix"/>
   <xsl:call-template name="calculateNumberSection"/>
   <xsl:number level="multiple" from="raweb" count="subsection"/>
   <xsl:value-of select="$numbersuffix"/>
</xsl:template>
```

This calls the template shown above. Is named `<subsection>` in the new DTD.

```
<xsl:template mode="xref" match="module|div2|div3|div4">
   <xsl:call-template name="calculateNumber"/>
</xsl:template>
```

It is not clear where the anchor of a section should be: on the division or its title. For this reason this rule is added for `<head>` (or `<bodyTitle>` in the new DTD); it is useless because we decided finally that the anchor should not be on the title. In a previous version of Tralics, `<anchor>` elements were generated (then removed, and re-inserted). Anchors should not be used in the Raweb.

```
<xsl:template match="head|anchor" mode ="xref">
    <xsl:call-template name="calculateNumber"/>
</xsl:template>
```

The idea is that every math formula (a `<formula>` element) that has an id is numbered.

```
<xsl:template match="formula" mode="xref">
   <xsl:number level = "any" count="formula[@id]"/>
</xsl:template>
```

In the case of a figure, we count only figures that are not inline, via the rend attribute.

```
1374 <xsl:template name="calculateFigureNumber">
1375   <xsl:number count="figure[@rend != 'inline']" level="any"/>
1376 </xsl:template>
1377 <xsl:template match='figure' mode="xref">
1378    <xsl:call-template name="calculateFigureNumber"/>
1379 </xsl:template>
```

Same for a table. In the new DTD, the test is: all tables whose ancestors are not `<object>`.

```
1380 <xsl:template name="calculateTableNumber">
1381     <xsl:number count="table[@rend != 'inline']" level="any"/>
1382 </xsl:template>
1383 <xsl:template match='table' mode="xref">
1384    <xsl:call-template name="calculateTableNumber"/>
1385 </xsl:template>
```

In the case of a `<note>`, we count all elements that have a place attribute with value 'foot'. This is called `<footnote>` in the new DTD.

```
1386 <xsl:template name="calculateFootnoteNumber">
1387   <xsl:number  level="any" count="note[@place='foot']"/>
1388 </xsl:template>
```

The case of an item is more complicated. Originally, there was a test: is the parent a `<list>` of type bibliography? in this case, '[25]' is produced instead of '25'. This test was removed. Thus, the only non trivial point is that numbering depends on the list level.

```
1389 <xsl:template match="item" mode="xref">
1390   <xsl:variable name="listdepth" select="count(ancestor::list)"/>
1391   <xsl:variable name="listNFormat">
1392    <xsl:choose>
1393     <xsl:when test="$listdepth=1"> <xsl:text>1</xsl:text> </xsl:when>
1394     <xsl:when test="$listdepth=2"> <xsl:text>i</xsl:text> </xsl:when>
1395     <xsl:when test="$listdepth=3"> <xsl:text>a</xsl:text> </xsl:when>
1396     <xsl:when test="$listdepth=4"> <xsl:text>I</xsl:text> </xsl:when>
1397    </xsl:choose>
1398   </xsl:variable>
1399   <xsl:number format="{$listNFormat}"/>
1400 </xsl:template>
```

In the new DTD, we use `<li>` instead of `<item>`. All items are counted the same, independently of the list depth. There is however a second rule, that looks like the previous one, except that the list level is computed but the sum of the number of ancestors of type `<glosslist>`, `<descriptionlist>`, `<simplelist>`, and `<orderedlist>`.

```
1401 <xsl:template match='li' mode="xref">
1402   <xsl:number count="li" level="single" />
1403 </xsl:template>
```

Trivial templates added for the new DTD.

```
1404 <xsl:template name="calculateObjectNumber">
1405   <xsl:number count="object" level="any"/>
1406 </xsl:template>
1407
1408 <xsl:template name="calculateRessourceNumber">
1409   <xsl:number count="ressource" level="any"/>
1410 </xsl:template>
```

These are the matching rules.

```
<xsl:template match='object' mode="xref">
   <xsl:call-template name="calculateObjectNumber"/>
</xsl:template>

<xsl:template match="ressource" mode="xref">
  <xsl:call-template name="calculateRessourceNumber"/>
</xsl:template>
```

7.9 Generic elements

A `<p>` element is translated into a `<fo:block>` element. It has some attributes, for instance a constant font size.

```
<xsl:template match="p">
  <fo:block font-size="{$bodySize}">
```

If the preceding sibling is a `<p>` element, and the current element has not noindent = 'true' as attribute, the paragraph will be indented, said otherwise, it will have a non-zero text-indent. The space-before attribute is computed as follows. We define a maximum value in the case where the preceding sibling is a `<p>`. We define an optimum value in the case where spacebefore is given as attribute to the current element. We define an optimum value if nothing is given, and the preceding sibling is a `<p>`. Note: we should always define all three values, because the default is zero, and this is wrong (but fotex interprets badly these quantities...)

```
    <xsl:if test="preceding-sibling::p">
       <xsl:if test ="not(@noindent)">
           <xsl:attribute name="text-indent">
              <xsl:value-of select="$parIndent"/>
           </xsl:attribute>
       </xsl:if>
       <xsl:choose>
         <xsl:when test="@spacebefore">
           <xsl:attribute name="space-before.optimum">
               <xsl:value-of select="@spacebefore"/>
           </xsl:attribute>
         </xsl:when>
         <xsl:otherwise>
           <xsl:attribute name="space-before.optimum">
              <xsl:value-of select="$parSkip"/>
           </xsl:attribute>
         </xsl:otherwise>
       </xsl:choose>
       <xsl:attribute name="space-before.maximum">
         <xsl:value-of select="$parSkipmax"/>
       </xsl:attribute>
    </xsl:if>
    <xsl:if test="not(preceding-sibling::p)">
       <xsl:if test="@spacebefore">
          <xsl:attribute name="space-before.optimum">
             <xsl:value-of select="@spacebefore"/>
          </xsl:attribute>
       </xsl:if>
    </xsl:if>
```

In the case where the rend attribute is 'centered' or 'center', we set text-align to 'center'. If it is 'flushed-left' or 'flushed-right' we set it to left or right. If it is 'quoted' or 'justify' we set it to 'justify' (in the case 'quoted', we also set both left and right margins to 1 cm). The new code has a comment that says "ARG, useless with the new DTD but ..."

```
    <xsl:choose>
      <xsl:when test="@rend ='centered'">
       <xsl:attribute name="text-align">center</xsl:attribute>
      </xsl:when>
      <xsl:when test="@rend ='center'">
       <xsl:attribute name="text-align">center</xsl:attribute>
      </xsl:when>
      <xsl:when test="@rend ='flushed-left'">
       <xsl:attribute name="text-align">left</xsl:attribute>
      </xsl:when>
      <xsl:when test="@rend ='flushed-right'">
       <xsl:attribute name="text-align">right</xsl:attribute>
      </xsl:when>
      <xsl:when test="@rend ='quoted'">
       <xsl:attribute name="text-align">justify</xsl:attribute>
       <xsl:attribute name="margin-left">1cm</xsl:attribute>
       <xsl:attribute name="margin-right">1cm</xsl:attribute>
      </xsl:when>
      <xsl:when test="@rend ='justify'">
       <xsl:attribute name="text-align">justify</xsl:attribute>
      </xsl:when>
    </xsl:choose>
```

We insert the content of the <p> here.

```
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Conversion of <hi>. The result is an inline object, the non trivial part concerns transformation of the rend attribute, but this is defined elsewhere.

```
<xsl:template match="hi">
  <fo:inline>
     <xsl:call-template name="rend" />
     <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

In the new DTD, <hi> has sometimes been replaced by .

```
<xsl:template match="em">
  <fo:inline>
    <xsl:call-template name="style"/>
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

This is currently unused.

```
<xsl:template match="code">
  <fo:inline font-family="{$typewriterFont}">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

This is currently unused.

```
<xsl:template match="ident">
  <fo:inline color="{$identColor}" font-family="{$sansFont}">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

This is trivial.

```
<xsl:template match="term">
  <fo:inline font-style="italic">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

7.10 Lists

This translates a `<list>`. In any case, the result is a `<fo:list-block>` with some constant right margin and a left margin that depends on the context: normal list, normal glossary, or glossary in a list. All items in the list are considered.

```
<xsl:template match="list">
  <xsl:call-template name="titlelist"/>
  <fo:list-block margin-right="{$listRightMargin}">
    <xsl:call-template name="setListIndents"/>
    <xsl:choose>
      <xsl:when test="@type='gloss'">
        <xsl:attribute name="margin-left">
          <xsl:choose>
            <xsl:when test="ancestor::list"><xsl:value-of
                        select="$listLeftGlossInnerIndent"/></xsl:when>
            <xsl:otherwise><xsl:value-of select="$listLeftGlossIndent"/></xsl:otherwise>
          </xsl:choose>
        </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="margin-left">
          <xsl:value-of select="$listLeftIndent"/></xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates select="item"/>
  </fo:list-block>
</xsl:template>
```

If the first child of a list is `<head>` we start with a block containing the title in italics. This is not used in the Raweb.

```
<xsl:template name="titlelist">
  <xsl:if test="child::head">
    <fo:block font-style="italic"
        text-align="start"
        space-before.optimum="4pt">
      <xsl:apply-templates select="head"/>
    </fo:block>
  </xsl:if>
</xsl:template>
```

Note: in the case of the new DTD, there are more than one type of lists. Thus the template named `<list>` has to be replaced by two templates: one for `<orderedlist>`, `<simplelist>`, `<descriptionlist>` and one for `<glosslist>`. The outer test (has this element an attribute type with value 'gloss'?) becomes: is this element a `<glosslist>` whose answer is trivial. The inner test (is the ancestor a list?) becomes more complicated. A list contains a sequence of `<item>` elements (renamed to `<li>` in the new DTD) optionally preceded by a `<label>`. The code shown above applies a rule to each `<item>`; this was changed to: apply a rule to everything but a `<title>`. Note that the default template for a `<label>` is to ignore it; only in 'print' mode is the action non-trivial.

```
<xsl:template match="label"/>

<xsl:template match="label" mode="print">
  <xsl:apply-templates/>
</xsl:template>
```

A list has also a space-before and space-after attributes, that depend on the level. The complete code is not shown here. In the new DTD, computing the list depth is a bit harder, because we must sum up the number of ancestors of type `<glosslist>`, `<descriptionlist>`, `<simplelist>`, and `<orderedlist>` (the list depth is used more than once: here it is used to compute the margins, below, when we typeset the item, and in section 7.8 when a reference is made to the item).

```
<xsl:template name="setListIndents">
  <xsl:variable name="listdepth" select="count(ancestor::list)"/>
  <xsl:choose>
    <xsl:when test="$listdepth=0">
      <xsl:attribute name="space-before">
        <xsl:value-of select="$listAbove-1"/>
      </xsl:attribute>
      <xsl:attribute name="space-after">
        <xsl:value-of select="$listBelow-1"/>
      </xsl:attribute>
    </xsl:when>
    <!-- Same for listdepth=1 2 or 3 -->
  </xsl:choose>
</xsl:template>
```

We use a template for this.

```
<xsl:template match="item">
  <xsl:call-template name="makeItem"/>
</xsl:template>
```

Translation of an `<item>`. The result is a `<fo:list-item>` that contains two sub-elements, we start with the first, the `<fo:list-item-label>`. It depends on the type attribute of our parent: simple, bullets, ordered, gloss, unordered (in the case of the new DTD, it depends on the name of the parent). In any case, if we have an id, we associate it to the element; then we construct a `<fo:block>`. The case where the list is in the bibliography is not shown here. The case where the label has an `n` attribute is not shown here. In the case where the list has type = 'ordered', we evaluate the item in 'xref' mode: this produces a number; this number will be flushed right (alignment end). In the case where the list is a glossary, we will left align the label using a bold font; here the label is either a `<label>` child or a `<label>` sibling. In all other cases, if there is a label sibling it will be used. In both these cases, the label is transformed using a special mode.

```
<xsl:template name="makeItem">
 <xsl:variable name="listdepth" select="count(ancestor::list)"/>
 <fo:list-item space-before.optimum="{$listItemsep}">
```

```
  <fo:list-item-label>
     <xsl:if test="@id">
       <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
    </xsl:if>
    <fo:block>
      <xsl:attribute name="margin-right">2.5pt</xsl:attribute>
      <xsl:choose>
        <xsl:when test="../@type='ordered'">
          <xsl:attribute name="text-align">end</xsl:attribute>
          <xsl:apply-templates mode="xref" select="."/>
          <xsl:text>.</xsl:text>
        </xsl:when>
        <xsl:when test="../@type='gloss'">
          <xsl:attribute name="text-align">start</xsl:attribute>
          <xsl:attribute name="font-weight">bold</xsl:attribute>
          <xsl:choose>
            <xsl:when test="label">
              <xsl:apply-templates mode="print" select="label"/>
            </xsl:when>
            <xsl:otherwise>
             <xsl:apply-templates mode="print" select="preceding-sibling::*[1]"/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:when>
         <xsl:when test="name(preceding-sibling::*[1])='label'">
          <xsl:apply-templates mode="print" select="preceding-sibling::*[1]"/>
         </xsl:when>
```

In all other cases, a default value is chosen, depending on the list level.

```
        <xsl:otherwise>
           <xsl:attribute name="text-align">center</xsl:attribute>
          <xsl:choose>
            <xsl:when test="$listdepth=1">
              <xsl:value-of select="$bulletOne"/>
            </xsl:when>
            <xsl:when test="$listdepth=2">
              <xsl:value-of select="$bulletTwo"/>
            </xsl:when>
            <xsl:when test="$listdepth=3">
              <xsl:value-of select="$bulletThree"/>
            </xsl:when>
            <xsl:when test="$listdepth=4">
              <xsl:value-of select="$bulletFour"/>
            </xsl:when>
          </xsl:choose>
        </xsl:otherwise>
      </xsl:choose>
    </fo:block>
  </fo:list-item-label>
```

Essentially, the translation is a `<fo:list-item-body>` that contains a `<fo:block>`. This block is constructed implicitly by a `<p>` or explicitly, with a normal weight for the font.

```
 <fo:list-item-body>
   <xsl:choose>
     <xsl:when test="p"> <xsl:apply-templates/> </xsl:when>
     <xsl:otherwise>
       <fo:block font-weight="normal"><xsl:apply-templates/></fo:block>
     </xsl:otherwise>
   </xsl:choose>
 </fo:list-item-body>
</fo:list-item>
</xsl:template>
```

7.11 Images

This converts an image, or something like that. The result is a `<fo:external-graphic>` object. The src attribute is taken to be the file attribute. Initially, there was a `$graphicsPrefix` prefix before the image; this was removed: images should be in the current directory. We consider the three attributes scale, width, height and angle. Let's hope that angle is not used: in TEX, putting the angle to ninety degrees exchanges the role of width and height, but you can specify the width after rotation. Since the order of attributes is irrelevant in XML, you lose[5]. But you cannot turn images in HTML. Note: if you specify a scale, you cannot specify a width nor a height. In the new DTD the attribute file is replaced by xlink:href. We removed a test to `$autoScaleFigures`, placed after all other tests.

```
<xsl:template name="generate-graphics">
  <fo:external-graphic src="{@file}">
    <xsl:choose>
      <xsl:when test="@scale">
         <xsl:attribute name="content-width">
           <xsl:value-of select="@scale * 100"/><xsl:text>%</xsl:text>
         </xsl:attribute>
      </xsl:when>
      <xsl:when test="@width">
        <xsl:attribute name="content-width">  <xsl:value-of select="@width"/>
        </xsl:attribute>
        <xsl:if test="@height">
          <xsl:attribute name="content-height">
            <xsl:value-of select="@height"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:if test="@angle">
           <xsl:attribute name="angle">
               <xsl:value-of select="@angle"/>
           </xsl:attribute>
        </xsl:if>
      </xsl:when>
      <xsl:when test="@height">
         <xsl:attribute name="content-height">
           <xsl:value-of select="@height"/>
         </xsl:attribute>
         <xsl:if test="@angle">
```

[5] This is a bug in Tralics, of course

```
            <xsl:attribute name="angle">
              <xsl:value-of select="@angle"/>
            </xsl:attribute>
          </xsl:if>
        </xsl:when>
      </xsl:choose>
    </fo:external-graphic>
</xsl:template>
```

A `<figure>` element, that has rend = 'inline', produces a graphic object by the routine shown above. If it has a `<head>` (renamed to `<caption>` in the new DTD), this is a caption, shown after the image. In the new DTD, the template is called `<ressource>`; if the attribute media is set to 'PRINT', or if the ancestor has no other resource whose media attribute is set to 'PRINT', then it will be used, otherwise discarded (this allows a different image in the Pdf or HTML version).

```
<xsl:template match="figure[@rend='inline']">
  <xsl:call-template name='generate-graphics'/>
  <xsl:if test="head">
     <fo:block text-align="center">
           <xsl:apply-templates select="head"/>
     </fo:block>
  </xsl:if>
</xsl:template>
```

All other images produce a `<fo:float>` object. It has an id, and contains an `<fo:block>`, this block is created by the procedure shown above, in case there is a file attribute, or by evaluating all `<p>` children. The content is centered. There is a second block, that contains the caption (from `<head>`).

```
<xsl:template match='figure'>
 <fo:float>
   <xsl:call-template name="addID"/>
   <fo:block text-align="center">
    <xsl:choose>
      <xsl:when test="@file">
       <xsl:call-template name='generate-graphics'/>
      </xsl:when>
      <xsl:otherwise><xsl:apply-templates select='p'/></xsl:otherwise>
    </xsl:choose>
   </fo:block>
   <fo:block>
     <xsl:call-template name="figureCaptionstyle"/>
     <xsl:value-of select="$figureWord"/>
     <xsl:call-template name="calculateFigureNumber"/>
     <xsl:text>. </xsl:text>
     <xsl:apply-templates select="head"/>
   </fo:block>
 </fo:float>
</xsl:template>
```

In the new DTD, we have `<object>` elements rather than `<figure>`. The content is always a table (so that there is no need to test for a file attribute).

```
<xsl:template match="object" >
  <fo:float>
    <xsl:call-template name="addID"/>
```

```
1680      <fo:block>
1681        <xsl:apply-templates select="table" mode="object"/>
1682      </fo:block>
1683      <fo:block>
1684        <xsl:call-template name="figureCaptionstyle"/>
1685        <xsl:value-of select="$figureWord"/>
1686        <xsl:call-template name="calculateObjectNumber"/>
1687        <xsl:text>. </xsl:text>
1688        <xsl:apply-templates select="caption"/>
1689      </fo:block>
1690    </fo:float>
1691  </xsl:template>
```

These are trivial.

```
1692  <xsl:template match="object/caption"><xsl:apply-templates/></xsl:template>
1693  <xsl:template match="ressource/caption"><xsl:apply-templates/></xsl:template>
```

7.12 Tables

Essentially, the table is handled by `blockTable`; this produces a `<fo:table>` element; but if the table is not marked inline, we use `floatTable`, this will produce a `<fo:table-and-caption>`. In the new DTD, a block table is used if the ancestor is an `<object>` (recall that an `<object>` is a floating object that contains a table that contains graphic images as cells).

```
1694  <xsl:template match="table">
1695   <xsl:choose>
1696    <xsl:when test="@rend='inline'"><xsl:call-template name="blockTable"/></xsl:when>
1697    <xsl:otherwise> <xsl:call-template name="floatTable"/> </xsl:otherwise>
1698   </xsl:choose>
1699  </xsl:template>
```

This produces a `<fo:table-and-caption>` containing a table and a caption. The table is computed as in the case without caption. There is a mechanism that turns the whole table-and-caption by ninety degree, this is not used by Tralics. The caption is preceded by something like 'Figure 99', and the number has to be computed.

```
1700  <xsl:template name="floatTable">
1701    <fo:table-and-caption>
1702        <xsl:if test="rend='landscape'">
1703          <xsl:attribute name="reference-direction">-90</xsl:attribute>
1704        </xsl:if>
1705        <xsl:call-template name="addID"/>
1706        <fo:table-caption>
1707          <fo:block text-align="{$tableCaptionAlign}"
1708                space-after="{$spaceBelowCaption}">
1709             <xsl:value-of select="$tableWord"/>
1710             <xsl:call-template name="calculateTableNumber"/>
1711             <xsl:text>. </xsl:text>
1712             <xsl:apply-templates select="head"/>
1713          </fo:block>
1714        </fo:table-caption>
1715        <xsl:call-template name="blockTable"/>
1716     </fo:table-and-caption>
1717  </xsl:template>
```

Transforming a table is easy: we select every row, and every cell in the rows. There is however a non trivial point: the current fotex implementation assumes that columns widths are given; hence we must compute them. Note: there is a `<xsl:text>` element, a contains a newline character; this improves the layout of the resulting XML; it seems useless. In the new version, the for-each has been replaced by an apply-templates to all elements of type `<th>` (this is strange) and `<tr>`. Moreover `deriveColSpecs` has been replaced by `calculateTableSpecs` (see below).

```
<xsl:template name="blockTable">
 <fo:table  text-align="{$tableAlign}">
   <xsl:call-template name="addID"/>
   <xsl:call-template name="deriveColSpecs"/>
   <fo:table-body text-indent="0pt">
     <xsl:for-each select="row">
       <xsl:text></xsl:text> <!-- this is a newline character -->
       <fo:table-row>
         <xsl:apply-templates select="cell"/>
       </fo:table-row>
      </xsl:for-each>
   </fo:table-body>
 </fo:table>
</xsl:template>
```

This seems useless. One can however imagine the situation where the table specifications are stored somewhere and we need a unique identifier for the table.

```
<xsl:template name="generateTableID">
 <xsl:choose>
  <xsl:when test="@id"> <xsl:value-of select="@id"/> </xsl:when>
  <xsl:otherwise><xsl:text>Table-</xsl:text><xsl:number level='any'/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

The procedure that computes the table specifications is in another file. The variable computed here is useless; it is perfectly legitimate to replace the call by this template by a call to inner template.

```
<xsl:template name="deriveColSpecs" >
  <xsl:variable name="no"> <xsl:call-template name="generateTableID"/>
  </xsl:variable>
  <xsl:call-template name="calculateTableSpecs"/>
</xsl:template>
```

7.13 Mathematics

This is the same as code given elsewhere. We leave the math unchanged.

```
<xsl:template match="m:math">
 <m:math>
  <xsl:copy-of select="@*"/>
  <xsl:apply-templates mode="math"/>
 </m:math>
</xsl:template>
```

```
<xsl:template match="m:*|@*|comment()|processing-instruction()|text()" mode="math">
  ...
</xsl:template>
```

We replace a formula by its content.

```
<xsl:template match="formula">
  <xsl:apply-templates/>
</xsl:template>
```

A special case is when the formula has 'display' as type. If there is an id, then the result is a <fotex:equation>, otherwise it is a <fotex:displaymath>. In the case of an equation, we use a <fo:inline> (why not use it in every case? or add the id to the equation?) Note that the for-each is useless.

```
<xsl:template match="formula[@type='display']">
  <xsl:choose>
    <xsl:when test="@id">
      <fo:inline>
        <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
        <fotex:equation>
          <xsl:for-each select="m:math">
            <xsl:apply-templates mode="math"/>
          </xsl:for-each>
        </fotex:equation>
      </fo:inline>
    </xsl:when>
    <xsl:otherwise>
      <fotex:displaymath>
        <xsl:for-each select="m:math">
          <xsl:apply-templates mode="math"/>
        </xsl:for-each>
      </fotex:displaymath>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

We convert a <simplemath> element to a normal math element.

```
<xsl:template match="simplemath">
  <m:math><m:mi><xsl:apply-templates/></m:mi></m:math>
</xsl:template>
```

7.14 Other elements

Translation of <head>. We look at the parent. If it is a division (it starts with 'div', or is a <module>) we do nothing (already done), otherwise, we just output it. In the case of the new DTD, this is called <bodyTitle> and nothing is done if the parent is <subsection> or if the grand-father of the parent is <raweb>.

```
<xsl:template match="head" priority="10">
  <xsl:variable name="parent" select="name(..)"/>
  <xsl:if test="not(starts-with($parent,'div')) and not($parent = 'module')">
    <xsl:apply-templates/>
  </xsl:if>
</xsl:template>
```

We copy all <?xmltex?> instructions.

```
<xsl:template match="processing-instruction()[name()='xmltex']" >
  <xsl:message>xmltex pi <xsl:value-of select="."/></xsl:message>
  <xsl:copy-of select="."/>
</xsl:template>
```

Is this needed?

```
<xsl:variable name="processor">
   <xsl:value-of select="system-property('xsl:vendor')"/>
</xsl:variable>
```

This is defined like in other style sheets.

```
<xsl:variable name="LeTypeProjet">
   <!-- see elsewhere -->
</xsl:variable>
```

Same as above.

```
<xsl:variable name="year">
  <!-- see elsewhere -->
</xsl:variable>
```

This could be: 'Team Foo'.

```
<xsl:variable name="PRID">
 <xsl:value-of select="$LeTypeProjet"/>
 <xsl:text> </xsl:text>
 <xsl:value-of select="/raweb/accueil/projet"/>
</xsl:variable>
```

This creates the first page. We start with a <fo:INRIA> special element, giving it the current year as year attribute. This will insert the logo. After that, we have a link to the Team's home page, with the short and long name. We have a link to each UR. We have also a special element <fo:RATHEME> that completes our page hacking. For the new DTD, replace attribute html by id, element <projet> by <shortname>, element <projetdeveloppe> by <projectName>, replace UR/* in the for-each by UR, and for each <UR> element, replace the attribute nom by name and url (which disappeared) by a value that depends on the name attribute.

```
<xsl:template  name="accueil.body"/>
   <fo:INRIA year="{$year}"/>
   <fo:block font-size= "25pt" text-align="center">
     <fo:basic-link external-destination="http://www.inria.fr/recherche/
         equipes/{@html}.en.html">
        <xsl:value-of select="$LeTypeProjet"/> <xsl:text> </xsl:text>
        <xsl:value-of select="projet"/>
     </fo:basic-link>
   </fo:block>
    <fo:block font-size= "25pt" text-align="center" space-before="1cm">
          <xsl:value-of select="projetdeveloppe"/>
    </fo:block>
   <fo:block font-size= "17.28pt" text-align="center" space-before="1cm">
     <xsl:for-each select ="UR/*">
        <fo:basic-link external-destination ="{@url}">
          <xsl:value-of select="@nom"/>
          <xsl:if test="position()!=last()"> - </xsl:if>
        </fo:basic-link>
     </xsl:for-each>
```

```
1823      </fo:block>
1824     <fo:block font-size= "10pt" font-style="normal"
1825         font-family="Helvetica" text-align="center" space-before="1cm">
1826         <fo:RATHEME><xsl:value-of select="theme"/></fo:RATHEME>
1827      </fo:block>
1828  </xsl:template>
```

A `<moreinfo>` is typeset in italics.

```
1829  <xsl:template match="moreinfo">
1830    <fo:block font-style="italic"> <xsl:apply-templates/> </fo:block>
1831  </xsl:template>
```

This adds an id.

```
1832  <xsl:template name="addID">
1833     <xsl:attribute name="id">
1834       <xsl:call-template name="idLabel"/>
1835     </xsl:attribute>
1836  </xsl:template>
```

These rules have been added so that TEX and LATEX are nicely printed.

```
1837  <xsl:template match="TeX">
1838     <TeX/>
1839  </xsl:template>
1840  <xsl:template match="LaTeX">
1841     <LaTeX/>
1842  </xsl:template>
```

This is a priori useless.

```
1843  <xsl:template match="anchor">
1844    <fo:inline>
1845       <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
1846    </fo:inline>
1847  </xsl:template>
```

Translation of `<keywords>`. We take all the `<term>` elements.

```
1848  <xsl:template match="keywords">
1849   <fo:block space-after.optimum="4pt">
1850     <fo:inline>
1851       <xsl:attribute name="font-weight">bold</xsl:attribute>
1852       <xsl:value-of select="./@titre"/>
1853     </fo:inline>
1854     <xsl:for-each select="term">
1855        <xsl:apply-templates select="."/> <xsl:call-template name="separateur.objet"/>
1856        </xsl:for-each>
1857   </fo:block>
1858  </xsl:template>
```

Keywords in the new DTD. We have a list of `<keyword>` elements. The default behavior is to omit them. In some cases, we output then in italic font style, with a comma as separator.

```
1859  <xsl:template match="keyword" mode="section">
1860   <fo:inline font-style="italic">
1861      <xsl:apply-templates/>
1862   </fo:inline>
1863   <xsl:call-template name="separateur.objet"/>
1864  </xsl:template>
```

```
1865
1866 <xsl:template match="keyword"/>
```

This means: if there is a keyword, we output them all, sorted in alphabetic order (this is strange), preceded by a title.

```
1867 <xsl:template name="keywords-list2">
1868   <xsl:if test="./keyword">
1869     <fo:block space-after.optimum="4pt">
1870        <fo:inline>
1871          <xsl:attribute name="font-weight">bold</xsl:attribute>
1872          Keywords:
1873        </fo:inline>
1874        <xsl:apply-templates select="(./keyword)" mode="section" >
1875          <xsl:sort />
1876        </xsl:apply-templates>
1877       </fo:block>
1878    </xsl:if>
1879 </xsl:template>
```

This code is similar, it is applied to a module (a `<subsection>` that has no `<subsection>` as ancestor). We output all keywords, of the module and its submodules.

```
1880 <xsl:template name="keywords-list">
1881   <xsl:if test=".//keyword">
1882     <fo:block space-after.optimum="4pt">
1883       <fo:inline>
1884         <xsl:attribute name="font-weight">bold</xsl:attribute>
1885         Keywords:
1886       </fo:inline>
1887       <xsl:apply-templates select="(.//keyword)" mode="section" >
1888         <xsl:sort />
1889       </xsl:apply-templates>
1890     </fo:block>
1891   </xsl:if>
1892 </xsl:template>
```

These include files are describe in the next section.

```
1893 <xsl:include href="raweb3-table.xsl"/>
1894 <xsl:include href="raweb3-makecolspec.xsl"/>
```

This is the end of the file.

```
1895 </xsl:stylesheet>
```

7.15 Computing column specifications

We consider here a file that starts like this. This uses extensions to XSLT. It is taken from the TEI distribution, see Copyright notice in the preceding chapter.

```
1896 <?xml version="1.0" encoding="utf-8"?>
1897 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
1898   xmlns:fotex="http://www.tug.org/fotex"
1899   xmlns:exsl="http://exslt.org/common"
1900   exclude-result-prefixes="saxon exsl"
1901   extension-element-prefixes="saxon exsl fotex"
```

```
1902    xmlns:saxon="http://icl.com/saxon"
1903    xmlns:fo="http://www.w3.org/1999/XSL/Format" version="1.0">
```

The computations are the following. For each cell in the table, we construct a variable, named `$stuff`, that contains the value of the cell (some templates are applied). We construct a `<cell>` element, that has a `<col>` attribute whose value is the position of the cell; hence this algorithm does not work in the case where the cell spans multiple columns. The content of the cell is its width (i.e., is the width of the cell to typeset); this is hard to compute (consider two cells, one with an image of size 10cm, another one with a caption for it). In order to make this guess more usable, we added 100[6]. All these cells are put in a variable `$tds`. Consider for instance the case where the cells are of width 1, 2, 3, and 4, there are two rows, the first two cells are in the first row. We compute the total, namely 10. The columns totals are 4 and 6, so that the width of the columns are 40% and 60%; these two quantities are computed by sorting the cells by column. For each cell that is the first (has no preceding sibling) we compute the sum of the following siblings. In the new DTD, `<cell>` is replaced by `<td>`, etc.

```
1904  <xsl:template name="calculateTableSpecs">
1905    <xsl:variable name="tds">
1906       <xsl:for-each select=".//cell">
1907          <xsl:variable name="stuff">
1908             <xsl:apply-templates/>
1909          </xsl:variable>
1910          <cell>
1911              <xsl:attribute name="col"><xsl:number/></xsl:attribute>
1912              <xsl:value-of select="string-length($stuff) + 100"/>
1913          </cell>
1914       </xsl:for-each>
1915    </xsl:variable>
1916    <xsl:variable name="total">
1917        <xsl:value-of select="sum(exsl:node-set($tds)/cell)"/>
1918    </xsl:variable>
1919    <xsl:for-each select="exsl:node-set($tds)/cell">
1920       <xsl:sort select="@col" data-type="number"/>
1921       <xsl:variable name="c" select="@col"/>
1922       <xsl:if test="not(preceding-sibling::cell[$c=@col])">
1923         <xsl:variable name="len">
1924            <xsl:value-of select="sum(following-sibling::cell[$c=@col]) + current()"/>
1925         </xsl:variable>
1926         <xsl:text>
1927         </xsl:text>
1928         <fo:table-column column-number="{@col}"
1929             fotex:column-align="L" column-width="{$len div $total * 100}%" />
1930       </xsl:if>
1931    </xsl:for-each>
1932    <xsl:text></xsl:text> <!-- this is a newline character -->
1933  </xsl:template>
1934
1935
1936  </xsl:stylesheet>
```

[6] Why not use a random number? there should be a parameter in Tralics that inhibits this template.

7.16 Converting cells

This is another file. It is taken from the TEI distribution, see Copyright notice in the preceding chapter.

```
<xsl:stylesheet
  xmlns:fotex="http://www.tug.org/fotex"
  xmlns:exsl="http://exslt.org/common"
  exclude-result-prefixes="exsl"
  extension-element-prefixes="exsl"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"  >
```

The translation of `<cell>` is `<fo:table-cell>`. If the cols or rows attributes are at least two, we add corresponding attributes to the result. If the cell, or its parent has a role attribute that says it is a label, we change the font to bold. Other attribute can also be given, see below. Changes for the new DTD: the element is called `<td>` or `<th>`. The test for the attribute role has been replaced by the test, is the current element `<th>`. If the ancestor is not `<object>`, then border-xx-style is set to solid (where xx is any of 'start', 'end', 'before', and 'after'. .

```
<xsl:template match="cell">
  <fo:table-cell>
    <xsl:if test="@cols &gt; 1">
      <xsl:attribute name="number-columns-spanned">
        <xsl:value-of select="@cols"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="@rows &gt; 1">
      <xsl:attribute name="number-rows-spanned">
        <xsl:value-of select="@rows"/>
      </xsl:attribute>
    </xsl:if>
   <xsl:call-template name="cellProperties"/>
   <fo:block>
    <xsl:choose>
      <xsl:when test="@role='label' or parent::row[@role='label']">
        <xsl:attribute name="font-weight">bold</xsl:attribute>
      </xsl:when>
    </xsl:choose>
    <xsl:apply-templates/>
   </fo:block>
  </fo:table-cell>
</xsl:template>
```

These are the possible cell properties. If the role is 'hi', we set the background color to 'silver'. In the case where the parent table has rend = 'frame', we add a frame around the cell. Only the first row has a border-before, only the last cell in the row has a border-end. Otherwise, we look at the properties of the cell, or of the row. In the new DTD, we should replace `<row>` by `<tr>`.

```
<xsl:template name="cellProperties" >
  <xsl:if test="@role='hi'">
    <xsl:attribute name="background-color">silver</xsl:attribute>
  </xsl:if>

  <xsl:choose>
    <xsl:when test="ancestor::table[1][@rend='frame']">
```

```
1974        <xsl:if test="not(parent::row/preceding-sibling::row)">
1975          <xsl:attribute name="border-before-style">solid</xsl:attribute>
1976        </xsl:if>
1977        <xsl:attribute name="border-after-style">solid</xsl:attribute>
1978        <xsl:if test="not(following-sibling::cell)">
1979          <xsl:attribute name="border-end-style">solid</xsl:attribute>
1980        </xsl:if>
1981        <xsl:attribute name="border-start-style">solid</xsl:attribute>
1982      </xsl:when>
1983      <xsl:otherwise>
1984        <xsl:if test="@left-border='true'">
1985          <xsl:attribute name="border-start-style">solid</xsl:attribute>
1986        </xsl:if>
1987        <xsl:if test="@right-border='true'">
1988          <xsl:attribute name="border-end-style">solid</xsl:attribute>
1989        </xsl:if>
1990        <xsl:if test="ancestor::row/@top-border='true'">
1991          <xsl:attribute name="border-before-style">solid</xsl:attribute>
1992        </xsl:if>
1993        <xsl:if test="ancestor::row/@bottom-border='true'">
1994          <xsl:attribute name="border-after-style">solid</xsl:attribute>
1995        </xsl:if>
1996        <xsl:if test="ancestor::row/@bottom-border='true'">
1997          <xsl:attribute name="border-after-style">solid</xsl:attribute>
1998        </xsl:if>
1999        <xsl:if test="@bottom-border='true'">
2000          <xsl:attribute name="border-after-style">solid</xsl:attribute>
2001        </xsl:if>
2002        <xsl:if test="@top-border='true'">
2003          <xsl:attribute name="border-before-style">solid</xsl:attribute>
2004        </xsl:if>
2005      </xsl:otherwise>
2006    </xsl:choose>
```

If the table is not tight, we add some padding to the cell.

```
2007    <xsl:if test="not(ancestor::table/@rend='tight')">
2008      <xsl:attribute name="padding">
2009         <xsl:value-of select="$tableCellPadding"/>
2010      </xsl:attribute>
2011    </xsl:if>
```

In the case where the cell has an horizontal alignment property we use it. Otherwise, we look at some database, and try to extract that information. This is not used by the Raweb.

```
2012    <xsl:choose>
2013      <xsl:when test="@halign">
2014        <xsl:attribute name="text-align">
2015           <xsl:value-of select="@halign"/>
2016        </xsl:attribute>
2017      </xsl:when>
2018      <xsl:otherwise>
2019        <xsl:variable name="thiscol">
2020          <xsl:value-of select="position()"/>
2021        </xsl:variable>
```

```
      <xsl:variable name="tid">...</xsl:variable>
      <xsl:variable name="align">... </xsl:variable>
      <xsl:choose>
          <xsl:when test="$align='R'">
                <xsl:attribute name="text-align">right</xsl:attribute>
          </xsl:when>
          <xsl:when test="$align='L'">
                <xsl:attribute name="text-align">left</xsl:attribute>
          </xsl:when>
          <xsl:when test="$align='C'">
                <xsl:attribute name="text-align">center</xsl:attribute>
          </xsl:when>
          <xsl:otherwise>
            <xsl:if test="not($align='')">
               <xsl:attribute name="text-align">
                   <xsl:value-of select="$align"/>
               </xsl:attribute>
            </xsl:if>
          </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This is the end of the file.

```
</xsl:stylesheet>
```

7.17 Customisation

We describe here the file raweb3-param.xsl. This is an adaptation of a file provided by the TEI. We changed the order of items, indicated which parameters are used, or unused.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

Vertical space before and after a list. The `$exampleMargin` is used as left and right margins for figure captions.

```
<xsl:param name="listAbove-1">6pt</xsl:param>
<xsl:param name="listBelow-1">6pt</xsl:param>
<xsl:param name="listAbove-2">4pt</xsl:param>
<xsl:param name="listBelow-2">4pt</xsl:param>
<xsl:param name="listAbove-3">0pt</xsl:param>
<xsl:param name="listBelow-3">0pt</xsl:param>
<xsl:param name="listAbove-4">0pt</xsl:param>
<xsl:param name="listBelow-4">0pt</xsl:param>
<xsl:param name="exampleMargin">12pt</xsl:param>
```

Quantities use to define list markers. U+2219 is bullet operator (defined to be `\ensuremath {\bullet}` in ucharacters.sty) is en dash (`\textendash`), U+002A is asterisk and U+002B is plus sign.

```
<xsl:param name="bulletOne">&#x2219;</xsl:param>
<xsl:param name="bulletTwo">&#x2013;</xsl:param>
```

```
<xsl:param name="bulletThree">&#x002A;</xsl:param>
<xsl:param name="bulletFour">&#x002B;</xsl:param>
```

Page dimensions and margins. The default values are used, except for the right margin (we use the same value as for the left margin). We use a single column. There is a template named `hookDefinepagemasters` that does nothing. This could be redefined to do some action. Idem for `blockStartHook`.

```
<xsl:param name="pageWidth">210mm</xsl:param>
<xsl:param name="pageHeight">297mm</xsl:param>
<xsl:param name="regionAfterExtent">12pt</xsl:param>
<xsl:param name="regionBeforeExtent">12pt</xsl:param>
<xsl:param name="bodyMarginBottom">24pt</xsl:param>
<xsl:param name="bodyMarginTop">24pt</xsl:param>
<xsl:param name="pageMarginTop">75pt</xsl:param>
<xsl:param name="pageMarginBottom">100pt</xsl:param>
<xsl:param name="pageMarginLeft">80pt</xsl:param>
<xsl:param name="pageMarginRight">150pt</xsl:param>
<xsl:param name="columnCount">1</xsl:param>
<xsl:template name="hookDefinepagemasters"/>
<xsl:template name="blockStartHook"/>
```

Fonts. The body and the headers use Times as font family. The two other fonts are used for typesetting `<code>` and `<ident>`, bu these two elements are not created by Tralics.

```
<xsl:param name="bodyFont">Times Roman</xsl:param>
<xsl:param name="divFont">Times Roman</xsl:param>
<xsl:param name="typewriterFont">Computer-Modern-Typewriter</xsl:param>
<xsl:param name="sansFont">Helvetica</xsl:param>
```

Font sizes. The main font size is 10pt, the quantity `$footnoteSize` is used for footnotes (8pt) and `$footnotenumSize` for footnote markers (is 7pt not too small?). We define a negative indentation for the section titles: `$headingOutdent`, but a positive paragraph indentation. Note: the first paragraph in a division is not indented.

```
<xsl:param name="bodyMaster">10</xsl:param>
<xsl:param name="bodySize">
 <xsl:value-of select="$bodyMaster"/><xsl:text>pt</xsl:text>
</xsl:param>
<xsl:param name="footnoteSize">8pt</xsl:param>
<xsl:param name="footnotenumSize">7pt</xsl:param>
<xsl:param name="headingOutdent">-3em</xsl:param>
<xsl:param name="parIndent">1em</xsl:param>
<xsl:param name="parSkip">0pt</xsl:param>
<xsl:param name="parSkipmax">12pt</xsl:param>
```

The following is used for a section title. We use a bold 18pt font. There is some space before and after: 12pt and 6pt.

```
<xsl:template name="setupDiv0">
 <xsl:attribute name="font-size">18pt</xsl:attribute>
 <xsl:attribute name="text-align">left</xsl:attribute>
 <xsl:attribute name="font-weight">bold</xsl:attribute>
 <xsl:attribute name="space-after">6pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">12pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

The following is used for a subsection (module) title. We use a bold 14pt font. There is some space before and after, but less than for a section: 9pt and 3pt.

```
<xsl:template name="setupDiv1">
 <xsl:attribute name="font-size">14pt</xsl:attribute>
 <xsl:attribute name="text-align">left</xsl:attribute>
 <xsl:attribute name="font-weight">bold</xsl:attribute>
 <xsl:attribute name="space-after">3pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">9pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

The following is used for a subsubsection title. We use a bold 12pt font. There is some space before and after, but less than for a subsection: 4pt and 2pt.

```
<xsl:template name="setupDiv2">
 <xsl:attribute name="font-size">12pt</xsl:attribute>
 <xsl:attribute name="text-align">left</xsl:attribute>
 <xsl:attribute name="font-weight">bold</xsl:attribute>
 <xsl:attribute name="font-style">italic</xsl:attribute>
 <xsl:attribute name="space-after">2pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">4pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

The following is used for a paragraph title. We use an italic 10pt font.

```
<xsl:template name="setupDiv3">
 <xsl:attribute name="font-size">10pt</xsl:attribute>
 <xsl:attribute name="text-align">left</xsl:attribute>
 <xsl:attribute name="font-style">italic</xsl:attribute>
 <xsl:attribute name="space-after">0pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">4pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

The following is used for a subparagraph title. We use a 10pt normal font.

```
<xsl:template name="setupDiv4">
 <xsl:attribute name="font-size">10pt</xsl:attribute>
 <xsl:attribute name="space-before.optimum">4pt</xsl:attribute>
 <xsl:attribute name="text-indent"><xsl:value-of select="$headingOutdent"/></xsl:attribute>
</xsl:template>
```

This contructs the caption of a figure. It is centered, with some indentation on both sides.

```
<xsl:template name="figureCaptionstyle">
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="font-style">italic</xsl:attribute>
  <xsl:attribute name="end-indent">
   <xsl:value-of select="$exampleMargin"/>
  </xsl:attribute>
  <xsl:attribute name="start-indent">
   <xsl:value-of select="$exampleMargin"/>
  </xsl:attribute>
</xsl:template>
```

The template here could be used for table captions. If we compare with figures above, we see that there is some vertical space before and after the caption.

```
<xsl:param name="tableCaptionAlign">center</xsl:param>
<xsl:param name="spaceAroundTable">8pt</xsl:param>
<xsl:param name="spaceBelowCaption">4pt</xsl:param>
```

```
<xsl:param name="tableAlign">center</xsl:param>
<xsl:param name="tableCellPadding">2pt</xsl:param>

<xsl:template name="tableCaptionstyle">
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="font-style">italic</xsl:attribute>
  <xsl:attribute name="end-indent">
   <xsl:value-of select="$exampleMargin"/>
  </xsl:attribute>
  <xsl:attribute name="start-indent">
   <xsl:value-of select="$exampleMargin"/>
  </xsl:attribute>
        <xsl:attribute name="space-before">
          <xsl:value-of select="$spaceAroundTable"/>
        </xsl:attribute>
  <xsl:attribute name="space-after">
          <xsl:value-of select="$spaceBelowCaption"/>
  </xsl:attribute>
  <xsl:attribute name="keep-with-next">always</xsl:attribute>
</xsl:template>
```

Parameters for the TOC. What is `$tocSize`? the font size of the title of the toc? we use 14pt, see above. The two variables `$headingNumberSuffix` and `$tocNumberSuffix` explain that we should put a period-plus-space after numbers in the toc and headings. The variable `$numberHeadings` says that we want something to appear in the TOC, and `$numberDepth` says that we want in fact everything (in the HTML version, the TOC contains only sections and subsections, for the Pdf, we accept everything. In general it fits on a double page.)

```
<xsl:param name="tocSize">16pt</xsl:param>
<xsl:param name="div0Tocindent">0in</xsl:param>
<xsl:param name="div1Tocindent">0.25in</xsl:param>
<xsl:param name="div2Tocindent">0.5in</xsl:param>
<xsl:param name="div3Tocindent">0.75in</xsl:param>
<xsl:param name="div4Tocindent">1in</xsl:param>
<xsl:param name="headingNumberSuffix">. </xsl:param>
<xsl:param name="tocNumberSuffix">. </xsl:param>
<xsl:param name="numberHeadings">true</xsl:param>
<xsl:param name="numberDepth">9</xsl:param>
```

Parameters for lists.

```
<xsl:param name="listRightMargin">10pt</xsl:param>
<xsl:param name="listNormalIndent">15pt</xsl:param>
<xsl:param name="listLeftGlossIndent">7mm</xsl:param>
<xsl:param name="listLeftGlossInnerIndent">4mm</xsl:param>
<xsl:param name="listLeftIndent">5pt</xsl:param>
<xsl:param name="listItemsep">4pt</xsl:param>
```

Four names, useful if we decide to switch from English to another language. We use only 'Figure' and 'Table'. Note the space at the end; there is no space for the other words.

```
<xsl:param name="figureWord">Figure </xsl:param>
<xsl:param name="tableWord">Table </xsl:param>
<xsl:param name="appendixWords">Appendix</xsl:param>
<xsl:param name="biblioWords">Bibliography</xsl:param>
```

Overwritten by the Raweb.

```
<xsl:param name="linkColor">black</xsl:param>
<xsl:param name="pdfBookmarks"></xsl:param>
```

Not used by the Raweb. The variable `$autoScaleFigures` is not used: scaling must be explicit. The variable `$graphicsPrefix` is not used, because we assume that images are in the current directory, and there is no need to add a prefix. The variable `$smallSize` is not not used (see page 243 and following how smaller and larger fonts are used). The quantities `$authorSize`, `$dateSize` and `$titleSize` could be used for the title page. In the case of the Raweb, the title page is created by the code on page 59, this defines fonts and size.

```
<xsl:param name="autoScaleFigures"></xsl:param>
<xsl:param name="graphicsPrefix"></xsl:param>
<xsl:param name="smallSize">
 <xsl:value-of select="$bodyMaster * 0.9"/><xsl:text>pt</xsl:text>
</xsl:param>
<xsl:param name="authorSize">14pt</xsl:param>
<xsl:param name="dateSize">14pt</xsl:param>
<xsl:param name="titleSize">16pt</xsl:param>
```

The variable `$hyphenate` is unused: we always hyphenate. The variable `$alignment` is unused: we always justify. The `$language` variable is unused.

```
<xsl:param name="hyphenate">true</xsl:param>
<xsl:param name="alignment">justify</xsl:param>
<xsl:param name="language">US</xsl:param>
```

This defines sizes for quantities that are not used here. The quantities `$runSize` and `$runFont` are for running heads. We use the default font.

```
<xsl:param name="runSize">9pt</xsl:param>
<xsl:param name="runFont">sans-serif</xsl:param>
```

These are unused.

```
<xsl:param name="flowMarginLeft"></xsl:param>
<xsl:param name="giColor">orange</xsl:param>
<xsl:param name="identColor">blue</xsl:param>
<xsl:param name="activeLinebreaks"></xsl:param>
<xsl:param name="activePagebreaks"></xsl:param>
<xsl:param name="tocFront">true</xsl:param>
<xsl:param name="tocBack">true</xsl:param>
<xsl:param name="tocStartPage">1</xsl:param>
<xsl:param name="exampleSize">
   <xsl:value-of select="$bodyMaster * 0.8"/><xsl:text>pt</xsl:text>
</xsl:param>
<xsl:param name="exampleBefore">4pt</xsl:param>
<xsl:param name="exampleAfter">4pt</xsl:param>
<xsl:param name="OUCS"></xsl:param>
<xsl:param name="titlePage"></xsl:param>
<xsl:param name="divRunningheads"></xsl:param>
<xsl:param name="forcePageMaster"></xsl:param>
<xsl:param name="twoSided">true</xsl:param>
<xsl:param name="frontMulticolumns"></xsl:param>
<xsl:param name="bodyMulticolumns"></xsl:param>
<xsl:param name="backMulticolumns"></xsl:param>
<xsl:param name="sectionHeaders">true</xsl:param>
<xsl:param name="formatFrontpage">i</xsl:param>
<xsl:param name="formatBodypage">1</xsl:param>
<xsl:param name="formatBackpage">1</xsl:param>
<xsl:param name="formatAppendix">A.1.</xsl:param>
```

```
<xsl:param name="numberBackHeadings">A.1</xsl:param>
<xsl:param name="numberFrontHeadings">1</xsl:param>
<xsl:param name="captionInlinefigures"></xsl:param>
<xsl:param name="xrefShowTitle"></xsl:param>
<xsl:param name="xrefShowHead"></xsl:param>
<xsl:param name="xrefShowPage"></xsl:param>
<xsl:param name="minimalCrossRef"/>
<xsl:param name="numberHeadingsDepth">9</xsl:param>
<xsl:param name="prenumberedHeadings"></xsl:param>
<xsl:param name="biblSize">16pt</xsl:param>
<xsl:param name="indentBibl">1em</xsl:param>
<xsl:param name="spaceBeforeBibl">4pt</xsl:param>
<xsl:param name="spaceAfterBibl">0pt</xsl:param>
<xsl:param name="inlineTables"></xsl:param>
<xsl:param name="rowAlign">left</xsl:param>
<xsl:param name="makeTableCaption">true</xsl:param>
<xsl:param name="readColSpecFile"></xsl:param>
```

This is the end of the file.

```
</xsl:stylesheet>
```

Chapter 8

Application to other examples

We explain in this chapter how Tralics can be used to convert general documents into HTML. Two examples are considered, the Tralics documentation and the thesis of C. Roméro.

One non-trivial point of the thesis is that it contains trees: these are formed of text, connected by straight lines, or other curves, see figure 8.1. The original Pdf version of the thesis was obtained by converting a dvi file to a PostScript file, adding PostScript commands for the connections, and converting everything to Pdf; an alternate solution would be to use pdfLaTeX, and the pdftricks package: this package automatically extracts the trees, writes them to an external file, uses LaTeX for obtaining a dvi file, which is converted in PostScript and Pdf, as explained above, and includes each tree as an image. The same idea will be used: the document is converted to XML, trees are automatically extracted, converted to dvi, PostScript, then png, then re-inserted as images.

In the simple case, we use a shell script like the following

```
#!/bin/bash
SGML_CATALOG_FILES=./catalog
export SGML_CATALOG_FILES
FILE=$1
tralics ${FILE} -noentnames -nostraightquotes -nozerowidthspace\
   -trivialmath=7 -usequotes
xsltproc --catalogs  --stringparam Main ${FILE} -o ${FILE}.html \
     ${FILE}html.xsl ${FILE}.xml
```

The idea is to convert a file, for instance foo.tex, that may input other source files, into foo.xml using Tralics (options will be explained later) and to convert this into foo.html, using the XML processor `xsltproc`; this second conversion depends on a style sheet, here it is foohtml.xsl (note how this depends on the initial file name).

Our examples do not use a configuration file, as a consequence the DTD will be 'std', this is a variant of the Raweb DTD, details are not important in this particular example, the catalog file explains where to find this DTD. In the case of the thesis, the shell script shown above can be applied (but trees are not fully rendered), but you can use a Perl script that proceeds exactly as above, but modifies the output of Tralics before converting it to HTML.

The following style sheets, that are part of thge Tralics distribution, will be used and explained:

- TR1, TR2: (real name tralics-rr1html.xsl and tralics-rr2html.xsl). These two files are identical, and will be denoted by TR. They are used to convert the Tralics documentation, which is a sequence of two Inria Research Reports.
- TF, TE: (real name thesehtml.xsl and thesishtml.xsl). These two files are used to convert the thesis (original French version, and its English translation).

- TC: (real name thesis.xsl), common code for French and English version of the thesis.
- CLS: (real name cls.xsl), common code for all files shown above.

This chapter describes the following points: the modifications to LaTeX source needed by Tralics, the style sheets used for the XML to HTML conversion, and the Perl script mentioned above.

8.1 Modifying the source document

If you have some complex document like a PhD thesis, that compiles well with LaTeX, trying to translate it into a beautiful HTML document using Tralics will often require non trivial changes to the document (or its environment, like style files or style sheets). Let's consider first the case of the research report (the document you are reading right now).

8.1.1 Case of the report

The foo.ult file is automatically loaded by Tralics, before anything else, when you translate foo.tex; you can redefine some commands that come from the LaTeX kernel, or use the `\AtBeginDocument` mechanism to overwrite commands defined later. We describe here the file tralics-rr.ult that corresponds to the first part of the Tralics documentation, the other file being a subset of this one. This is the start of the file.

```
%% -*- latex -*-
%$Id: tralicsR10.tex,v 1.21 2007/01/24 18:49:07 grimm Exp $
%% This is the ult file for tralics-rr.tex
```

The three lines shown here define two skip registers and an environment. Our document modifies the content of the registers, this changes the page layout (Tralics ignores such parameters) and redefines the environment: we add a comment at the start of the index, that says that page numbering could be off by one, because the `\index` command of a word appearing in a verbatim block is inserted before the environment. Since Tralics does not typeset the index (i.e., the `theindex` environment produced by the `makeindex` program), this comment is lost; it can be re-inserted in the XML to HTML transformation phase.

```
\newskip\footskip
\newskip\topmargin
\newenvironment{theindex}{}{}
```

Here we are faced to a big problem: we want to explain that an italic Delta is not the same as a normal Delta, and we do not know how to translate the character. We found the following solution: these two lines of code provide the definitions required for the text to compile; we extracted an image of the glyphs from the dvi file, and inserted them conditionally (with a comment that explains the problem).

```
\def\DeclareMathSymbol#1#2#3#4{}
\let\itDelta\Delta
```

Same problem as above. We try to illustrate another difference between LaTeX and MathML.

```
\let\mathbbm\mathbb
\let\mathscr\mathcal
```

These redefinitions are a bit annoying. Our document gives a usage of these commands and shows the result (explaining that this cannot be translated by Tralics). It is like the italic Delta above, but concerns text fonts rather than math characters.

```
\def\fontencoding{\textit{fontencoding U}}
\def\fontfamily{\textit{fontfamily U}}
```

```
\def\fontseries{\textit{fontseries U}}
\def\fontshape{\textit{fontshape U}}
\def\fontsize{\textit{fontesize U}}
\def\selectfont{\textit{selectfont U}}
```

The first command shown here redefines `\item` (this is required in the case of the thesis, and since we want to avoid code duplication, it is also needed here). The third line redefines a command that is defined in a macro file, and we have to delay its evaluation. The effect is to add an attribute to the main XML element; we shall explain later how it will be used. The point is that the two parts of the report must have different values.

```
\let\item\@item
\AtBeginDocument{
 \def\htmlprefix#1{\addattributestodocument{htmlprefix}{#1}}
```

For some reason, our document modifies the `\tabcolsep` length and the equation counter, these are quantities not managed by Tralics; a non-trivial problem is how to manage the layout of tables; currently, we use narrow tables, and sometimes add some space to widest element in each column. Note that the thesis starts with a list of tables, manually split across pages; they had to be edited, in order to improve the layout.

```
\newlength \tabcolsep
\newcounter{equation}
```

All web pages use an italic font for the Tralics name, and this document uses a sans-serif font; this explains the redefinition here. On the other hand, we prefer a normal font for the word 'Pdf' in the HTML version.

```
\def\Tralics{\textit{Tralics}}
\def\Pdf{Pdf}
```

We have no idea how to write the command `\cstok` that puts a box around its argument; it is used by Knuth in the TeXbook to represent a token; we underline the argument.

```
\def\cstok#1{\ul{#1}}
```

Here comes the big hack. Our document uses a lot of verbatim material, and we have chosen two different colors for LaTeX and XML code. As a consequence, we have removed all occurrences to the `\verb` command, and the short hands provided by the short-verb mechanism. This means that we had to replace a simple expression like `\verb+\'e\"e+` by `\LTC{\BS verb+\BS\apos e\BS \DQ e+}` (remember that Tralics may be called with options that make single and double quote character intelligent, i.e., appropriate for text, but are no more verbatim).

A lot of commands use internally the first two commands shown here; but because of our laziness, the main text uses the shorter names that follow. The `\xmlclor` command is used in a case where we want a non-verbatim `\xmlcode`[1], it is a no-op in LaTeX. The last two lines are modifications to the verbatim environment provided by Tralics; of course we had to change the verbatim environments, as explained in the first part of this report.

```
\def\latexcode#1{\xbox{latexcode}{#1}}
\def\xmlcode#1{\xbox{xmlcode}{#1}}}
\def\LTC#1{\xbox{latexcode}{#1}}
\def\XC#1{\xbox{xmlcode}{#1}}
\def\xmlcolor#1{\xbox{xmlcolor}{#1}}
\def\verbatim@hook{}
\def\verbatimnumberfont#1{\xbox{vbnumber}{#1}
} % end of \AtBeginDocument commands
```

The following two lines appear in the main text. They are useful for conditional compilation.

[1] Normal text font, but XML-like color

```
\newif\iftralics\tralicsfalse
\ifx\tralicsversion\undefined\else\tralicstrue\fi
```

We show here the definition of the `\image` command, and an example of use: there are two LaTeX commands that produce a delta character, whose Tralics translation is identical; we conditionally include an image showing both glyphs. Note that we do not use `\includegraphics`.

```
\def\image#1#2#3{\iftralics#1\xbox{rawimage}{#2}#3\fi}
%\image{This is shown in the dvi file as }{deltadelta.png}{.}
```

We had to make some further changes to the report: the document contains code, preceded by comments, and some names in the code are indexed. Thus Tralics, creates an anchor; these are positionned before the verbatim text. In some pages, there is an end-of-paragraph between the comment and the code, this is invisible. However, if the `\index` command is set before the end-of-paragraph, the anchor is inside the `<p>` element (this is good), otherwise outside (this gives an invalid HTML document). We removed these spurious empty lines.

8.1.2 Case of the thesis

We shall explain later how the title page had to be modified. We already explained that the tables at the start of the document had to be modified. Here is one modification:

```
Langues :\par
\iftralics \else\hspace{1cm}\fi
\begin{tabular}[t]{ll}
AC\Tonly{\quad} & Anglais Contemporain \\
AE & Anglais Elisab\'ethain \\
MA & Moyen-Anglais \\
VA & Vieil-Anglais \\
\end{tabular}
```

The intented purpose of the `\hspace` command is to shift the table to the right with respect to its title. But Tralics never puts tables in horizontal mode, thus inserts a `\par`, and this give a blank line. As a consequence, there is too much vertical space betwen the table and its title. This explains why the space is conditionally inserted. The `\Tonly` command is a no-op in LaTeX, in inserts a `\quad` in Tralics (there is not enough space between columns).

Here is another modification. The table was too big to fit on the page, and was replaced by two tables with a `\newpage` between them. The code was modified; Tralics sees a single table.

```
\begin{tabular}[t]{ll}
...
+gen & trait g\'enitif \\
\iftralics\else
\end{tabular}

\newpage
\iftralics \else\hspace{1cm}\fi
\begin{tabular}[t]{ll}
\fi
+irr\'eel & trait irr\'eel \\
...
\end{tabular}
```

The thesis contains constructions like `Mod$_{Necessity}$`. One of the parameters of Tralics (shown in the shell script above) tells it to translate this as if it were `\textsubscript`; this is a non-standard LaTeX command, a variant of `\textsuperscript`; the HTML translation is obviously `Mod <sub> Necessity </sub>`. In the French version, an e with acute accent is needed, but Tralics

does nothing special if commands like `\acute` appear in a Math formula. Hence, one of the modifications to the thesis was to replace such expressions by a command, that was conditionally defined:

```
\def\modneces{Mod$_{N\acute{e}cessit\acute{e}}$\xspace }
\iftralics\def\modneces{Mod\textsubscript{Nécessité}\xspace}\fi
```

The thesis contains a math expression like the following (we have shown only the first two lines of the table). It is currentlty impossible to put anything than text in a `\mbox` in a math formula. In the case where the `\mbox{X}` is the same as `\mbox{A}` B `\mbox{C}`, math expressions are allowed. This is not the case here because the math is hidden by the font change.

```
$\left. \begin{array}{c}
\mbox{<{\bf $\alpha$}we>} \\
\mbox{<c{\bf $\alpha$}n>} \\
\end{array}
\right \}$
```

We had to edit the code and replace it by:

```
$\left. \begin{array}{c}
\mbox{<}\alpha\mbox{we>} \\
\mbox{<c}\alpha\mbox{n>} \\
\end{array}
\right \}$
```

Finally, the thesis contains the following two lines:

```
\font\vag=cmoebx10 scaled 1200
{\vag g}
```

This code compiles perfectly well. However, the intented purpose was to use the character ezh, Unicode U+292. For this reason, we modified the source, replacing the first second line by `{\jgvagg}`, with the following definition.

```
\def\jgvagg{\vag g}
```

After this change, it was possible to conditionally redefine the command, for instance like this:

```
\def\jgvag{^^^^0292}
```

8.2 The Perl script for extracting trees

We explain here the content of a Perl script that converts a file into HTML, it replaces trees by images. This is the header of the file.

```
#! /usr/bin/perl
# -*- perl -*-
$empty =""; # hack
#$Id="\$ Id: extract-table.pl,v 2.1 2006/11/06 18:09:19 grimm Exp $empty";
package main;
use strict;
```

This software is part of the Tralics distribution and has the same license.

```
$::Id =~ /,v (\S*?) /;
print "extract-table.pl $1 Copyright INRIA/APICS 2006-2007, Jos\'e Grimm\n";
print "Licensed under the CeCILL Free Software Licensing Agreement\n";
```

We start with the some global variables. If the argument of the procedure is ‘foo’, then we create foo.xml, then foo_e.xml will contain all elements that must be converted into images, and foo_g.xml is the same file with elements replaced by image names. We make the assumption that

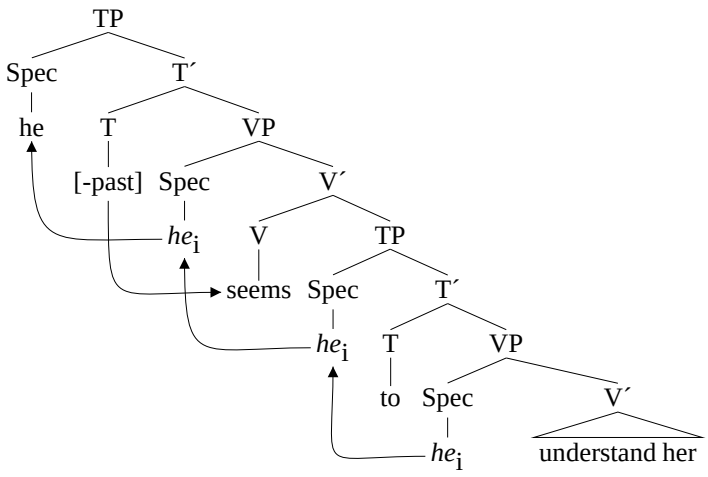


Figure 8.1: Example of a tree from the thesis of C. Roméro

the DTD is in the current directory, as well as the catalog (this explains in particular where the MathML DTD files are located).

```
$::name = shift;   ## argument of the script
$::name_o = $::name . ".xml";  ## tralics file
$::name_a = $::name . "_e.xml"; ## the trees
$::name_t = $::name . "_g.xml"; ## file to be converted to HTML
$::name_x = $::name . "html.xsl"; ## the style sheet
$::SGML_CATALOG_FILES= "./catalog";
$::xml_dir="../../xml";
$::pstoimg_cmd = "pstoimg"; ## pathname of pstoimg
$::use_convert = 0;  ## Use convert instead of pstoimg if non-zero
$::img_ctr = 0;
```

Note the arguments given to Tralics. The 'noentnames' options says to use character entities instead of entity names, so that an XML processor can ignore the DTD. Note that the DTD is needed for cross references. The second options tells Tralics to not use straight quotes, this is better in two cases, if the quote is used as an apostrophe, or when it is a right quote that should match a left quote. The next option tells Tralics not to use the zero-width space character, because some browsers show it as some unknown character, rather that some invisible object. We enable all math hacks. Finally, the last options tells Tralics to convert double quotes as quotations marks; said otherwise '`"foo"`' is interpreted in the same way as '`‘‘foo’’`' (giving '`« foo »`' in French).

```
$::tralics_options=
   "-noentnames -nostraightquotes -nozerowidthspace -trivialmath=7 -usequotes";
```

We should test if the script is called with at least one argument; an error is signaled if the file foo.tex does not exist (if no argument is given, we test for .tex that should not exist).

```
print "Working on $::name\n";
-f $::name . ".tex" or die ("No tex source");
```

We have a hacked version of fotex.sty, and we have to make sure that is found by LaTeX; this file is incompatible with the current version of fotex.xmt, this should be fixed, so that the old file

must be visible. The two files fotex-add.sty and raweb-uni.sty are needed so that we link them also. Finally, in order to convert an XML file via a driver named images.tex we must link raweb-cfg.sty to images.cfg.

```
sub link_files() {
  unlink("images.cfg");
  `ln -s $::xml_dir/raweb-cfg.sty images.cfg`;
  unlink("fotex-add.sty");
  `ln -s $::xml_dir/fotex-add.sty`;
  unlink("fotex.sty");
  `ln -s $::xml_dir/fotex.sty`;
  unlink("fotex.xmt");
  `ln -s $::xml_dir/fotex.xmt`;
  unlink("raweb-uni.sty");
  `ln -s $::xml_dir/raweb-uni.sty`;
}
```

This creates the driver file images.tex. It tells LaTeX to convert foo_e into a dvi file. We also create the hack file. This is like a configuration file (but the name 'images.cfg' is already in use). The first line says that the text width should be large enough (12cm is a bit small), and the second line says that we need the tree-dvips package. Finally we say that the hyperref package should not be loaded at all.

```
sub prepare_for_latex {
  open OUT,"> images.tex" || die "cannot create images.tex\n";
  print OUT "\\def\\xmlfile{$::name_a}\n";
  print OUT "\\def\\LastDeclaredEncoding{T1}\n";
  print OUT "\\input{xmltex.tex}\n";
  print OUT "\\end{document}\n";
  close OUT;
  open OUT,"> fotex-supp.tex" || die "cannot create fotex-supp.tex\n";
  print OUT "\\textwidth=17cm\n";
  print OUT "\\RequirePackage{tree-dvips}\n";
  print OUT "\\hyperreffalse\n";
  close OUT;
}
```

This is a routine that converts a file into a character string. This seems complicated, but it works, so that there is no reason why something else should be tried. The job is aborted in case where the XML file cannot be opened.

```
sub slurp_input {
    my($file) = @_;
    my(@file_string);
    open(INPUT, "<$file") or die "File $file cannot be opened";
    while (<INPUT>) {
        push(@file_string, $_ );
    };
    close INPUT;
    join('',@file_string);
}
```

This creates the XML file with only the trees. The style sheet will be explained below.

```
sub extract() {
  $ENV{SGML_CATALOG_FILES}= $::SGML_CATALOG_FILES;
```

```
  `xsltproc --catalogs -o  $::name_a extract.xsl $::name_o`;
}
```

We use a Perl script to replace `<table>` by `<img>`. You may wonder why this is not done by the style sheet. One problem is that we do not know the number associated to the image (because we cannot increment a counter like `img_ctr` in a style sheet). The second reason is that we might insert in the element attributes known only after conversion (for instance the size of the image), so that the style sheet extract.xsl used above cannot be used here. It is however important that the same elements are considered by both programs. The style sheet considers elements T, this is a `<table>` element that has a `<row>` child that has a `<cell>` child that has a `<node>` child. A `<preview>` element that has a T as a child is handled, and a T element is handled (unless it is the child of a preview). The function below is applied to all elements Y of name X, where X is table or preview. We replace the element if it is `<preview>`, or a table that contains a `<node>` somewhere. This works in most cases.

```
sub convert_table {
  my $x = $_[0];
  my $y = $_[1];
  if($x eq "preview" || $y =~ /<\/node>/) {
    $::img_ctr ++;
    return "<img src='images/tree_image_$::img_ctr.png' alt='Tree $::img_ctr'/>";
  } else { return $y; }
}
```

This functions reads the transcript file of LaTeX. Processing image number N gives a line of the form 'l2hSize :N:H::D::W', where W, H and D are dimensions; we convert them from TeX points to PostScript points by applying the magic factor 72bp =72.27pt. These three quantities are remembered in a global table. The value of 'align' is not used (it is middle if H=D and bottom if D=0, but in general no such condition is true, and the value of align is ignored). One important point is '$ps_counter', the number of images found. This has to be the same as the number of elements to be replaced.

```
sub read_log {
  my ($logfile) = @_;
  my ($name);
  my $TeXpt = 72/72.27;
  my $image_counter;
  open(LOG, "<$logfile") || die "\nCannot read logfile $logfile\n";
  while (<LOG>) {
    if (/latex2htmlSize|l2hSize/) {
      /:([^:]*):/;
      $name = $1;
      ++$image_counter;
      s/:([0-9.]*)pt/$::x_height{$name} = $1*$TeXpt;''/e;
      s/::([0-9.]*)pt/$::x_depth{$name} = $1*$TeXpt;''/e;
      s/::([0-9.]*)pt/$::x_width{$name} = $1*$TeXpt;''/e;
      $::x_align{$name} = "align = 'bottom'";
      if($::x_depth{$name}) { $::x_align{$name} = "align='middle'";}
    }
  }
  print STDOUT "Processing $image_counter images \n";
  $::ps_counter = $image_counter;
  close(LOG);
}
```

This piece of code is a bit tricky. The argument is a number, say 13, and we construct tmpdir/images013 which is the name of the PostScript file, that will be converted into a file named tree_image_13.png that we move into the images directory. For each image we have three dimensions, W, H and D. We compute H+D the total height (note how these numbers are converted into integers). The width and total height are parameters to the converter. After conversion, we use the `file` command in order to extract the size after conversion and store it in a table.

```
sub create_one_image {
  my $name = $_[0];
  my $im_name = "images";
  if($name<100) { $im_name .= "0";}
  if($name<10) { $im_name .= "0";}
  $im_name .= $name;
  my $w = $::x_width{$name};
  my $h = $::x_height{$name};
  my $d = $::x_depth{$name};
  $w = int($w + 0.6);
  $h = int($h + $d + 0.6);
  my $size = "-geometry ${w}x$h";
  my $cmd = "";
  print "$name ";
  if ($::use_convert) {
    $cmd = $::convert_cmd;
    $cmd .= " -crop ${w}x$h+64+44 ";
    $cmd .= " tmpdir/$im_name tree_image_$name.png ";
  } else {
    $cmd = $::pstoimg_cmd;
    $cmd .= " -type png -tmp tmpdir";
    $cmd .= " -discard -interlace -antialias";
    $cmd .= " -depth 1 -scale 1.4 $size";
    # marges 78,72 ou 72,72 ???
    $cmd .= " -margins 62,41 -crop abls -transparent";
    $cmd .= " -out tree_image_$name.png tmpdir/$im_name";
  }
  print LOG "$cmd\n";
  print LOG `$cmd`;

  my $info = `file tree_image_$name.png`;
  if($info =~ /PNG image data, (\d+) x (\d+),/) {
    $::x_align{$name} .= " width ='$1' height ='$2'";
  }
  `mv tree_image_$name.png images`;
}
```

This subroutine creates the temporary directory, call `dvips` and converts all images.

```
sub convert_to_png {
  `rm -rf tmpdir`;
  `mkdir tmpdir`;
  `rm -rf images`;
  `mkdir images`;
  my $DVIPSOPT="";
  my $dvips  = "dvips -S1 -i $DVIPSOPT -otmpdir/images ./images.dvi";
  open LOG, "> pstoimg.log";
```

```
137   print LOG `$dvips 2>&1`;
138   print "dvips done\n";
139   my $i;
140   foreach $i (1 .. $::ps_counter) { create_one_image ($i); }
141   print "\n";
142   `rm -rf tmpdir`;
143   close LOG;
144 }
```

This is the main routine. In case of trouble, you should first check that Tralics has correctly translated the input.

```
145 sub main()  {
146 print "Working on $::name\n";
147   `tralics $::name $::tralics_options`;
148   $::file = slurp_input($::name_o);
149   link_files();
150   prepare_for_latex();
151   print "Creating  $::name_a\n";
152   extract;
153   print "Running tex\n";
154   system("latex", "images");
155   read_log("images.log");
156   print "Converting images\n";
157   convert_to_png;
158   $::file =~ s!<(table|preview).*?</\1>!convert_table($1,$&)!egs;
159   open OUT,"> $::name_t" || die "cannot open $::name_t\n";
160   print OUT "$::file";
161   close OUT;
162   print "Seen $::img_ctr elements\n";
163   ($::ps_counter == $::img_ctr) or die "Wrong number of images";
164   print "xsltproc --catalogs  --stringparam Main $::name
165          -o $::name.html  $::name_x $::name_t\n";
166   `xsltproc  --stringparam Main $::name --catalogs
167     -o $::name.html  $::name_x $::name_t`
168 }
169 main;
```

For the thesis of C. Roméro, we have the following statistics. The runtime of Tralics and the style sheets is between one and two seconds. The time needed by LaTeX to convert the 130 images is over two minutes (nearly one second per image), and `pstoimg` is even slower (the total time is over five minutes). The XML file produced by Tralics has 1323K, that is reduced to 995K after replacement of tables by images. The XML file containing the images has a size of 920K, this is much more than the the difference of the two sizes given above. The essential reason is that the XSLT processor reads the DTD file, and replaces missing attributes by their value. For instance, an empty cell is translated into

```
<cell halign='center'/>
```

and then converted to

```
<cell halign="center" role="data" rows="1" cols="1" right-border="false"
left-border="false" top-border="false" bottom-border="false"/>
```

It should be possible to divide the size by two, hoping that the LaTeX runtime is also divided by two.

8.3 The style sheet for extracting trees

We give here a style sheet whose purpose is to extract some elements and convert them into images. In a first approximation, we select all `<table>` or `<preview>` elements containing a `<node>`, and handle the connectors. Details are given later.

This is the start of the file extract.xsl. We do not show the attributes of the style sheet elements, it is the same as for other files.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet>
<!--
   Copyright INRIA/APICS 2006-2007, Jos\'e Grimm
   Licensed under the CeCILL Free Software Licensing Agreement
-->
```

All connectors are in a dummy cell; if we do not take care, this cell produces a huge amount a space. The good solution would be to modify the LaTeX source, using a real element instead of this dummy cell. The temporary solution consists in ignoring spaces in cells, rows, tables.

```
<xsl:strip-space elements="cell row table"/>
<xsl:output method='xml' encoding='iso-8859-1'/>
```

In order to convert the XML file created here into dvi, it suffices to put a `<fo:block>` in a `<fo:root>`. We do something for each `<table>` and `<preview>` element.

```
<xsl:template match="std">
  <fo:root>
    <fo:block>
      <xsl:apply-templates select ="//table | //preview "/>
    </fo:block>
  </fo:root>
</xsl:template>
```

In the case of the `<table>` element, if it contains a a `<row>` that contains a `<cell>` that contains a `<node>`, we construct a `<tree>` element, containing the table, translated via a template described below. A newline character is inserted in the XML file, to separate elements.

```
<xsl:template match="table">
  <xsl:if test="row/cell/node">
    <xsl:text>&#xA;</xsl:text>
    <tree>
      <xsl:call-template name="table" select="."/>
    </tree>
  </xsl:if>
</xsl:template>
```

In the case of the `<preview>` element, if it contains a `<table>` that contains a `<row>` that contains a `<cell>` that contains a `<node>`, we construct a `<tree>` element, containing the `<table>` in preview mode. The important point here is that we copy the attributes.

```
<xsl:template match="preview">
  <xsl:if test="table/row/cell/node">
     <xsl:text>&#xA;</xsl:text>
    <tree>
       <xsl:copy-of select="@*"/>
       <xsl:apply-templates mode="preview"/>
    </tree>
```

```
  </xsl:if>
</xsl:template>
```

If a `<table>` is in a `<preview>`, but the previous rule does not match, we do nothing.

```
<xsl:template match="preview/table"/>
```

Translating the table in preview mode consists of applying the common template.

```
<xsl:template match="table" mode="preview">
  <xsl:call-template name="table" select="."/>
</xsl:template>
```

Action is trivial: we copy the table and add the connectors. We assume that the table contains only rows.

```
<xsl:template name="table">
  <table>
     <xsl:copy-of select="@*"/>
     <xsl:apply-templates/>
  </table>
  <xsl:call-template name="extractconnectors"/>
</xsl:template>
```

Action is trivial: we copy the row. We assume that the row contains only cells.

```
<xsl:template match="row">
  <row>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates/>
  </row>
</xsl:template>
```

Action is trivial: we copy the content in copy mode.

```
<xsl:template match="cell">
  <cell>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates mode="copy"/>
  </cell>
</xsl:template>
```

This is a simple recursive copy of everything.

```
<xsl:template match="*|@*|text()" mode="copy">
 <xsl:copy>
   <xsl:apply-templates mode="copy" select="*|@*|text()"/>
 </xsl:copy>
</xsl:template>
```

In fact, we copy everything but the connectors.

```
<xsl:template mode="copy"
 match="nodeconnect|anodeconnect|barnodeconnect| abarnodeconnect
    |nodecurve|anodecurve|nodetriangle"/>
```

Connectors are copied by this piece of code.

```
<xsl:template name="extractconnectors">
  <xsl:for-each select="row/cell/nodeconnect| row/cell/anodeconnext
     |row/cell/barnodeconnect| row/cell/abarnodeconnect
     |row/cell/nodecurve| row/cell/anodecurve|row/cell/nodetriangle">
    <xsl:copy-of select="."/>
  </xsl:for-each>
</xsl:template>
```

8.4 The title page

In the case of the thesis, we have completely changed the text of the first page. There is a first part that contains some information (place where the thesis has been defended, on two lines, the thesis type, the name of the student, the title of the thesis, the name of the supervisor, the defense date, and, for the English version, the original French title). We show here the French version:

```
\begin{metadata}
\begin{center}
\lieuthese{Université Paris III - La Sorbonne Nouvelle}
\lieuthesesuite{U.F.R. d'Anglais}
\typethese{Linguistique}
\doctorant{Céline}{Roméro}
\thesetitre{L'évolution syntaxique des verbes modaux dans l'histoire
de l'anglais}
\directeur{Jacqueline}{Guéron}
\datesoumission{18 novembre 2005}
\end{center}
```

There is a second part that contains the Jury.

```
\begin{jury}
\membre[Pr\'esident du Jury]{M.}{Claude}{Delmas}
\membre[Directrice de Recherche]{Mme}{Jacqueline}{Guéron}
\membre{Mme}{Annie}{Lancri}
\membre{Mme}{Jacqueline}{Lecarme}
\membre{Mme}{Susan}{Pintzuk}
\end{jury}
\end{metadata}
```

In order for these commands to work, we have defined a file titlepage.plt, starting like this:

```
\ProvidesPackage{titlepage}[2007/01/10 v1.1 Thesis TitlePage]
% Copyright Inria/Apics (Jos\'e Grimm) 2006-2007
```

The translation of `\foo{bar}` is `<foo>bar<foo/>`.

```
\newenvironment{metadata}{\begin{xmlelement*}{metadata}}{\end{xmlelement*}}
\newcommand\thesetitre[1]{\xbox{thesetitre}{#1}}
\newcommand\titrefrancais[1]{\xbox{titrefrancais}{#1}}
\newcommand\lieuthese[1]{\xbox{lieuthese}{#1}}
\newcommand\typethese[1]{\xbox{typethese}{#1}}
\newcommand\lieuthesesuite[1]{\xbox{lieuthesesuite}{#1}}
\newcommand\datesoumission[1]{\xbox{datesoumission}{#1}}
\newcommand\directeur[2]{\xbox{directeur}{#1 #2}}
\newcommand\doctorant[2]{\xbox{doctorant}{#1 #2}}
```

The command `\membre` takes three mandatory arguments plus an optional one. If arguments are A, B, C and D, the result is a `<membre>` element, with attributes type, prenom, nom, with values B, C, and D, and the content of the element is the optional argument A.

```
\newcommand\membre[4][]{%
   \xbox{membre}{#1%
         \XMLaddatt{type}{#2}\XMLaddatt{prenom}{#3}\XMLaddatt{nom}{#4}}}
\newenvironment{jury}{\begin{xmlelement*}{jury}}{\end{xmlelement*}}
```

This command is used elsewhere. Normally, a dedication is a small text on a page. It is preceded by a `\chapter*` command.

```
\newcommand{\dedicace}[1]
{\xbox{dedicace}{#1}}
```

This is now the XML translation. As you can see, the `\chapter*` command produces a `<div0>` element, with an empty title; it will appear in the table of contents as an invisible line, so that it is better to comment it out.

```
<metadata>
<lieuthese>Université Paris III - La Sorbonne Nouvelle</lieuthese>
<lieuthesesuite>U.F.R. d'Anglais</lieuthesesuite>
<typethese>Linguistique</typethese><doctorant>Céline Roméro</doctorant>
<thesetitre>L'évolution syntaxique des verbes modaux dans l'histoire
de l'anglais</thesetitre>
<directeur>Jacqueline Guéron</directeur>
<datesoumission>18 novembre 2005</datesoumission>
<jury><membre nom='Delmas' prenom='Claude' type='M.'>Président du Jury</membre>
<membre nom='Guéron' prenom='Jacqueline' type='Mme'>
          Directrice de Recherche</membre>
<membre nom='Lancri' prenom='Annie' type='Mme'/>
<membre nom='Lecarme' prenom='Jacqueline' type='Mme'/>
<membre nom='Pintzuk' prenom='Susan' type='Mme'/></jury>
</metadata>
<frontmatter>
<div0 id='uid1' rend='nonumber'><head/>
<dedicace>
Text of the dedication
</dedicace></div0>
```

The research report describing Tralics starts like this:

```
\RRtitle{Tralics, un traducteur de \LaTeX\ vers XML\\Partie II}
\RRetitle{Tralics, a \LaTeX\ to XML translator\\Part II}
\RRauthor{José Grimm\thanks{Email: Jose.Grimm@sophia.inria.fr}}
\RRprojet{Apics}
\RRtheme{\THNum}
\RRNo{310}
\RRresume{bla bla}
\RRabstract{bla bla}
\RRdate{September 2005}
\URSophia
\motcle{xyz}
\keyword{xyz}
\RRversion{2}
\RRdater{January 2007}
```

All the commands used above are defined in the file RR.sty, and Tralics uses RR.plt instead (the content of the file has been given in the first part of this document). The effect of these commands is to memorize the arguments (LaTeX case) or add some elements to the XML tree (Tralics case). The start of the document contains also the following two lines; the first line is ignored in Tralics mode, and the second one in LaTeX mode, otherwise the effect is to produce the title page, or add an attribute to the document element respectively. Part one and two of the Tralics documentation have different HTML prefixes.

```
\makeRT
\htmlprefix{tdoc2}
```

The XML translation is the following.

```
<ftitle>Tralics, un traducteur de <LaTeX/> vers XML Partie II</ftitle>
<title>Tralics, a <LaTeX/> to XML translator Part II</title>
<author>José Grimm
  <note id='uid1' place='foot'>Email: Jose.Grimm@sophia.inria.fr</note>
</author>
<inria-team>Apics</inria-team>
<theme>THnum</theme>
<rrnumber>310</rrnumber>
<resume><p>bla bla</p></resume>
<abstract><p>bla bla</p></abstract>
<date>September 2005</date>
<location>Sophia Antipolis</location>
<motcle>xyz</motcle>
<keyword>xyz</keyword>
<version-number>2</version-number>
<rev-date>January 2007</rev-date>
```

This is the main template for conversion to HTML. If the document contain some text, then a table of contents then a bibliography, then an index, then footnotes, it is difficult to find the TOC, and for this reason a short TOC is included. However, if the TOC is before the main text, this is redundant; if the variable `$shorttoc` is set to false, there is no short TOC. In the case where `$split` is not false, then the main document is split, and there is no need for a short TOC.

This template allows two modes for the meta data: in the case of the thesis, the meta data are children of `<metadata>`, and the template `header` is empty; in the case of the technical report, the translation of elements like `<rev-date>` is empty, and `header` constructs the title page.

```
1  <xsl:template match="std">
2    <html>
3      <xsl:call-template name="html-meta" />
4      <body>
5        <xsl:call-template name="header"/>
6        <xsl:if test="$shorttoc='true' or
7             ($shorttoc='maybe' and $split='false')">
8          <xsl:call-template name="shorttoc"/>
9        </xsl:if>
10       <xsl:apply-templates/>
11       <xsl:call-template name="footnotes" />
12     </body>
13   </html>
14 </xsl:template>
```

This is how we construct the HTML meta data. Three auxiliary templates are used, since we do not know how to access to the author, title and keywords.

```
15 <xsl:template name="html-meta">
16   <head>
17     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
18     <meta name="author">
19       <xsl:attribute name="content">
20         <xsl:call-template name="author" />
21       </xsl:attribute>
22     </meta>
23     <title> <xsl:call-template name="title" /></title>
24     <link rel="stylesheet" href="tralics.css" />
25     <meta name="keywords">
```

```
      <xsl:attribute name="content">
        <xsl:call-template name="keywords" />
      </xsl:attribute>
    </meta>
  </head>
</xsl:template>
```

These three templates are used in the code above; since there are no keywords in the TEX source of the thesis, we invented some; for this reason the last rule exists in both French and English versions. In the case of the technical report, we select elements `<author>`, `<title>`, `<keyword>`. Note that these elements are converted in 'text' mode; the main idea is to omit the footnote in the author's field.

```
<xsl:template name="author">
  <xsl:apply-templates select="/std/metadata/doctorant" mode="text"/>
</xsl:template>
<xsl:template name="title">
  <xsl:apply-templates select="/std//metadata/thesetitre" mode="text"/>
</xsl:template>
<xsl:template name="keywords">
   <xsl:text>verbe anglais histoire</xsl:text>
</xsl:template>
```

In the case of the thesis, the `metadata` template outputs the title in a `<h1>` heading, then all meta data in a two column table in a `<blockquote>` with non-zero margins. Each data produces a row in the table (for instance, `<lieuthese>` gives two cells, one with 'Lieu', and one with the value of the element. The `<jury>` command produces two cells, the second one being a table; each `<membre>` produces a row of this table. The code is obvious, the result is given here.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
 <meta name="author" content="Céline Roméro" />
 <title>L'évolution syntaxique des verbes modaux dans l'histoire
  de l'anglais</title>
 <link rel="stylesheet" href="tralics.css" />
<meta name="keywords" content="verbe anglais histoire" />
</head>
<body>
<h1>L'évolution syntaxique des verbes modaux dans l'histoire
  de l'anglais</h1>
<blockquote  class="thesis-heading">
 <table>
  <tr><td>Lieu</td><td>Université Paris III - La Sorbonne Nouvelle</td></tr>
  <tr><td></td><td>U.F.R. d'Anglais</td></tr>
  <tr><td>Doctorat en</td><td>Linguistique</td></tr>
  <tr><td>Doctorant</td><td>Céline Roméro</td></tr>
  <tr><td>Thèse dirigée par</td><td>Jacqueline Guéron</td></tr>
  <tr><td>Date de soutenance</td><td>18 novembre 2005</td></tr>
  <tr><td>Jury</td><td>
    <table>
      <tr><td>M.</td><td>Claude</td><td>Delmas</td>
            <td>Président du Jury</td></tr>
     <tr><td>Mme</td><td>Jacqueline</td><td>Guéron</td>
            <td>Directrice de Recherche</td></tr>
```

```
     <tr><td>Mme</td><td>Annie</td><td>Lancri</td><td></td></tr>
     <tr><td>Mme</td><td>Jacqueline</td><td>Lecarme</td><td></td></tr>
     <tr><td>Mme</td><td>Susan</td><td>Pintzuk</td><td></td></tr>
  </table></td></tr>
 </table></blockquote>
<hr />
```

The style sheet TF that converts the thesis contains a variable for each name used above; the style sheet TE contains the same variable, with English names.

```
<xsl:variable name="Bibliographyname">Bibliographie</xsl:variable>
<xsl:variable name="Lieuname">Lieu</xsl:variable>
<xsl:variable name="Frenchtitlename">Titre Français</xsl:variable>
<xsl:variable name="Typethesename">Doctorat en</xsl:variable>
<xsl:variable name="Directorname">Thèse dirigée par</xsl:variable>
<xsl:variable name="Doctorantname">Doctorant</xsl:variable>
<xsl:variable name="Datesoumissionname">Date de soutenance</xsl:variable>
<xsl:variable name="Juryname">Jury</xsl:variable>
```

In the case of the report, the template `header` selects 14 items, whose translation is otherwise empty. The code is obvious, if you compare the XML shown above and the HTML translation given here.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
 <meta name="author" content="José Grimm" />
 <title>Tralics, a LaTeX to XML translator Part II</title>
 <link rel="stylesheet" href="tralics.css" />
 <meta name="keywords" content="xyz" />
</head>
<body>
<h1>Tralics, a LaTeX to XML translator Part II</h1>
<p>French title: Tralics, un traducteur de LaTeX vers XML Partie II</p>
<p> Author: José Grimm<a id="uid1" href="#note1"
   title="Email: Jose.Grimm@sophia.inria.fr"><small>(note:
   </small>&#10163;<small>)</small></a></p>
<p> Location: Sophia Antipolis</p>
<p>Inria Research Theme: THnum</p>
<p>Inria Research Report Number: 310</p>
<p>Revision: 2</p>
<p>Team: Apics</p>
<p>Date: September 2005</p>
<p>Revised Date: January 2007</p>
<p>Keywords: xyz.</p>
<p>rench keywords: xyz.</p>
<h2>Abstract</h2><p>bla bla</p>
<h2>French Abstract</h2><p>bla bla</p>
```

8.5 The style sheets

The style sheets we use for the two examples are modifications of those explained in previous chapters; we shall explain only the differences.

8.5.1 Splitting the text

The default translation consists in generating a single page from the text. However, since the thesis and our reports are very long, we allow the possibility to generate a HTML page for each main division. Since the report uses the report class, a main division is associated to a `\chapter`. In the case of the thesis, the book class is used, and by default, the main division is `\part`; since there are no chapters, we have to use `\toplevelsection {\chapter}` in order to say that the chapter is a main division. The technical report describing Tralics has a huge index (30 LaTeX pages in two-column mode); for this reason, we allow the possibility of a separate index. Finally, you can also ask for a separate page for the bibliography and the footnotes.

Assume that A is the main file, B, C, D, etc., the files associated to the first, second, third, chapter, etc. We shall make the following assumption: the whole document consists in an front section A1, a main part A2, and a back section A3, where the main part is essentially a sequence of chapters. More precisely, we assume that the translation of A2 is a sequence of links to pages B, C, D, and some HTML elements (like rules), but no text. Said otherwise, if the reader reads A1, B, C, D, A3, then he reads everything. This implies that there should be no text between chapters (because there is no end-of-chapter command in LaTeX, this is true in general). In order to complete our notations: if a separate page is used, we shall denote it by I (for the index), F (for the footnotes) and H (for the bibliography).

We also assume that the table of contents and the bibliography appear in A (not hidden in a chapter), or use a separate page. Putting the TOC between chapter 1 and chapter 2 is strange but valid. Normally, it is at the start, after the introduction or at the end of the document. The easy case is when the document contains a `<mainmatter>` (with the chapters B, C, D, etc), and is followed by a `<backmatter>` (containing A3). We can assume that everything before the mainmatter is in a `<frontmatter>`, but this is not needed. In this case the TOC is either in the front matter or in the back matter. In the case where the document looks like a report, some care has to be taken, because, by default, the bibliography is appended to the last chapter. Our thesis ends like this:

```
\backmatter
\include{Conclusion}
\printindex
\nocite{*} \bibliographystyle{these} \bibliography{these}
```

Our technical report does not use `\printindex`, but `\input`s the index; however the commands produced by `makeindex` are not understood by Tralics, so that the file is conditionally included. The important point here is that `\endsec` terminates the current chapter. The is the end of the report:

```
\cleardoublepage
\iftralics
  \endsec{\chapter}
\else
  \input tralics-rr2.ind
\fi
\bibliography{tralics}
\tableofcontents
```

The pages mentioned above are linked in the following way

- There is a link from A to B, C, and D (in fact, part A2 of A contains only these links).
- There is a link from B, C, and D to A (in fact, we point to the table of contents).
- There is a link from B to C to D to A3 (the last link is to the start of A3, the start of the `<backmatter>` if possible).

- There is a link from D to C to B to A1 (the last link is to the end of the front matter, this is the start of the `<mainmatter>` if possible).
- There is a link from A to I, H, and F (the table of contents points to the index, bibliography and footnotes in the case where these are on separate pages).
- There are links from I, H and F to A (of the form "back to main page").
- There may be links from any of these pages to any other page (internal links).

We assume that the value `$Main` contains the name A (without extension); a link to this file will be of the form `A#foo`, but we use the simple form `#foo` in the case where the file is not split (this allows you to rename the file at your leisure).

```
<xsl:template name="mainfile">
  <xsl:if test="not($split='false') ">
    <xsl:value-of select="concat($Main,'.html')" />
  </xsl:if>
</xsl:template>
```

The names B, C, D, etc are computed as follows: we use a prefix, followed by the id of the chapter. This prefix is defined in the style sheet for the thesis, or in the TeX source, via the `\htmlprefix` command.

```
<xsl:template match="div0" mode="fileprefix">
  <xsl:value-of select="concat($prefix,@id,'.html')" />
</xsl:template>
```

This is how we compute the names I, H and F. There are three similar templates, they depend on the variables `$separate-index`, for the index, `$separate-footnote`, for footnotes, and `$separate-biblio` for the bibliography. The names are obtained by concatenation of the prefix and 'uidI', 'uidF' and 'uidH'.

```
<xsl:template name="indexfile">
  <xsl:choose>
    <xsl:when test="not ($split='false') and $separate-index='true' ">
      <xsl:value-of select="concat($prefix,'uidI','.html')" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="mainfile" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="footnotefile"> ...
<xsl:template name="bibliofile"> ...
```

What precedes the first chapter is the main file; if we have a main matter, then the first chapter is the start of the main matter, so that we link to the rule that separates front and main matter.

```
<xsl:template name="first.page" >
  <xsl:call-template name="mainfile" />
  <xsl:choose>
    <xsl:when test="//mainmatter">
      <xsl:text>#mainmatter</xsl:text>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

What follows the last chapter is the main file; if we have a back matter, we use its anchor; if there is a non-separate bibliography we use it otherwise.

```
<xsl:template name="last.page" >
  <xsl:call-template name="mainfile" />
  <xsl:choose>
    <xsl:when test="//backmatter">
      <xsl:text>#backmatter</xsl:text>
    </xsl:when>
    <xsl:when test="//biblio and not($separate-biblio='false')">
      <xsl:text>#bibliography</xsl:text>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

The variable `$split` can be 'true', 'false' or 'all'. If this quantity is false, a single page is generated, if true, a HTML page is generated only for chapter in the main matter (child of `<mainmatter>`), otherwise for each chapter. This is controlled by the following template rule:

```
<xsl:template match="div0">
  <xsl:choose>
    <xsl:when test="$split='false'">
      <xsl:apply-templates select='.' mode="normal"/>
    </xsl:when>
    <xsl:when test="$split='all'">
      <xsl:apply-templates select='.' mode="split"/>
    </xsl:when>
    <xsl:when test="ancestor::mainmatter">
      <xsl:apply-templates select='.' mode="split"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates select='.' mode="normal"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This template returns the filename in which a current element is located. This assumes that `<div0>` is the only element that generates a page, is defined as shown above, and the name of the page matches the previous template. Note: if we replace 'ancestor-or-self' by 'ancestor', then the link to a `<div0>` will be an anchor in the main file, this means that you have to click twice in order to see the text.

```
<xsl:template match="*" mode="targetfile">
  <xsl:choose>
    <xsl:when test="$split='false'" />
    <xsl:when test="not(ancestor-or-self::div0)">
      <xsl:call-template name="mainfile" />
    </xsl:when>
    <xsl:when test="(ancestor::mainmatter) or $split='all'">
      <xsl:apply-templates select="ancestor-or-self::div0" mode="fileprefix" />
    </xsl:when>
   <xsl:otherwise>
      <xsl:call-template name="mainfile" />
    </xsl:otherwise>
```

```
  </xsl:choose>
</xsl:template>
```

This returns the name of the previous file. The algorithm is simpler than the case of the Raweb. In particular, since this does not apply to the main page, there is a previous and a following link.

```
<xsl:template match="div0" mode="file-prev-prefix">
  <xsl:choose>
    <xsl:when test="preceding-sibling::div0">
      <xsl:apply-templates select="preceding-sibling::div0[1]" mode="fileprefix"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="first.page" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

In the case of the last chapter, we put the link to the backmatter or the bibliography if available.

```
<xsl:template match="div0" mode="file-next-prefix">
  <xsl:choose>
    <xsl:when test="following-sibling::div0">
      <xsl:apply-templates select="following-sibling::div0[1]" mode="fileprefix"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="last.page" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This creates a link with a button to the previous page. The access key is P.

```
<xsl:template match="div0" mode="button-prev-prefix">
  <a accesskey='P' title="previous page">
    <xsl:attribute name='href'>
      <xsl:apply-templates select="." mode ="file-prev-prefix" />
    </xsl:attribute>
    <img style="border:0" alt="previous" src="previous_motif.gif" />
  </a>
</xsl:template>
```

Same for the preceding page. The access key is N.

```
<xsl:template match="div0" mode="button-next-prefix">
  <a accesskey='N' title="next page">
    <xsl:attribute name='href'>
      <xsl:apply-templates select="." mode ="file-next-prefix" />
    </xsl:attribute>
    <img style="border:0" alt="next" src="next_motif.gif" />
  </a>
</xsl:template>
```

This creates three buttons, that point to the previous, top, and next pages. They float to the left and right (as defined by the raweb.css file).

```
<xsl:template match="div0" mode="button">
  <div class="float-left">
    <xsl:apply-templates select="." mode ="button-prev-prefix" />
```

```
118   </div>
119   <div class="float-left">
120     <a accesskey='T' title="table of contents" >
121       <xsl:attribute name='href'>
122         <xsl:call-template name ="mainfile" />
123         <xsl:text>#tableofcontents</xsl:text>
124       </xsl:attribute>
125       <img style="border:0" alt="TOC" src="contents_motif.gif" />
126     </a>
127   </div>
128   <div class="float-right">
129     <xsl:apply-templates select="." mode ="button-next-prefix" />
130   </div>
131 </xsl:template>
```

If we have a separate page for the index, bibliography or footnotes, the previous and next buttons do not depend on the page. This creates three buttons, to the first page, the last page and the TOC.

```
132 <xsl:template name="std.buttons">
133   <div class="float-left">
134     <a accesskey='P' title="previous page">
135       <xsl:attribute name='href'>
136         <xsl:call-template name="first.page" />
137       </xsl:attribute>
138       <img  style="border:0"  alt="previous" src="previous_motif.gif" />
139     </a>
140   </div>
141   <div class="float-left">
142     <a accesskey='T' title="table of contents" >
143       <xsl:attribute name='href'>
144         <xsl:call-template name ="mainfile" />
145         <xsl:text>#tableofcontents</xsl:text>
146       </xsl:attribute>
147       <img style="border:0"  alt="TOC" src="contents_motif.gif" />
148     </a>
149   </div>
150   <div class="float-right">
151     <a accesskey='N' title="next page">
152       <xsl:attribute name='href'>
153         <xsl:call-template name="last.page" />
154       </xsl:attribute>
155       <img  style="border:0" alt="next" src="next_motif.gif" />
156     </a>
157   </div>
158 </xsl:template>
```

The following template creates the page for a chapter. Note that `xsl:document` is not a XSLT1.0 command, but an extension accepted by many XSLT processors.

```
159 <xsl:template match="div0" mode ="xsplit">
160   <xsl:variable name="filename">
161     <xsl:apply-templates select="." mode="fileprefix" />
162   </xsl:variable>
163   <xsl:document href="{$filename}"
```

```
      encoding='iso-8859-1' doctype-public='-//W3C//DTD XHTML 1.0 Strict//EN'
     doctype-system='http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>
      method="xml" >
```

The HTML page created here contains the same meta data as the main page. It has the navigation buttons shown above on the top and bottom. On the top, between the navigation button, there is the title of the document, on the back, there is a link to the main file.

```
    <html>
      <xsl:call-template name="html-meta" />
      <body>
        <xsl:apply-templates select="." mode="button" />
        <div class="float-center"><xsl:call-template name="title" /></div>
        <h1>
          <xsl:call-template name="id"/>
          <xsl:call-template name="calculateNumberSpace"/>
          <xsl:apply-templates select="head" mode="caption"/>
        </h1>
        <xsl:apply-templates/>
        <xsl:apply-templates select="." mode="button" />
        <xsl:call-template name="back.to.main.buttons" />
      </body>
    </html>
  </xsl:document>
</xsl:template>
```

This is for the index. We use the same attributes for the xsl:document element, but different navigation buttons. The same rule is used for the bibliography, but `bibliofile` is used instead of `indexfile` for the name of the HTML file.

```
<xsl:template match="theindex" mode ="xsplit">
  <xsl:variable name="filename">
    <xsl:call-template name="indexfile" />
  </xsl:variable>
  <xsl:document ... >
    <html>
      <xsl:call-template name="html-meta" />
      <body>
        <xsl:call-template name="title.buttons" />
        <xsl:apply-templates select="." mode="ysplit" />
        <xsl:call-template name="back.to.main.buttons" />
      </body>
    </html>
  </xsl:document>
</xsl:template>

<xsl:template match="biblio" mode ="xsplit"> ...
```

This is for the footnotes. The idea is the same, but we have a named template, so that we cannot use modes.

```
<xsl:template name="footnotes-xsplit">
  <xsl:variable name="filename">
    <xsl:call-template name="footnotefile" />
  </xsl:variable>
  <xsl:document ... >
```

```
206       <html>
207         <xsl:call-template name="html-meta" />
208         <body>
209           <xsl:call-template name="title.buttons" />
210           <xsl:call-template name="footnotes-ysplit" />
211           <xsl:call-template name="back.to.main.buttons" />
212         </body>
213       </html>
214     </xsl:document>
215   </xsl:template>
```

This prints 'Back to main page' on the bottom of a page.

```
216   <xsl:template name="back.to.main">
217     <xsl:variable name="mainpage">
218       <xsl:call-template name ="mainfile" />
219     </xsl:variable>
220     <div class="float-center">
221       <a href="{$mainpage}">Back to main page</a>
222     </div>
223   </xsl:template>
```

This creates the buttons at the end of a separated page.

```
224   <xsl:template name="back.to.main.buttons">
225     <hr/>
226     <xsl:call-template name="std.buttons" />
227     <xsl:call-template name="back.to.main" />
228   </xsl:template>
```

This creates the buttons at the start of a separated page. Same buttons, but we use the title of the document.

```
229   <xsl:template name="title.buttons">
230     <xsl:call-template name="std.buttons" />
231     <div class="float-center"><xsl:call-template name="title" /></div>
232   </xsl:template>
```

This piece of code translates id = 'foo' into href = 'bar#foo', the non trivial point being how to compute the file name 'bar'. This works only if the DTD declares the id attribute of the current element to be of type 'ID'.

```
233   <xsl:template name="id-to-href">
234     <xsl:attribute name="href">
235       <xsl:apply-templates mode="targetfile" select="id(@id)" />
236       <xsl:text>#</xsl:text><xsl:value-of select="./@id"/>
237     </xsl:attribute>
238   </xsl:template>
```

This piece of code translates target = 'foo' into href = 'bar#foo', using the same method as above.

```
239   <xsl:template name="target-to-href">
240     <xsl:attribute name="href">
241       <xsl:apply-templates mode="targetfile" select="id(@target)" />
242       <xsl:text>#</xsl:text><xsl:value-of select="@target"/>
243     </xsl:attribute>
244   </xsl:template>
```

8.5.2 Footnotes

Footnotes in the Raweb are inlined, on the contrary, here, we gather them at the end of the main page, or on a separate page, using the following rule. The non-obvious point is that, if we denote by A the code created here, and by B the translation of a footnote, we have a link from A to B, and a backlink from B to A (this could be omitted, just use then 'back' button of your browser). Assume that this is note 25, with uid33 as ID; in this case element B will have uid33 as unique id, and element A will have note25 as identifier (this means that a link to a footnote refers to its text). In fact, A contains a rule (separating it from the previous note) and a `<p>` that serves as anchor, containing and a header, like 'Note 25.', and the text of the note; the backlink is activated by clicking on the header.

There is a non trivial question here: Are paragraphs allowed in a footnote? They are (by default) removed by Tralics; but we have to take both cases into account.

```
<xsl:template match='note' mode="footnote">
    <hr/>
    <p class='nofirst noindent'>
      <xsl:attribute name="id">
        <xsl:text>note</xsl:text>
        <xsl:call-template name="calculateFootnoteNumber"/>
      </xsl:attribute>
      <a title="back to text">
        <xsl:call-template name="id-to-href"/>
        <xsl:text>Note </xsl:text>
        <xsl:call-template name="calculateFootnoteNumber"/>
        <xsl:text>. </xsl:text>
      </a>
      <xsl:if test="not(p)">
        <xsl:apply-templates />
      </xsl:if>
    </p>
    <xsl:if test="p">
       <xsl:apply-templates />
    </xsl:if>
</xsl:template>
```

Element B is an anchor, whose text is '(note: A)', where A is the character U+27B3, a kind of arrow. There is also a title, formed of the start of the text.

```
<xsl:template match='note'>
  <a id="{@id}">
    <xsl:call-template name="note-to-href"/>
    <xsl:apply-templates mode="xreftitle" select="."/>
    <small>(note: </small>
    <xsl:text>&#x27B3;</xsl:text>
    <small>)</small>
  </a>
</xsl:template>
```

This removes footnotes in the meta data.

```
<xsl:template match='note' mode="text" />
```

This returns a unique ID for a footnote, namely its number.

```
<xsl:template match='note' mode="xref">
   <xsl:call-template name="calculateFootnoteNumber"/>
</xsl:template>
```

In the case of a footnote, say note 25, we create a link from the current page to the page that contains the notes, with attribute href = 'foo#note25'; the non trivial point being how to compute the file name 'foo'.

```
<xsl:template name="note-to-href">
  <xsl:attribute name="href">
    <xsl:call-template name="footnotefile" />
    <xsl:text>#note</xsl:text>
    <xsl:apply-templates mode="xref" select =".">/>
  </xsl:attribute>
</xsl:template>
```

This completes the translation of a footnote. The non-trivial point here is the title: we convert the note in a string, and if the string is small enough, it will be the title. Otherwise, we consider only the first one hundred characters.

```
<xsl:template mode="xreftitle" match="note">
  <xsl:variable name="text" select="string(normalize-space(.))" />
  <xsl:attribute name="title">
    <xsl:choose>
      <xsl:when test="string-length($text) &gt; 110">
        <xsl:value-of select="substring($text,1,100)"/>
        <xsl:text>...</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$text"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
</xsl:template>
```

Dispatcher template. The action depends on whether a separate page is needed.

```
<xsl:template name="footnotes">
 <xsl:choose>
   <xsl:when test="not($split='false') and $separate-footnote='true' ">
     <xsl:call-template name="footnotes-xsplit" />
   </xsl:when>
   <xsl:otherwise>
     <xsl:call-template name="footnotes-ysplit" />
   </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

This is obvious. we output a title, followed by all the notes.

```
<xsl:template name="footnotes-ysplit">
  <h1>Notes</h1>
  <xsl:for-each select="//note">
    <xsl:apply-templates select="." mode="footnote"/>
  </xsl:for-each>
</xsl:template>
```

8.5.3 The index

The index is created by the following trivial rule.

```
<xsl:template match="theindex" mode="ysplit">
  <h1 id="index">Index</h1>
  <xsl:apply-templates/>
</xsl:template>
```

The situation becomes complicated when we try to use a separate page for the index. Hence the real rule is the following.

```
<xsl:template match="theindex">
 <xsl:choose>
   <xsl:when test="not($split='false') and $separate-index='true' ">
     <xsl:apply-templates select="." mode="xsplit" />
   </xsl:when>
   <xsl:otherwise>
     <xsl:apply-templates select="." mode="ysplit" />
   </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

We use the same method for the bibliography, but a different variable is used.

```
<xsl:template match="biblio">
 <xsl:choose>
   <xsl:when test="not($split='false') and $separate-biblio='true' ">
     <xsl:apply-templates select="." mode="xsplit" />
   </xsl:when>
   <xsl:otherwise>
     <xsl:apply-templates select="." mode="ysplit" />
   </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

This is how we insert the index in the TOC.

```
<xsl:template match="theindex" mode="xtoc">
  <xsl:variable name="link">
    <xsl:call-template name="indexfile" />
    <xsl:text>#index</xsl:text>
  </xsl:variable>
  <xsl:text>&#x0A;</xsl:text> <br/>
  <a href="{$link}"> <b>Index</b></a>
</xsl:template>
```

If the source contains `\index{foo!bar}`, Tralics will add an anchor at the current location (unless there is already an anchor there); let's denote it by A. Moreover, an `<index>` element with value 'foo' (at level one) and a second `<index>` with value 'bar' (at level two) will be added at the end of the document, sorted in alphabetic order; let's denote these elements by B and C. Element A has an id, say D, and element C has a target attribute, which is a list of ids, and D is added to it.

The HTML translation of elements like B and C is a `<div>` element, with a class attribute, with value 'idx1', 'idx2', or 'idx3', depending on the level; each D in the attribute target is translated as a link to A.

```
<xsl:template match="index">
  <xsl:text>&#x0A;</xsl:text>
```

```
  <div class="idx{@level}">
   <xsl:apply-templates/>
   <xsl:for-each select="id(@target)">
     <xsl:if test="position()=1">: </xsl:if>
     <a>
       <xsl:call-template name="id-to-href"/>
       <xsl:apply-templates mode="xref" select="."/>
     </a>
     <xsl:call-template name="separateur.objet"/>
   </xsl:for-each>
  </div>
</xsl:template>
```

This what an `<anchor>` produces in the index; a character that looks like a star.

```
<xsl:template match="anchor" mode ='xref'>
  <xsl:text>&#x273B;</xsl:text>
</xsl:template>
```

The following lines come from the raweb.css style sheet and explain how to indent the lines of the index.

```
.idx1 { text-indent:0em; margin-top:2px}
.idx2 { text-indent:2em;}
.idx3 { text-indent:4em;}
```

8.5.4 Divisions

We have a template `calculateNumber` that returns the number of a section. In the case of a thesis, we restrict attention to the mainmatter (front matter and back matter contain only unnumbered chapters). The code shown here must be changed in the case of the report. There is also `calculateNumberSpace` that adds a dot and a space after the number.

```
<xsl:template name="calculateNumber">
 <xsl:if test= "ancestor::mainmatter">
   <xsl:number level="multiple" from="/mainmatter" grouping-separator="."
               count="div0|div1|div2|div3|div4|div5"/>
 </xsl:if>
</xsl:template>
```

This is how we output a section of level three. Note that the default translation of `<head>` is empty. If the head appears in the TOC, it will be translated in mode 'caption', in which case anchors, footnotes, etc., are removed. Here we use full mode.

```
<xsl:template match="div3">
  <h4>
    <xsl:call-template name="id"/>
    <xsl:call-template name="calculateNumberSpace"/>
    <xsl:apply-templates select="head" mode="full"/>
  </h4>
  <xsl:apply-templates/>
</xsl:template>
```

The translation of `<tableofcontents>` is the TOC, it is a `<h1>` element (named 'Table des Matières' in the French thesis, 'Table of contents' in the English report) containing all divisions (the short TOC has only chapters), plus bibliography, index, and TOC (for the short TOC only), evaluated in 'xtoc' mode. Translation is formed of a newline character, a `<br>` element, some space, and a anchor to the section (with the number and title of the section).

```
<xsl:template match="div3" mode="xtoc">
  <xsl:text>&#x0A;</xsl:text>
  <br/>
  <xsl:apply-templates mode="prefix" select="."/>
  <xsl:call-template name="calculateNumberSpace"/>
  <a>
    <xsl:call-template name="id-to-href"/>
    <xsl:apply-templates select="head" mode="caption"/>
  </a>
</xsl:template>
```

This is how we indent the section. A `<div2>` has less `<spaces>` and a `<div4>` has more. Each template creates 5 non-break space characters.

```
<xsl:template match="div3" mode="prefix">
  <xsl:call-template name="spaces"/>
  <xsl:call-template name="spaces"/>
  <xsl:call-template name="spaces"/>
</xsl:template>
```

All elements that can be referenced to have a rule in mode 'xreftitle', the action is generally empty; in the case of `<citation>`, we select the authors in mode 'xref-fullauthor'; in the case of a note, we select the text of the note (see above); in the case of a division, we select the title. In all cases, the text becomes the value of the attribute title of the anchor. We show here only the case of a division of level 3.

```
<xsl:template mode="xreftitle" match="div3">
  <xsl:attribute name="title">
     <xsl:apply-templates select="head" mode="caption"/>
  </xsl:attribute>
</xsl:template>
```

When a reference is made to an item, we evaluate it in mode 'xref'; the result is, for instance, a section number computed by the rule given here. The case of a figure, a footnote, a table, a citation, is similar.

```
<xsl:template mode="xref" match="div3">
  <xsl:call-template name="calculateNumber"/>
</xsl:template>
```

In the case of an `<item>`, we use the label if there is any (this is important, in the case where Tralics has calculated the number, and we do not want the style sheet to computed something different).

```
<xsl:template match="item" mode="xref">
  <xsl:choose>
    <xsl:when test="@label">
      <xsl:value-of select="@label" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="calculateItemNumber"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This is how we translate a `<p>` element; the result is a `<p>`, unless we are in a verbatim section, case where new lines are obeyed. If the paragraph should be centered, we add a style attribute, if it should not be indented, we add a class attribute.

```
<xsl:template match="p">
  <xsl:choose>
    <xsl:when test="parent::pre">
      <xsl:apply-templates/>
    </xsl:when>
    <xsl:otherwise>
      <p>
        <xsl:if test="@noindent = 'true'">
          <xsl:attribute name="class">nofirst noindent</xsl:attribute>
        </xsl:if>
        <xsl:if test="@rend = 'center'">
          <xsl:attribute name="style">text-align:center</xsl:attribute>
        </xsl:if>
        <xsl:apply-templates/>
      </p>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This is for the dedication.

```
<xsl:template match="dedicace">
  <blockquote><p><i><xsl:apply-templates/></i></p></blockquote>
</xsl:template>
```

This piece of code is used for inserting a non-floating image, without caption (in the case where we want to show in the HTML file the LaTeX output, for instance).

```
<xsl:template match="rawimage">
  <img>
    <xsl:attribute name="src">
      <xsl:apply-templates/>
    </xsl:attribute>
    <xsl:attribute name="alt">
      <xsl:apply-templates/>
    </xsl:attribute>
  </img>
</xsl:template>
```

Chapter 9

The DTDs

9.1 The Raweb DTD

In this chapter we give the Raweb DTD (old and new one). The file starts like this.

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

There is the Copyright notice of the file:

```
<!--
  Copyright 2002-2004, 2005 Jose Grimm/ INRIA/ Apics Project
  You have the right to use this file in any way, with the following
  restriction: you cannot distribute modifications of this file under
  its name raweb3.dtd, the name raweb.dtd,
  nor rawebXXXX.dtd,  where XXXX are four digits like 2004.
  The name 'raweb3.dtd' is associated to the raweb DTD of YEAR 2003.
-->
```

We define here the `&` and `<` entities. I'm not sure that these definitions are required.

```
<!ENTITY amp "&#x26;#x26;" >
<!ENTITY lt  "&#x26;#x3C;" >
```

We define here some entities. They are used to typeset things like 1^{st}, 2^{nd}, etc., in French.

```
<!ENTITY  ier "<hi rend='sup'>er</hi>">
<!ENTITY  iers "<hi rend='sup'>ers</hi>">
<!ENTITY  iere "<hi rend='sup'>re</hi>">
<!ENTITY  ieres "<hi rend='sup'>res</hi>">
<!ENTITY  ieme "<hi rend='sup'>e</hi>">
<!ENTITY  iemes "<hi rend='sup'>es</hi>">
<!ENTITY  numero "n<hi rend='sup'>o</hi>">
<!ENTITY  Numero "N<hi rend='sup'>o</hi>">
```

This includes the MathML DTD. Note that `<ident>` and `<list>` are defined here and in the MathML DTD, they are in a different namespace.

```
<!ENTITY % list.qname "m:list" >
<!ENTITY % ident.qname "m:ident" >

<!ENTITY % mathml PUBLIC "mathml" "mathml2.dtd">
%mathml;
```

9.1.1 General purpose elements

We list here the elements that can appear in the body of a division. There was a `<cit>` element here, it should appear only inside a `<p>`. Since version 2.5, Tralics inserts a `\leavevmode` before a `\cite`. Note that `<formula>`, `<table>` or `<figure>` can be inline or not; inline versions are refused here.

```
<!ENTITY % tei-aux "(p | list | note | formula | table | figure)+" >
```

We list here elements that can only appear in certain places (for instance in a title). In the case of a `<formula>`, it should be inline.

```
<!ENTITY % texte-restreint
  "ident | code |  hi | term | ref | xref | formula | TeX | LaTeX" >
```

This is the concatenation of the two lists, with `<cit>` added.

```
<!ENTITY % texte-general
   "ident | code | hi | term| ref| xref | formula | cit | label | list |
     note | figure | table | TeX | LaTeX" >
```

The TEI defines a lot of common attributes. Here we use only two of them. Note: the id is used only in the following cases: a division (including modules, and sections like 'fondements'), an item in a list, a footnote, a table, a figure, a formula, an entry in the bibliography.

```
<!ENTITY % tei-common-atts
        'id ID #IMPLIED rend CDATA #IMPLIED'>
```

Definition of `<code` *common-atts*`>`*text*`</code>`. The content of the element is assumed to be some verbatim material, hence it contains only text. Tralics does not produce directly such an element.

```
<!ELEMENT code (#PCDATA) >
<!ATTLIST code  %tei-common-atts; >
```

Definition of `<ident` *common-atts*`>`*text*`</ident>`. The content of the element is assumed to be the name of an identifier, hence it contains only text. Tralics does not produce directly such an element.

```
<!ELEMENT ident (#PCDATA) >
<!ATTLIST ident %tei-common-atts; >
```

Definition of `<cit rend=`*A*`><ref></cit>`. The content of the element is a reference to the bibliography; Tralics sets the rend attribute to 'foot' in the case of `\footcite`.

```
<!ELEMENT cit (ref) >
<!ATTLIST cit rend CDATA #IMPLIED >
```

Definition of `<list type=`*X*`>` *body*`</list>`. This was simplified a lot, because we removed all references to `<anchor>`. As a result, a list can have an optional title (a `<head>`, but Tralics does not produce it), followed by a sequence of `<item>`. If the list has type gloss or description, the content is a sequence of `<label>` plus `<item>`.

```
<!ELEMENT list
        ( head?, (item* | (label, item)+)) >
<!ATTLIST list
        %tei-common-atts;
        type (simple|gloss|ordered|description)  "simple" >
```

Definition of `<item id=`*A*`>`*text*`</item>`. The content can be characters, paragraphs, and some general text (Tralics produces `<p>` elements in general). You can make a reference to an item; the rend attribute is not used.

```
<!ELEMENT item
        (#PCDATA | %texte-general; | p )* >
```

```
<!ATTLIST item
        id ID #IMPLIED
        rend CDATA #IMPLIED >
```

Definition of `<label>text</label>`. The content is the label of an item; it is the optional argument of the `\item` command. Only simple text is accepted (in particular, if you want a bullet, you should use `\textbullet` rather than `\bullet`).

```
<!ELEMENT label
        (#PCDATA | %texte-restreint; )* >
<!ATTLIST label %tei-common-atts; >
```

This defines the content of a division; at level 2, we accept only divisions of level 3 or 4. We need two variables for each level, since we cannot give a name `foo` to `(div3|div4)` and use it like `foo+` or `foo*`.

```
<!ENTITY % div0-textp "(div1|div2|div3|div4)+">
<!ENTITY % div0-texts "(div1|div2|div3|div4)*">
<!ENTITY % div1-textp "(div2|div3|div4)+">
<!ENTITY % div1-texts "(div2|div3|div4)*">
<!ENTITY % div2-textp "(div3|div4)+">
<!ENTITY % div2-texts "(div3|div4)*">
<!ENTITY % div3-textp "(div4)+">
<!ENTITY % div3-texts "(div4)*">
```

We explain here what can be put in the header of a module: participants, keywords, or moreinfo.

```
<!ENTITY % particip "participant|participants|participante|participantes" >
<!ENTITY % ramodule-header "(moreinfo|keywords|%particip;)*">
```

This describes the attributes for a division. For the Raweb, only id is used, rend and type are ignored.

```
<!ENTITY % tei-div-atts '
        %tei-common-atts;
        type CDATA #IMPLIED '>
```

This describes a `<module>`. There are some attributes: the html attribute is marked 'required', but it is nowadays unused, and id is preferred. The topic attribute, if present, must be a reference to a `<topic>` element (identified by its num attribute). Other attributes are unused. The content is: a `<head>` that indicates the title of the module, followed by some meta data (participants, keywords, more info), and some text (paragraphs, formulas, etc.) or divisions (`<div2>`, `<div3>`, etc.).

```
<!ELEMENT module
    (head, %ramodule-header;, (%div1-textp; | (%tei-aux;, %div1-texts;))) >

<!ATTLIST module %tei-div-atts;
       html CDATA #REQUIRED
       topic CDATA #IMPLIED >
```

A `<div2>` is like a `<module>`; but there are no topic and no html attribute. It is one level below a module.

```
<!ELEMENT div2
    (head, %ramodule-header;, (%div2-textp; | (%tei-aux;, %div2-texts;))) >
<!ATTLIST div2 %tei-div-atts; >
```

A `<div3>` is like a `<div2>`. It is two levels below a module.

```
<!ELEMENT div3
   (head, %ramodule-header;, (%div3-textp; | (%tei-aux;, %div3-texts;))) >
<!ATTLIST div3 %tei-div-atts; >
```

A <div4> is like a <div2>. It is three levels below a module. The Raweb does not define <div5>!

```
<!ELEMENT div4
        (head, %ramodule-header;, %tei-aux;) >
<!ATTLIST div4 %tei-div-atts; >
```

Definition of a <table>. It is formed of a <head>, the caption of the table, optional, and a sequence of <row>. The rows and cols attributes are defined by the TEI as the number of rows in the table and the number of columns per row. They are not used by the Raweb.

```
<!ELEMENT table
       (head*, row+) >

<!ATTLIST table
        %tei-common-atts;
        rows NMTOKEN #IMPLIED
        cols NMTOKEN #IMPLIED >
```

A <row> is a sequence of <cell>. In the initial versions, a row could contain other things, like a table, but this seems stupid. The attributes top-border and bottom-border can be set to true if you want a border for each cell in the row. The space-before controls the space between this row and the preceding one. The role could be used to emphasize (for instance change the background color, for the first row).

```
<!ELEMENT row
        (cell)+ >
<!ATTLIST row
        %tei-common-atts;
        top-border (true|false) "false"
        bottom-border (true|false) "false"
        space-before CDATA #IMPLIED
        role CDATA "data" >
```

A <cell> can contain text, or more complicated things. It can be empty. Normally, the horizontal or vertical span is one; this can be changed by setting the attributes rows and cols. The attributes right-border, left-border, top-border, and bottom-border can be set to true if you want a border on either side. The halign indicates horizontal alignment (it can be left, center, or right). The role could be used to emphasize (for instance change the background color, for the first column).

```
<!ELEMENT cell
        (#PCDATA | %texte-general;)* >
<!ATTLIST cell
        %tei-common-atts;
        role CDATA "data"
        rows NMTOKEN "1"
        cols NMTOKEN "1"
        right-border (true|false) "false"
        left-border (true|false) "false"
        halign CDATA #IMPLIED
        top-border (true|false) "false"
        bottom-border (true|false) "false"
   >
```

A <figure> element has a rend attribute. If this is 'inline', the content should be empty, and the file attribute should define an image. You can also specify a non-inline figure, with no attributes and a content formed uniquely of a caption (a <head> element), or you could specify an optional caption and a sequence of paragraphs (no file in this case). In the case a file attribute is given, this should be the name of an image; in this case, you may say framed = 'true' if you want a frame around the image; you can set width and height if you want to alter the dimensions of the image, or scale, if you want to rescale it (you cannot give scale together with height or depth); you can specify angle if you want to turn the image.

```
<!ELEMENT figure
        (head?, p*)>
<!ATTLIST figure
        id ID #IMPLIED
        rend (inline|float) "float"
        file CDATA #IMPLIED
        framed CDATA #IMPLIED
        width CDATA #IMPLIED
        height CDATA #IMPLIED
        scale CDATA #IMPLIED
        angle CDATA #IMPLIED>
```

You can say <simplemath>N</simplemath> if you want the TEX equivalent of N. This must be in a formula; it could be rendered as \textit{N}. It contains a single character.

```
<!ELEMENT simplemath (#PCDATA) >
```

A <formula> is a wrapper for a math expression; this could be a simple math expression (as above), or a true math expression (as defined by MathML). It has an attribute type. Normally, simple math expressions should be inline. You can reference a formula via its id, but this works only for display math formulas.

```
<!ELEMENT formula
        (simplemath |math) >
<!ATTLIST formula
        %tei-common-atts;
        type (inline|display) "inline" >
```

A <keywords> element contains a list of <term>. The title can be used to typeset the list.

```
<!ELEMENT keywords (term+) >
<!ATTLIST keywords %tei-common-atts;
        titre CDATA #FIXED "Key words: "    >
```

A <term> is a keyword in a list. Attributes are currently unused.

```
<!ELEMENT term
        (#PCDATA | %texte-restreint;)* >
<!ATTLIST term
        %tei-common-atts;
        type CDATA #IMPLIED >
```

A <p> element is a paragraph. It contains some text (inline math, citations, etc). It is generally indented (but the first paragraph is a section, could have zero indentation), unless noindent is true. The attribute spacebefore can be used if more vertical space is wanted before the start of the paragraph.

```
<!ELEMENT p
        (#PCDATA | %texte-general; )* >

<!ATTLIST p
```

```
137         %tei-common-atts;
138         spacebefore CDATA #IMPLIED
139         noindent CDATA #IMPLIED>
```

The <hi> element can be used to hilight some text; the rend attribute explains how; on page 161 we list the different values accepted in HTML, and on page 243, we give a longer list for the Pdf case.

```
140 <!ELEMENT hi
141         (#PCDATA | %texte-general; )* >
142 <!ATTLIST hi
143         id ID #IMPLIED
144         rend CDATA #REQUIRED >
```

The <ref> element defines a reference to some element whose id is the value of the target attribute in the current document; its content is in general empty. The TEI says that the type is IDREFS, and the <ref> element can point to zero, one and more objects; the Raweb says that one and only one target must be given. The content of the element is useful only if the reference points to the bibliography (i.e., is in a <cit>). The type attribute is not used by the Raweb.

```
145 <!ELEMENT ref
146         (#PCDATA | %texte-general; )* >
147 <!ATTLIST ref
148         %tei-common-atts;
149         type CDATA #IMPLIED
150         target IDREF #IMPLIED >
```

The <xref> element defines a reference to an external document, defined by the url attribute (which should be a valid URL). Its content could be the value of the link, or maybe something else; it defines the zone in which the link is active. The type attribute is not used by the Raweb.

```
151 <!ELEMENT xref
152         (#PCDATA | %texte-general; )* >
153 <!ATTLIST xref
154         %tei-common-atts;
155         type CDATA #IMPLIED
156         url CDATA #IMPLIED>
```

The <head> element can be used as a section title or a caption. Attributes are ignored.

```
157 <!ELEMENT head
158         (#PCDATA | %texte-general; )* >
159 <!ATTLIST head
160         %tei-common-atts;
161         type CDATA #IMPLIED >
```

A <note> element can be used for footnotes, for instance if you say place = 'foot'. Other attributes like rend, anchored and target are currently ignored. Paragraphs are allowed.

```
162 <!ELEMENT note
163         (#PCDATA | %texte-general; |  p )* >
164 <!ATTLIST note
165         id ID #IMPLIED
166         rend CDATA #IMPLIED
167         type CDATA #IMPLIED
168         place CDATA "unspecified"
169         anchored (yes | no) "yes"
170         target IDREFS #IMPLIED >
```

The <anchor/> element is currently unused. We do not show the attribute list.

```
<!ELEMENT anchor EMPTY >
```

9.1.2 Elements specific to the Raweb

The main element is <raweb>; it is formed by a <accueil>, an optional <moreinfo>, and ten sections. All sections should be present, the mandatory ones are <composition> (who is in the team), <presentation> (a short presentation of the team), <resultats> (new result of the teams this year) and the bibliography. The language attribute indicates the language of the document (currently English), the year attribute indicates the year, and the creator holds the name of the software that created the document.

```
<!ELEMENT raweb (accueil, moreinfo?, composition, presentation,
  fondements?,domaine?, logiciels?, resultats,contrats?,international?,
  diffusion?,biblio) >
<!ATTLIST raweb year CDATA #IMPLIED >
<!ATTLIST raweb language CDATA #IMPLIED >
<!ATTLIST raweb creator CDATA #IMPLIED >
```

The <composition> element is one of the ten sections of the Raweb, as such it has some constant attributes like titre, html and numero. It has a possible id (making a reference to a section is not a good idea, because no HTML page is associated to a section; in the case without topics, a section is a continuous sequence of pages, so a link to the first page could be used; this is no more true in the case with topics). This section contain an optional <moreinfo> and a sequence of <catperso> elements.

```
<!ELEMENT composition (moreinfo?,catperso+)>
<!ATTLIST composition
           titre CDATA #FIXED "Team"
           html CDATA #FIXED "composition"
           numero CDATA #FIXED "1"
           id ID #IMPLIED>
```

The <presentation> section contains some <module> that explain briefly the main objectives of the Team.

```
<!ELEMENT presentation (module+) >
<!ATTLIST presentation
          titre CDATA #FIXED "Overall Objectives"
          numero CDATA #FIXED "2"
          id ID #IMPLIED>
```

The <fondements> section contains some <module> that explain the scientific foundations of the research of the Team.

```
<!ELEMENT fondements (module+) >
<!ATTLIST fondements
          titre CDATA #FIXED "Scientific Foundations"
          numero CDATA #FIXED "3"
          id ID #IMPLIED>
```

The <domaine> section contains some <module> that explain the application domains of the research of the Team.

```
<!ELEMENT domaine (module+) >
<!ATTLIST domaine
        titre CDATA #FIXED "Application Domains"
        numero CDATA #FIXED "4"
        id ID #IMPLIED>
```

The <logiciels> section contains some <module> that describe the software developed in the Team.

```
<!ELEMENT logiciels (module+) >
<!ATTLIST logiciels
        titre CDATA #FIXED "Software"
        numero CDATA #FIXED "5"
        id ID #IMPLIED>
```

The <resultats> section is the main section of the Raweb; it contains some <module> that explain the results of the Team for the current year.

```
<!ELEMENT resultats (module+) >
<!ATTLIST resultats
        titre CDATA #FIXED "New Results"
        numero CDATA #FIXED "6"
        id ID #IMPLIED>
```

The <contrats> section contains some <module> that explain the how the team is funded (it lists the contracts with industry, etc.)

```
<!ELEMENT contrats (module+) >
<!ATTLIST contrats
        titre CDATA #FIXED "Contracts and Grants with Industry"
        numero CDATA #FIXED "7"
        id ID #IMPLIED>
```

The <international> section contains some <module> that explain national, european, and international activities of the Team.

```
<!ELEMENT international (module+) >
<!ATTLIST international
        titre CDATA #FIXED "Other Grants and Activities"
        numero CDATA #FIXED "8"
        id ID #IMPLIED>
```

The <diffusion> section contains some <module> that explain the how the research of the Team is diffused (teaching, etc.)

```
<!ELEMENT diffusion (module+) >
<!ATTLIST diffusion
        titre CDATA #FIXED "Dissemination"
        numero CDATA #FIXED "9"
        id ID #IMPLIED>
```

The <accueil> element provides some information about the Team. The <addresse> element is obsolete, replaced by <UR>. The <typeprojet> is obsolete, replaced by the isproject attribute. The html attribute defines the name of the Team, using only ASCII characters, like 'Miro'; the <projet> element can be more complicated, like 'Miró', and the <projetdeveloppe> may contain "Systèmes à objets, types et prototypes : sémantique et validation" (whenever possible, the language should be the same as in the main document).

```
<!ELEMENT accueil (theme,projet,projetdeveloppe,UR,topic*) >
<!ATTLIST accueil html CDATA #REQUIRED >
<!ATTLIST accueil isproject (true|false) "true">
<!ELEMENT typeprojet (#PCDATA)>
<!ELEMENT addresse (#PCDATA) >
```

The <theme> should be one of com, cog, num, bio, sym. The official list is defined in t the Tralics configuration file.

```
<!ELEMENT theme (#PCDATA)>
<!ELEMENT projet (#PCDATA|hi)*>
<!ELEMENT projetdeveloppe (#PCDATA|hi)* >
```

The <UR> element contains some (at least one) references to Inria Research Units. Currently, there are six of them. This may change in a near future, in which case this file has be modified, if you want to validate the Tralics output; on the other hand, the default attribute defined here (for instance the URLs) are not used in the new DTD; so that modifications to the file are not required for construction the Raweb.

```
<!ELEMENT UR (URSophia|URRocquencourt|URRhoneAlpes|URRennes|URLorraine|URFuturs)+>
```

The definition of <URRocquencourt/> includes the full name (Rocquencourt) and a link to the official web page.

```
<!ELEMENT URRocquencourt EMPTY>
<!ATTLIST URRocquencourt
 url  CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-rocq.en.html"
 nom  CDATA #FIXED "Rocquencourt" >
```

Same for <URRennes>.

```
<!ELEMENT URRennes EMPTY>
<!ATTLIST URRennes
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-ren.en.html"
 nom CDATA #FIXED "Rennes" >
```

Same for <URSophia>. Is the official name "Sophia-Antipolis" with a dash or without? the official web site[1] uses no dash, on Inria's front page (http://www-sop.inria.fr/) there are sometimes dashes, but not always.

```
<!ELEMENT URSophia EMPTY>
<!ATTLIST URSophia
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-sop.en.html"
 nom CDATA #FIXED "Sophia Antipolis" >
```

Some for <URLorraine>.

```
<!ELEMENT URLorraine EMPTY>
<!ATTLIST URLorraine
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-lor.en.html"
 nom CDATA #FIXED "Lorraine" >
```

Same for <URRhoneAlpes>.

```
<!ELEMENT URRhoneAlpes EMPTY>
<!ATTLIST URRhoneAlpes
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-ra.en.html"
 nom CDATA #FIXED "Rhône-Alpes" >
```

Same for <URFuturs>.

```
<!ELEMENT URFuturs EMPTY>
<!ATTLIST URFuturs
 url CDATA #FIXED "http://www.inria.fr/inria/organigramme/fiche_ur-futurs.en.html"
 nom CDATA #FIXED "Futurs" >
```

Definitions of four similar elements. They contain a list of <pers>. You should use an 's' if more than one person is concerned. Use an 'e' if only women are concerned. This distinction is useless in English.

[1] http://www.sophia-antipolis.org/

```
<!ELEMENT participants (pers)+ >
<!ELEMENT participantes (pers)+ >
<!ELEMENT participante (pers)+ >
<!ELEMENT participant (pers)+ >

<!ATTLIST participants titre CDATA #FIXED "Participants: ">
<!ATTLIST participantes titre CDATA #FIXED "Participants: ">
<!ATTLIST participante titre CDATA #FIXED "Participant: ">
<!ATTLIST participant titre CDATA #FIXED "Participant: ">
```

The `<catperso>` element defines a category of staff. It has a title (for instance "Boss"), and is followed by the list of all persons in the category (for instance, the big boss, the small boss, etc). Usually, alphabetic order is used.

```
<!ELEMENT catperso (head,pers+)>
```

A `<pers>` element has two attribute: first name and last name. It can have a content, that explains the position of the person. We allow footnotes, links, font changes, etc. Note that the whole element (including the name) should fit on a line.

```
<!ELEMENT pers (#PCDATA|hi|note|xref|ref)* >
<!ATTLIST pers prenom CDATA #REQUIRED
               nom CDATA #REQUIRED>
```

A `<moreinfo>` contains paragraphs. In can be used globally for the Raweb, or in modules, for adding a short sentence (like "Work done in collaboration with Team X").

```
<!ELEMENT moreinfo (p+) >
```

A `<topic>` contains a `<t_titre>`, that contains text, and has a unique attribute (currently a number...). The text describes the topic. Most modules can reference a topic.

```
<!ELEMENT topic (t_titre) >
<!ELEMENT t_titre (#PCDATA) >
<!ATTLIST topic num CDATA #IMPLIED>
```

The `<biblio>` element contains the whole bibliography, all `<citation>` elements. It has three constant attributes: the title, the section number, the HTML file name.

```
<!ELEMENT biblio (citation)* >
<!ATTLIST biblio
        html CDATA #FIXED "bibliography"
        titre CDATA #FIXED "Bibliography"
        numero CDATA #FIXED "10">
```

Definition of `<TeX/>` and `<LaTeX/>`. They are empty, and should be rendered as TEX and LATEX.

```
<!ELEMENT TeX EMPTY>
<!ELEMENT LaTeX EMPTY>
```

9.1.3 The bibliography

These are all the possible children of a citation. The `<xref>` element is defined above, others are specific to the bibliography.

```
<!ENTITY % bibliostuff "bnote|bauteurs|bediteur|btitle|borganization|
     bschool|byear|bmonth|xref|bseries|bnumber|bvolume|bedition|
     binstitution|baddress|bpages|bhowpublished|bbooktitle
     |bpublisher|bjournal|bchapter|btype|bdoi">
```

The `<citation>` contains the elements indicated above. It has five attributes. The type attribute reflects the BibTeX type. Entries that have a different type are lost by the Raweb mechanism. There are three required attributes `<key>`, `<from>`, `<id>`. The key is the quantity that will appear in the text, before the reference (however, a postprocessor could re-arrange the list, sort it, and recompute the key). The id identifies the reference, so that it can be referenced. The from attribute is something required by the Raweb, it is used for sorting the citations. Finally, userid is optional: this is the cite key, that appears in the BibTeX source. Useful for debug.

```
<!ELEMENT citation (%bibliostuff;)*>
<!ATTLIST citation key CDATA #REQUIRED
                   userid CDATA #IMPLIED
                   id ID #REQUIRED
                   type (book|booklet|proceedings|phdthesis|article|inbook|
         incollection|inproceedings|conference|manual|techreport|coursenotes
        |unpublished |misc|masterthesis|mastersthesis) #REQUIRED
                   from (year|foot|refer) #REQUIRED >
```

Definition of `<etal/>`. There is a nom attribute, that helps typesetting this element. You can use this element in an author list, like 'etc.' in an enumeration.

```
<!ELEMENT etal EMPTY>
<!ATTLIST etal nom CDATA #FIXED "et al." >
```

Definition of `<bpers prenom=A nom=B part=C junior=D/>` In the case where the BibTeX entry is "de la Porte, Fils, {\'Emile}", the prenom would be '`Émile`', the nom would be '`Porte`', the part would be '`de la`' and the junior would be '`Fils`'.

```
<!ELEMENT bpers EMPTY>
<!ATTLIST bpers prenom CDATA #REQUIRED
                part CDATA #IMPLIED
                nom CDATA #REQUIRED
                junior CDATA #IMPLIED>
```

Definition of `<bauteurs>` and `<bediteur>`. This is the list of authors or editors of a citation. The content should be a sequence of `<bpers>`, optionally followed by a `<etal>`. The bname attribute is used for typesetting the element.

```
<!ELEMENT bauteurs (bpers|etal)* >
<!ATTLIST bauteurs bname CDATA #FIXED "authors" >
<!ELEMENT bediteur (bpers|etal)* >
<!ATTLIST bediteur bname CDATA #FIXED "editors" >
```

For the elements that follow, explanations are taken from the LaTeX companion [7]. The bname attribute can be used to typeset the field.

The organization that sponsors a conference or that publishes a manual.

```
<!ELEMENT borganization (#PCDATA) >
<!ATTLIST borganization bname CDATA #FIXED "organisation" >
```

Institution sponsoring a technical report. We allow font changes here.

```
<!ELEMENT binstitution (#PCDATA|hi)* >
<!ATTLIST binstitution bname CDATA #FIXED "institution" >
```

Usually the address of the publisher or other institution. For major publishing houses, just give the city. For small publishers, specifying the complete address might help the reader.

```
<!ELEMENT baddress (#PCDATA) >
<!ATTLIST baddress bname CDATA #FIXED "address" >
```

Journal name. Abbreviations are provided for many journals (but Tralics knows none of them). Font changes are allowed.

```
<!ELEMENT bjournal (#PCDATA|hi)* >
<!ATTLIST bjournal bname CDATA #FIXED "journal" >
```

The name of a series or set of books. When citing an entire book, the `<btitle>` field gives its title and an optional `<series>` field gives the name of a series or multivolume set in which the book is published.

```
<!ELEMENT bseries (#PCDATA|hi)* >
<!ATTLIST bseries bname CDATA #FIXED "series" >
```

Title of a book, part of which is being cited. For book entries, use the `<btitle>` field. This allows font changes.

```
<!ELEMENT bbooktitle (#PCDATA|hi)* >
<!ATTLIST bbooktitle bname CDATA #FIXED "booktitle" >
```

The publishers' name.

```
<!ELEMENT bpublisher (#PCDATA |hi)* >
<!ATTLIST bpublisher bname CDATA #FIXED "publisher" >
```

One or more page numbers or range of numbers (e.g. `42-111` or `7,41,,73-97` or `43+`, where the '`+`' indicates pages that do not form a simple range).

```
<!ELEMENT bpages (#PCDATA) >
<!ATTLIST bpages bname CDATA #FIXED "pages" >
```

A chapter (or section or whatever) number.

```
<!ELEMENT bchapter (#PCDATA) >
<!ATTLIST bchapter bname CDATA #FIXED "chapter" >
```

The type of a technical report (e.g. "Research Notes"). This name is used instead of the default "Technical Report". For the type 'phdthesis' you could use the term "Ph.D. dissertation" by specifying `type="{Ph.D.}dissertation"`. Similarly, for the `<inbook>` and `<incollection>` entry types you can get "section 1.2" instead of the default "chapter 1.2" with `chapter = "1.2"`, `type="Section"`. Note: the `<btype>` field is set by Tralics; the semantics is different. Currently, the bname value of `<bchapter>` is used instead of the quantity given here. This might change in a future version.

```
<!ELEMENT btype (#PCDATA|hi)* >
<!ATTLIST btype bname CDATA #FIXED "type" >
```

How something strange has been published. It can contain an external link.

```
<!ELEMENT bhowpublished (#PCDATA|xref|hi)* >
<!ATTLIST bhowpublished bname CDATA #FIXED "howpublished" >
```

The edition of a book (e.g. "Second"). This should be an ordinal, and should have the first letter capitalized, as shown above; the standard styles convert to lowercase when necessary. Note. The Raweb leaves this currently unchanged. Thus, be careful.

```
<!ELEMENT bedition (#PCDATA) >
<!ATTLIST bedition bname CDATA #FIXED "edition" >
```

The number of a journal, magazine, technical report, or work in a series. An issue of a journal or magazine is usually identified by its volume and number; a technical report normally has a number; and sometimes books in a named series carry numbers.

```
<!ELEMENT bnumber (#PCDATA) >
<!ATTLIST bnumber bname CDATA #FIXED "number" >
```

The volume of a journal or multivolume book.

```
<!ELEMENT bvolume (#PCDATA|hi)* >
<!ATTLIST bvolume bname CDATA #FIXED "volume" >
```

The month in which this work was published or, for an unpublished work, in which it was written. For reasons of consistency, the standard three-letter abbreviations (`jan`, `feb`, `mar`, etc.) should be used. Note. Currently, Tralics replaces abbreviations by values; maybe we could add an attribute that says: this is an abbreviation.

```
<!ELEMENT bmonth (#PCDATA) >
<!ATTLIST bmonth bname CDATA #FIXED "month" >
```

The year of publication or, for an unpublished work, the year it was written. Generally, it should consist of four numerals, such as `1984`, although the standard styles can handle any year whose last four nonpunctuation characters are numerals, such as `about 1984`.

```
<!ELEMENT byear (#PCDATA|hi)* >
<!ATTLIST byear bname CDATA #FIXED "year" >
```

This is the doi (Digital Object Identifier). Use it whenever possible.

```
<!ELEMENT bdoi (#PCDATA)* >
<!ATTLIST bdoi bname CDATA #FIXED "DOI" >
```

Any additional information that can help the reader. This can contain math formulas. It can contain an external link. It can contain a link to the bibliography.

```
<!ELEMENT bnote (#PCDATA|xref|hi|cit|formula)* >
<!ATTLIST bnote bname CDATA #FIXED "note" >
```

The name of the school where the thesis was written.

```
<!ELEMENT bschool (#PCDATA|hi)* >
<!ATTLIST bschool bname CDATA #FIXED "school" >
```

The work's title, typed as explained in Section 13.2.2. Note: the companion explains how BibTeX converts some uppercase letters into lower case ones. This does not apply here.

```
<!ELEMENT btitle (#PCDATA|hi|TeX|LaTeX|formula)* >
<!ATTLIST btitle bname CDATA #FIXED "title" >
```

9.1.4 Research Reports

This DTD could also be used for other documents, like a research report. We declare here divisions of level zero and one.

```
<!ELEMENT div0
    (head, (%div0-textp; | (%tei-aux;, %div0-texts;))) >
<!ELEMENT div1
    (head, (%div1-textp; | (%tei-aux;, %div1-texts;))) >
```

The title page of a Research Report will contain the following items.

```
<!ELEMENT RRstart (UR,title, etitle, projet, theme, motcle, keyword,
   resume, abstract, author,date, RRnumber)*>
```

The `<UR>` element is described above, it indicates in which Research Unit is located the author (since there can be more than one author, more than one UR could be given, but this is refused by LaTeX). The quantities `<projet>` and `<theme>` define the name of the Team and its theme (this can be repeated). We have then `<title>` and `<etitle>`, the title of the report, in French and English, we have `<motcle>` and `<keyword>`, the keywords in French and English. The content could be more elaborate: for instance we could allow math in the title or keywords. We have also `<resume>` and `<abstract>`, the French and English abstract (paragraphs are allowed here; the abstract should be less than half of a page in length). The element `<auth>` defines a single author, while `<author>` defines a list of authors.

```
<!ELEMENT title (#PCDATA|hi|LaTeX)* >
<!ELEMENT etitle (#PCDATA|hi|LaTeX)* >
```

```
<!ELEMENT resume (#PCDATA|hi|p|LaTeX)* >
<!ELEMENT abstract (#PCDATA|hi|p|LaTeX)* >
<!ELEMENT motcle (#PCDATA|hi|LaTeX)* >
<!ELEMENT keyword (#PCDATA|hi|LaTeX)* >
<!ELEMENT RRnumber (#PCDATA)* >
<!ELEMENT date (#PCDATA)* >
<!ELEMENT author (auth)* >
<!ELEMENT auth (#PCDATA)* >
```

The could be the document element for a research report.

```
<!ELEMENT rr (RRstart,div0*)>
<!ATTLIST rr language CDATA #IMPLIED type CDATA #IMPLIED>
```

9.2 The raweb2 DTD

This is the start of the file. Current version is 1.1.2.3, dated 2004/12/16.

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

For some reason, every element has a prefix; this is currently empty.

```
<!ENTITY % prefix "">
```

A name is given for the Xlink namespace.

```
<!ENTITY % XLINK.xmlns "http://www.w3.org/1999/xlink" >
```

The DTD starts by listing everything in alphabetic order.

```
<!ENTITY % anchor        "%prefix;anchor">
<!ENTITY % b             "%prefix;b">
<!ENTITY % big           "%prefix;big">
<!ENTITY % caption       "%prefix;caption">
<!ENTITY % code          "%prefix;code">
<!ENTITY % descriptionlist "%prefix;descriptionlist">
<!ENTITY % em            "%prefix;em">
<!ENTITY % LaTeX         "%prefix;LaTeX">
<!ENTITY % TeX           "%prefix;TeX">
<!ENTITY % object        "%prefix;object">
<!ENTITY % formula       "%prefix;formula">
<!ENTITY % glosslist     "%prefix;glosslist">
<!ENTITY % i             "%prefix;i">
<!ENTITY % identification "%prefix;identification">
<!ENTITY % keyword       "%prefix;keyword">
<!ENTITY % label         "%prefix;label">
<!ENTITY % li            "%prefix;li">
<!ENTITY % moreinfo      "%prefix;moreinfo">
<!ENTITY % projectName    "%prefix;projectName">
<!ENTITY % footnote      "%prefix;footnote">
<!ENTITY % orderedlist   "%prefix;orderedlist">
<!ENTITY % p             "%prefix;p">
<!ENTITY % br            "%prefix;br">
<!ENTITY % participants  "%prefix;participants">
<!ENTITY % person        "%prefix;person">
<!ENTITY % raweb         "%prefix;raweb">
<!ENTITY % ref           "%prefix;ref">
<!ENTITY % refperson     "%prefix;refperson">
```

```
<!ENTITY % presentation  "%prefix;presentation">
<!ENTITY % fondements    "%prefix;fondements">
<!ENTITY % domaine       "%prefix;domaine">
<!ENTITY % logiciels     "%prefix;logiciels">
<!ENTITY % ressource     "%prefix;ressource">
<!ENTITY % resultats     "%prefix;resultats">
<!ENTITY % contrats      "%prefix;contrats">
<!ENTITY % international "%prefix;international">
<!ENTITY % diffusion     "%prefix;diffusion">
<!ENTITY % shortname     "%prefix;shortname">
<!ENTITY % simplelist    "%prefix;simplelist">
<!ENTITY % simplemath    "%prefix;simplemath">
<!ENTITY % small         "%prefix;small">
<!ENTITY % span          "%prefix;span">
<!ENTITY % sub           "%prefix;sub">
<!ENTITY % subsection    "%prefix;subsection">
<!ENTITY % sup           "%prefix;sup">
<!ENTITY % strong        "%prefix;strong">
<!ENTITY % table         "%prefix;table">
<!ENTITY % team          "%prefix;team">
<!ENTITY % term          "%prefix;term">
<!ENTITY % td            "%prefix;td">
<!ENTITY % th            "%prefix;th">
<!ENTITY % tr            "%prefix;tr">
<!ENTITY % tt            "%prefix;tt">
<!ENTITY % theme         "%prefix;theme">
<!ENTITY % bodyTitle     "%prefix;bodyTitle">
<!ENTITY % topic         "%prefix;topic">
<!ENTITY % UR            "%prefix;UR">
```

This list comes from a second file.

```
<!ENTITY % address       '%prefix;address'>
<!ENTITY % contact       '%prefix;contact'>
<!ENTITY % email         '%prefix;email'>
<!ENTITY % firstname     '%prefix;firstname'>
<!ENTITY % lastname      '%prefix;lastname'>
<!ENTITY % status        '%prefix;status'>
```

A decoration is either , <tt>, <i>, <b>, <big>, <small>, <sub>, <sup>, or . This corresponds to the <hi> element of the old DTD. A inline object is a decoration, or a <ressource>, <ref>, <code>, <anchor>, <formula>, <br>, , <LaTeX>, or <TeX>. A list is <glosslist>, <orderedlist>, <simplelist>, or a <descriptionlist>. A doc-block is a <table>, a list, or a <p>. Finally, a block is a doc-block, a <footnote> or a <object>.

```
<!ENTITY % decoration "%em; | %tt; | %i; | %b; | %big; | %small; | %sub; |
   %sup; | %strong; ">
<!ENTITY % inline "%decoration; | %ressource; | %ref; | %code; | %anchor; |
   %formula; | %br; | %span; | %LaTeX; | %TeX;" >
<!ENTITY % list "%glosslist; | %orderedlist; | %simplelist; | %descriptionlist;">
<!ENTITY % doc-block "%table; | %list; | %p;">
<!ENTITY % block  "%doc-block; | %footnote; | %object;">
```

We define here two entities; they say that the id attribute is required or optional. In any case it is an ID.

```
<!ENTITY % id   "id          ID      #IMPLIED">
<!ENTITY % id_r "id          ID      #REQUIRED">
```

We define an attribute that says that the language can be French or English. We declare also some attributes used for links.

```
<!ENTITY % xml-lang     'xml:lang   ( fr | en )             "en"        '>

<!ENTITY % xlink
        'xmlns:xlink    CDATA                   #FIXED "%XLINK.xmlns;"
         xlink:href     CDATA                   #REQUIRED
         xlink:type     CDATA                   #FIXED "simple"
         xlink:show     ( new | replace | embed) "replace"
         xlink:actuate  ( onLoad | onRequest)    "onRequest"'>
```

We declare here three files and include them. We have inlined the persons.mod file. We do not show the biblio.dtd file; it is a copy of the TEI, and is rather long.

```
<!ENTITY % persons-mod  SYSTEM "persons.mod">
<!ENTITY % mathmlDTD    SYSTEM "mathml2.dtd">
<!ENTITY % biblioDTD    SYSTEM "biblio.dtd">
%mathmlDTD;
%biblioDTD;
%persons-mod;
```

We define here some abbreviations. A listSection defines the list of the eight standard sections: we have <presentation>, <fondements>, <domaine>, <logiciels>, <resultats>, <contrats>, <international>, <diffusion>. These are the same sections as in the old DTD. A contentSection describes what you can put in a section: an optional <bodyTitle> followed by some <block> or <subsection>.

```
<!ENTITY % listSection "(%presentation;), (%fondements;)?,
   (%domaine;)?, (%logiciels;)?, (%resultats;), (%contrats;)?,
   (%international;)?, (%diffusion;)?">
<!ENTITY % contentSection "(%bodyTitle;)?, ((%block;)* | (%subsection;)*) ">
```

Here are the sections. They all have id as required attribute.

```
<!ELEMENT %presentation; ( %contentSection; )>
<!ATTLIST %presentation; %id_r;>

<!ELEMENT %fondements; ( %contentSection; )>
<!ATTLIST %fondements; %id_r;>

<!ELEMENT %domaine; ( %contentSection; )>
<!ATTLIST %domaine; %id_r;>

<!ELEMENT %logiciels; ( %contentSection; )>
<!ATTLIST %logiciels; %id_r;>

<!ELEMENT %resultats; ( %contentSection; )>
<!ATTLIST %resultats; %id_r;>

<!ELEMENT %contrats; ( %contentSection; )>
<!ATTLIST %contrats; %id_r;>

<!ELEMENT %international; ( %contentSection; )>
<!ATTLIST %international; %id_r;>
```

```
475
476 <!ELEMENT %diffusion; ( %contentSection; )>
477 <!ATTLIST %diffusion; %id_r;>
```

We define identSection to be the identification part of a section; this is a list of `<participants>`, a list of `<keyword>` and an optional `<moreinfo>`.

```
478 <!ENTITY % identSection "( (%participants;)*, (%keyword;)*, (%moreinfo;)?)">
```

A `<subsection>` is formed of an optional `<bodyTitle>`, an identSection, followed by blocks or `<subsection>`. It can have a topic attribute. Note: only children of a section can have a topic.

```
479 <!ELEMENT %subsection; ( (%bodyTitle;)?, (%identSection;),
480     ((%block;) | (%subsection;))* ) >
481 <!ATTLIST %subsection;
482         %id;
483         topic   IDREF   #IMPLIED>
```

The `<raweb>` is formed of the eight standard sections, in `%listSection` plus the bibliography, plus a `<identification>` part and a list of `<topic>` declarations.

```
484 <!ELEMENT %raweb; (%identification;, (%topic;)*, %listSection;, biblio) >
485 <!ATTLIST %raweb; year NMTOKEN #IMPLIED
486         %xml-lang;
487         xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
488         xmlns:html  CDATA #FIXED "http://www.w3.org/1999/xhtml"
489         %id;>
```

The `<identification>` element contains `<shortname>` `<projectName>`, `<theme>`, `<team>`, a sequence of `<UR>` and an optional `<moreinfo>`. It has a isproject attribute. A `<topic>` contains only text.

```
490 <!ELEMENT %identification; ( %shortname;, %projectName;, %theme;, %team;,
491       (%UR;)+, (%moreinfo;)?) >
492 <!ATTLIST %identification; isproject    (true|false) "true"  %id; >
493
494 <!ELEMENT %topic; (#PCDATA) >
495 <!ATTLIST %topic; %id_r;>
```

Trivial elements.

```
496 <!ELEMENT %theme; (#PCDATA)>
497 <!ATTLIST %theme;  id ID #IMPLIED>
498 <!ELEMENT %bodyTitle; (#PCDATA|%inline;)* >
499 <!ATTLIST %bodyTitle; id ID #IMPLIED>
500 <!ELEMENT %projectName; (#PCDATA|%inline;)* >
501 <!ATTLIST %projectName; id ID #IMPLIED>
502 <!ELEMENT %shortname; (#PCDATA|%inline;)* >
503 <!ATTLIST %shortname; id ID #IMPLIED>
```

An `<UR>` element is empty, but has a name attribute that identifies it.

```
504 <!ENTITY % listUR "Sophia|Rocquencourt|RhoneAlpes|Rennes|Lorraine|Futurs">
505 <!ELEMENT %UR; EMPTY>
506 <!ATTLIST %UR; name (%listUR;) #REQUIRED >
507 <!ATTLIST %UR; id ID #IMPLIED>
```

A `<moreinfo>` element can contain block, inline objects, footnotes.

```
508 <!ELEMENT %moreinfo; (#PCDATA | %doc-block; | %footnote; | %inline;)* >
509 <!ATTLIST %moreinfo; %id;>
```

A `<caption>` contains inline objects.

```
<!ELEMENT %caption; (#PCDATA|%inline;)* >
<!ATTLIST %caption; id ID #IMPLIED>
```

A element contains inline stuff. It has a style attribute that explains how it can be emphasized.

```
<!ENTITY % styles "highlight | underline">
<!ELEMENT %em; (#PCDATA | %inline;)*>
<!ATTLIST %em; style (%styles;) #IMPLIED>
<!ATTLIST %em; id ID #IMPLIED>
```

We define here some elements: , <i>, <tt>, <b>, <big>, <small>, <sub>, and <sup>. They contain inline stuff.

```
<!ELEMENT %strong;  (#PCDATA | %inline;)*>
<!ATTLIST %strong;  id ID #IMPLIED>
<!ELEMENT %i;       (#PCDATA | %inline;)*>
<!ATTLIST %i;       id ID #IMPLIED>
<!ELEMENT %tt;      (#PCDATA | %inline;)*>
<!ATTLIST %tt;      id ID #IMPLIED>
<!ELEMENT %b;       (#PCDATA | %inline;)*>
<!ATTLIST %b;       id ID #IMPLIED>
<!ELEMENT %big;     (#PCDATA | %inline;)*>
<!ATTLIST %big;     id ID #IMPLIED>
<!ELEMENT %small;   (#PCDATA | %inline;)*>
<!ATTLIST %small;   id ID #IMPLIED>
<!ELEMENT %sub;     (#PCDATA | %inline;)*>
<!ATTLIST %sub;     id ID #IMPLIED>
<!ELEMENT %sup;     (#PCDATA | %inline;)*>
<!ATTLIST %sup;     id ID #IMPLIED>
```

I don't understand this.

```
<!ELEMENT %span; (#PCDATA | %inline;)*>
<!ATTLIST %span; align (left|center|right) "left" id ID #IMPLIED >
```

The <code> element is unused by the Raweb, as well as <anchor>.

```
<!ELEMENT %code;   (#PCDATA | %inline;)*>
<!ELEMENT %anchor; EMPTY>
<!ATTLIST %anchor; %id_r;>
```

A <ref> element is a link. It can contain text. It has attributes.

```
<!ELEMENT %ref; (#PCDATA | %inline;)* >
<!ATTLIST %ref;
        location (intern | biblio | extern) "extern"
        %xlink; >
<!ATTLIST %ref; id ID #IMPLIED>
```

A <keyword> contains only characters (what about "λ-calculus" ?). The theme attribute is currently unused.

```
<!ELEMENT %keyword; (#PCDATA) >
<!ATTLIST %keyword; %id; theme CDATA #IMPLIED>
```

A <footnote> can contain arbitrary text (well, there are restrictions...). The place attribute is not used.

```
<!ELEMENT %footnote; (#PCDATA | %block;)* >
<!ATTLIST %footnote; %id; place CDATA "unspecified">
```

We have four types of lists. They contain all some `<li>` children; in the case of a glossary or description, we can have `<label>`. The purpose of the title attribute is unclear.

```
<!ELEMENT %simplelist; (%li;)+ >
<!ATTLIST %simplelist; %id; title CDATA #IMPLIED>

<!ELEMENT %orderedlist; (%li;)+ >
<!ATTLIST %orderedlist;  %id; title CDATA #IMPLIED>

<!ELEMENT %glosslist; (%label; , %li;)+ >
<!ATTLIST %glosslist; %id; title CDATA #IMPLIED>

<!ELEMENT %descriptionlist; (%label; | %li;)+ >
<!ATTLIST %descriptionlist; %id; title CDATA #IMPLIED>
```

A `<label>` in a list contains only inline elements, while a `<li>` can be more complicated.

```
<!ELEMENT %label; (#PCDATA|%inline;)* >
<!ATTLIST %label; %id;>

<!ELEMENT %li; (#PCDATA | %doc-block; | %inline;)* >
<!ATTLIST %li; %id;>
```

A `<table>` contains some `<tr>` and an optional `<caption>`. It can have a title attribute. A `<tr>` contains some `<th>` and `<td>`. These two elements have the same structure. However, a `<td>` element can have a style attribute.[2]

```
<!ELEMENT %table; ((%tr;)+, (%caption;)?) >
<!ATTLIST %table;  %id; title CDATA #IMPLIED >
<!ELEMENT %tr; (%th; | %td;)+>
<!ATTLIST %tr; %id; >
<!ELEMENT %td; (#PCDATA | %doc-block; | %inline;)*>
<!ATTLIST %td; style CDATA #IMPLIED %id; >
<!ELEMENT %th; (#PCDATA | %doc-block; | %inline;)*>
<!ATTLIST %th; %id; >
```

This defines a `<p>` element.

```
<!ELEMENT %p; (#PCDATA | %inline; | %footnote;)*>
<!ATTLIST %p; %id;
              noindent CDATA #IMPLIED
              rend CDATA #IMPLIED>
```

A `<pers>` is formed of an optional `<firstname>`, `<lastname>`, an optional `<contact>`, an optional `<status>`, and an optional `<moreinfo>`. .

```
<!ELEMENT %person; ( (%firstname;)?, %lastname;, (%contact;)?,
   (%status;)?, (%moreinfo;)?) >
<!ATTLIST %person;       %id;>
```

The `<firstname>` and `<lastname>` elements, as well as `<email>`, contain characters. The content of `<contact>` is email or address; the adress is defined by the TEI[3].

```
<!ELEMENT %firstname; (#PCDATA) >
<!ATTLIST %firstname; %id;>
<!ELEMENT %lastname; (#PCDATA) >
<!ATTLIST %lastname; %id;>
```

[2] How is this attribute defined?

[3] It is too complicated to explain here

```
<!ELEMENT %contact; (%email; | %address;)* >
<!ELEMENT %email; (#PCDATA) >
```

The <status> element contains an optional <moreinfo>. Its type attribute is an element of list not shown here (I don't understand the purpose of the list).

```
<!ENTITY % listStatus "Professor... None">
<!ELEMENT %status; (%moreinfo;)? >
<!ATTLIST %status; type (%listStatus;) #IMPLIED >
```

The <team> section contains some <participants> elements, and an optional <moreinfo>.

```
<!ELEMENT %team; ( (%participants;)+, (%moreinfo;)?)>
<!ATTLIST %team; %id;>
```

A <participants> element contains some <person> or <refperson> elements, followed by an optional <moreinfo>.

```
<!ELEMENT %participants; ( (%person; | %refperson;)+, (%moreinfo;)?)>
<!ATTLIST %participants; category  CDATA #IMPLIED id ID #IMPLIED>
```

A <refperson> is a link to a <person>.

```
<!ELEMENT %refperson; EMPTY>
<!ATTLIST %refperson; ref IDREF #REQUIRED>
```

This defines an <object> and a <ressource>. An object can be used to include a floating figure.

```
<!ELEMENT %object; ( (%bodyTitle;)?, (%table;)+, (%caption;)? ) >
<!ATTLIST %object; %id_r;>
<!ELEMENT %ressource; ( (%caption;)? ) >
<!ATTLIST %ressource;
        %id;
        %xlink;
        media (WEB | PRINT ) "WEB"
        width NMTOKEN #IMPLIED
        height NMTOKEN #IMPLIED
        preview CDATA #IMPLIED
        type (inline|display) "inline"
        framed CDATA #IMPLIED
        scale CDATA #IMPLIED
        angle CDATA #IMPLIED>
```

Math formulas are defined like in the old DTD.

```
<!ELEMENT %formula;  (%simplemath; | %math.qname;) >
<!ATTLIST %formula;  %id;  type (inline|display) "inline" >
<!ELEMENT %simplemath; (#PCDATA | %inline;)* >
<!ATTLIST %simplemath;  %id;  type (inline|display) "inline" >
```

9.3 The classes DTD

We use the classes DTD for converting the Tralics examples and documentation into XML and HTML. It is a modification of the raweb DTD, hence we show here only the differences. It is hightly experimental.

These defintions are needed to 'rawxml' example.

```
<!ENTITY Dollar  "&#x24;" >
<!ENTITY Euro  "&#x20AC;" >
<!ENTITY Equals  "&#x3D;" >
```

The raweb does not use <div0> and <div1>. Its DTD defines the element and the content, but we forget to define the attribute list. It is however important that the style sheets sees the 'ID' type of the id of the element.

```
<!ATTLIST div0 %tei-div-atts; >
<!ATTLIST div1 %tei-div-atts; >
```

We added a <div5> element, that can be converted into a <h6> HTML element. What about <div6>?

```
<!ELEMENT div5  (head, %ramodule-header;, %tei-aux;) >
<!ATTLIST div5 %tei-div-atts; >
```

An <index> element has three attributes. We declare here only target. Our style sheet does not work without it.

```
<!ELEMENT index (#PCDATA | %texte-general; |  p )* >
<!ATTLIST index  target IDREFS #REQUIRED>
```

Index

In this index, the page number of a command refers often to the start of the verbatim block where it is used or defined. In many cases, you will find the command one page after the number given in the index.

Bibliography

[1] David Carlisle. XMLTEX: A non validating (and not 100% conforming) namespace aware XML parser inplemented in TEX. *TUGboat*, 21(3):193–199, 2000.

[2] David Carlisle, Patrick Ion, Robert Miner, and Nico Poppelier (editors). Mathematical Markup Language (MathML) Version 2.0. http://www.w3.org/TR/MathML2/, 2001.

[3] Consortium WWW. La spécification xml. *Cahier Gutenberg*, (33-34), 1999. This is a French translation of the specifications, http://www.w3.org/TR/1998/REC-xml-19980210.

[4] Michael Kay. *XSLT, Programmer's Reference*. Wrox Press Ltd, 2nd edition, 2001.

[5] Michael Kay. *XSLT 2.0, Programmer's Reference*. Wrox Press Ltd, 3rd edition, 2004.

[6] Donald E. Knuth. *The TEXbook*. Addison Wesley, 1984.

[7] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The LaTeX companion, second edition*. Addison Wesley, 2004.

[8] The Unicode Consortium. *The Unicode Standard, version 4.0*. Addison Wesley, 2003.

[9] W3C. Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/REC-xml/, 1998. Third edition published in 2004.

[10] W3C. Extensible Markup Language (XML) 1.1. http://www.w3.org/TR/xml11/, 2004.

[11] Web consortium. Namespaces in XML. http://www.w3.org/TR/1999/REC-xml-names-19990114, 1999.

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803