



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

*Tralics, a  $\text{\LaTeX}$  to XML translator*  
*Part I*

José Grimm

**N° 309 — version 3**

initial version September 2005 — revised version April 2008

Thème NUM



*R*apport  
*technique*



# Tralics, a $\text{\LaTeX}$ to XML translator

## Part I

José Grimm\*

Thème NUM — Systèmes numériques  
Projet Apics

Rapport technique n° 309 — version 3 — initial version September 2005 — revised version April 2008 — 313 pages

**Abstract:** In this paper we describe *Tralics*, a  $\text{\LaTeX}$  to XML translator. A previous version of the software (written in Perl) was used to obtain the Pdf version of Inria’s “Rapport d’Activité” for year 2001. The current version of the software (written in C++) is used for both the HTML and Pdf version since year 2002: the XML generated by *Tralics* is conforming to a local DTD, similar to the TEI; it is converted to Pdf via  $\text{pdf\TeX}$  and the `xmltex` package, and the HTML via a xslt processor.

We explain here the philosophy of the software, its usage, its limitations, and how to customize it. All commands defined by *Tralics* are explained in this document, for most of them, we give an example of use. The index lists all commmands, environments, and options.

Version 2 of this document holds additions for *Tralics* 2.9. Version 3 of this document holds additions for *Tralics* 2.12.

**Key-words:** Latex, XML, HTML, MathML, Perl, PostScript, Pdf

\* Email: [Jose.Grimm@sophia.inria.fr](mailto:Jose.Grimm@sophia.inria.fr)

# Tralics, un traducteur de $\text{\LaTeX}$ vers XML

## Partie I

**Résumé :** Dans cet article nous décrivons le logiciel *Tralics*, un traducteur de  $\text{\LaTeX}$  vers XML. Une version antérieure de ce logiciel, écrite en Perl, a été utilisée pour générer la version Pdf du Rapport d'activité de l'Inria en 2001. La version actuelle du logiciel, écrite en C++, a été utilisée pour obtenir à la fois le HTML et le Pdf depuis 2002 : nous avons utilisé une DTD locale, similaire à la TEI, et pdf $\text{\TeX}$  plus *xm $\text{\LaTeX}$*  pour obtenir le Pdf.

Nous expliquons ici la philosophie de *Tralics*, son usage, ses limitations, et comment paramétrer le logiciel. Toutes les commandes définies par *Tralics*, sont expliquées, pour la plupart d'entre elles on donne un exemple d'utilisation. Un index regroupe l'ensemble des commandes, environnements et options.

La version 2 de ce document contient des mises à jour pour *Tralics* 2.9. La version 3 de ce document contient des mises à jour pour *Tralics* 2.12.

**Mots-clés :** Latex, XML, HTML, MathML, Perl, PostScript, Pdf

# Chapter 1

## Introduction

The Tralics software was designed as a tool for the Raweb. In this chapter, we explain some of our motivations. In the next chapter, we study some T<sub>E</sub>X commands and explain how they are handled in the same fashion by Tralics. Following chapters explain some differences, merely because XML is not dvi. In a final chapter, we explain how to configure Tralics. There is a second part, that explains how the XML files can be used, converted into Pdf or HTML; it describes also the Raweb DTD. The last chapter of the second part describes additions to the program made since 2007.

### 1.1 A short history of the Raweb

A short history of the Raweb may be found on the Inria internal web site<sup>1</sup>. The question concerns Inria's Annual Activity Report, also known as "Rapport d'activité", or "Annexe technique" to the RA or "annexes scientifiques" to the RA. This is a document, written by the research teams, at the end of the year  $N$  (October, November), and published in March of year  $N + 1$ .

Until the 1993 edition (published in 1994), only a paper version existed. A L<sup>A</sup>T<sub>E</sub>X model was used since 1987, designed by Jacques André then Martin Jourdan. See the reference [5], by Louarn in the first Cahiers Gutenberg.

In 1993, contacts were made with the Grif S.A. society, for the design of a SGML DTD and a L<sup>A</sup>T<sub>E</sub>X-to-SGML converter<sup>2</sup>. As a result, Philippe Louarn was able to put on the web the RA (year 1994) in its HTML version<sup>3</sup>. But this converter was judged too complicated (rules were too strict) and for several years, the HTML was directly produced from the L<sup>A</sup>T<sub>E</sub>X source, using `latex2html`.

In 1996, a working group (conducted by Albert Benveniste) gave new specifications: independent modules, grouped into ten sections, etc. A technical group was created (conducted by Gérard Paget), whose objective was to find a company that could sell a software (maybe using XML as intermediate language). None was found, but the design of modules (in L<sup>A</sup>T<sub>E</sub>X syntax) was well-defined by Laurent Pierron and José Grimm with the aid of Marie-Pierre Durolet and Jean-Claude Le Moal. For the Ra98, a Perl script did some preprocessing, splitting the L<sup>A</sup>T<sub>E</sub>X source into modules (one module per HTML page). The author wishes to thank all these people (including A. Quadrat), who gave him the idea to work on L<sup>A</sup>T<sub>E</sub>X and write a translator.

In 1999, the Scientific Annexes to Inria's Annual Report were renamed RAWEB, to emphasize the role played by the Web (it is available as a CD-Rom, but no more printed by Inria).

<sup>1</sup>See <http://www.inria.fr/interne/disc/apropos/chantiers/raweb-xml/histoire.html>

<sup>2</sup>Grif stands for 'GRe noble Interactive Formatter'; further developments of this editor by the Opera team led to the Amaya software.

<sup>3</sup>See <http://www.inria.fr/rapportsactivite/RA94/RA94.html>

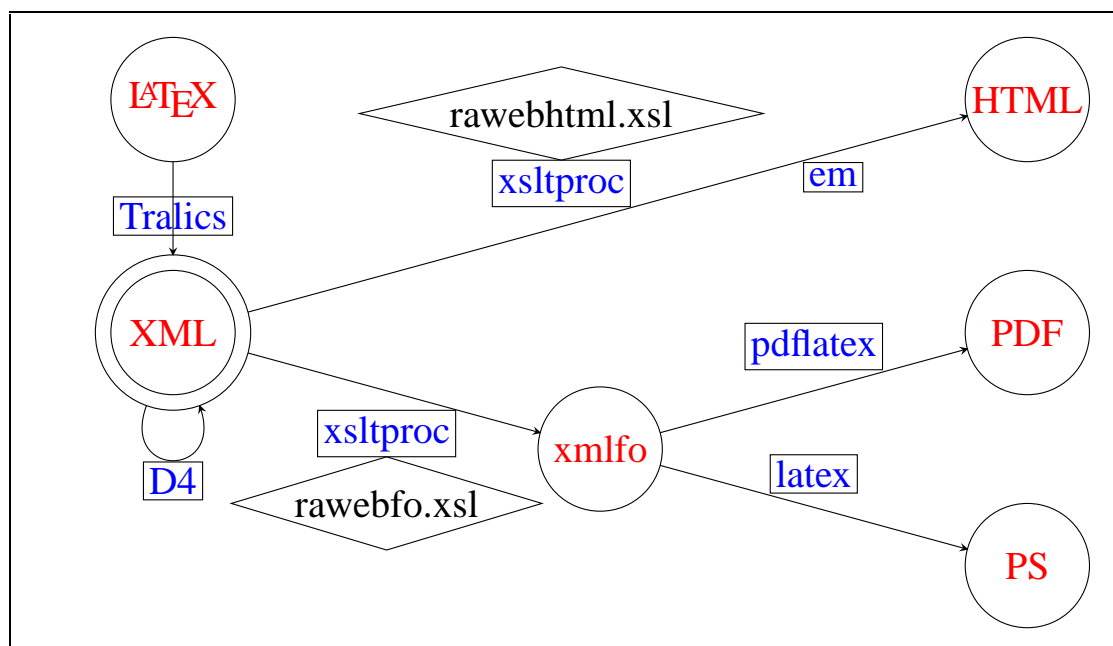


Figure 1.1: A diagram that explains how the Raweb operates. Rectangular boxes contain tools, diamond-shape boxes are style sheets, and circles contain language names. The name XML is in a double circle, it is the central object; the arrow labeled ‘D4’ that connects it to itself indicates conversion from one DTD to the other, used in 2004. The box containing ‘em’ represents the Perl script `extract-math.pl` that handles the math formulas; it uses tools borrowed from `latex2html`. The diagram is written in ‘pgf’, a format that Tralics cannot interpret yet.

In 2001, the Perl scripts mentioned above evolved into a  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ -to-XML converter (some ideas were borrowed from `latex2html`, which is also a Perl script). The main trouble was conversion from XML to Pdf, and we used tools from the  $\text{T}_{\text{E}}\text{X}$  community (by S. Rahtz and D. Carlisle [1]) and `pdf $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$` . On the other hand, the images in the HTML files were converted by `latex`. Additional software (for creation on a global index, etc.) was written/used by Marie-Pierre Durollet.

This gave a complicated object: a Perl script, that converts a  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  file into XML (using Omega as a subprocess for the math) followed by another Perl script that extracts the math, calls `latex`, then `dvips`, then `pstoimg` (a Perl script that calls `ppmquant`, `pnmcrop`, and so forth, whose job is to produce a `png` image for the math formula), and finally an XSLT processor for the effective conversion. This became even more complicated in 2004, where a new DTD was introduced (designed by Bruno Marmol and people mentioned above), hence a XML-to-XML translator. See Figure 1.1.

## 1.2 Birth of Tralics

The big Perl script was rewritten as a C++ translator, renamed Tralics, and got (for version 1.6) a first IDD<sup>4</sup> number in December 2002. This software was still able to produce a  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  preview of the Raweb: The source is read, the syntax is tested, a  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  file is written for each module, `latex` and `bibtex` are called, the resulting dvi is converted to PostScript. But the same Tralics

<sup>4</sup>InterDepositDigitalNumber = IDD<sup>4</sup>.FR.001.510030.001.S.P.2002.000.31235 is the IDD<sup>4</sup> for version 2.9.4, February 2007

can be used in a different way: the source is read, the syntax is tested, an XML file is created, an XSLT processor is called to generate the XSL/Format, and `pdflatex` generates the Pdf (you can also generate an HTML version).

Since 2003, there are a few people writing their RA directly in XML. As a consequence, the new Tralics, that is used in 2004, does only the bare minimum: it converts the  $\LaTeX$  code into XML code. There is a Perl script that does everything else (calling external programs like `latex`, `xsltproc`, etc). A non-obvious point concerns the math and the images. For the math, see above; for the images, a Perl script (the same as above) is used for conversion from PostScript to `png`. In some cases, the image already exists in `png` format and it is unwise to re-create it.

In 2007, the name of the Raweb changed; it is now RalyX. Here is a quote from the Web page:

The objective of the RalyX<sup>5</sup> project was to publish and exploit dynamically the annual INRIA activity reports; Ralyx is based on the Xylème<sup>6</sup> system, a native XML database. Xylème stores the XML version of the activity reports and supports queries to the reports involving both their structure and their content.

The first objective was to offer a tool for browsing the activity report with the same interface as the legacy HTML version. However, thanks to Xylème, pages and links are no longer statics, but computed on the fly, which can offer more flexibility in the future (different styles of display, editing along the year, etc).

### 1.3 Main objectives

The main objectives of Tralics are described in [3]. Recall that we want a program that converts from one language to another, such that

- the result is easily analyzable,
- the structure is preserved,
- the translation is univocal.

By the very choice of the XML language as target, the first objective is automatically satisfied. The only little problem concerns spaces: in which cases are spaces used as delimiter or as text? The translation of `'\par_x_{ }_{ }x\par'` is `'<p>x_{ }_{ }x</p>'`, and this is often interpreted in the same fashion as a single space.

The third item has the following meaning: the translation of `'\foo'` and `'\bar'` should be different in case these objects have different meanings in  $\LaTeX$ , but should be the same otherwise. One could argue that some tokens could be translated more than once in a different context (remember the `\texorpdfstring` command). We could imagine special rules for the bibliography; in fact, in math mode you must use a different function to add attributes to an element. On the other hand, we have an application where `'\tt'` and `'\bf'` are treated alike: in fact, all font information is removed, by redefining element names.

The important point is preservation of the structure. This means of course that the initial document has some structure and the target has one also. Clearly the structure of an XML document is given by its DTD, but this is completely ignored by Tralics. The first version of the program was very strict; nowadays, in your document, you can omit the `'\documentclass'`, the `'\begin{document}'`, or the `'\end{document}'`.<sup>7</sup>

<sup>5</sup>Web site is <http://ralyx.inria.fr>

<sup>6</sup>Web site is <http://www.xyleme.com>

<sup>7</sup>Put 'Hello, word!' in a tex file, compile, and admire the result!

There are some implicit rules, for instance that ‘\section’ terminates a ‘\subsection’, the occurrence of a character in vertical mode triggers the start of a <p> element, etc.

In [3], we wrote that the document should satisfy three kinds of assumptions: some technical assumptions (like: there is no ‘\0’ in the document), some validity assumptions (the document can be compiled by L<sup>A</sup>T<sub>E</sub>X without errors, it respects the rules of the Raweb as given on the web pages), and more general, unwritten, rules. The current translator is more flexible: there is no restriction on the syntax of commands, except that null characters are not allowed, plain T<sub>E</sub>X documents are translated (for instance the file xii.tex by D. Carlisle that starts with \let~\catcode~‘76 and ends with Yer.W, :jbye), and we have an application where only one part of the document is translated: only code before ‘\maketitle’ is used. In this document we shall explain these things.

## 1.4 Notes on the distribution

The Tralics software is formed of five different types of files. On one hand we have the documentation; this is formed of a sequence of L<sup>A</sup>T<sub>E</sub>X documents (like the one you are reading), converted by Tralics into XML then HTML, together with a sequence of HTML pages that describe all commands, features, and packages. These files have no explicit Copyright notice (so that default rules apply). The distribution contains a lots of tools that can be used for the conversion from XML to XML, HTML, or Pdf, written by various authors, with different copyright notices.

The Tralics executable can be obtained by compiling the sources; some precompiled versions are also available. Some configuration files are also provided (files with extension .tcf, .clt, or .plt). The binary may contain a path to these auxiliary files, if this is not the case, the path must be given at runtime to the program. Sources are distributed according to the CeCILL Free Software Licensing Agreement. This gives anybody the right to modify the sources, redistribute the software, and include it (or part of it) into another Software (see the Copyright Notice for details).

Finally, the distribution contains some test files. These are examples of source code, and their XML translation. Most of these file compile without error, but some files test the error handling mechanism. These files are distributed under the same conditions as the source files, and it is assumed that whoever changes the sources modifies the test files accordingly.

## 1.5 An example

Let’s consider the following piece of code.

```
\left [1=2\right\]$ est une \emph{formule de mathématiques}~!
```

You would expect this to be understood by L<sup>A</sup>T<sub>E</sub>X as ‘[1 = 2] est une *formule de mathématiques*!’ but<sup>8</sup> the result is ‘Hello, world!’, since we have preceded it by this (curious) list of commands

```
\catcode‘\$=\active\def$#1~{\catcode‘\$=3 Hello, world} %$macs
```

In T<sub>E</sub>X, translation depends on a great number of tables, that assign values to numeric or symbolic quantities. For instance, each character (a number between 0 and 255) in each font (for instance \OT1/cmr/m/it/10) is associated to its dimensions, ligature informations, etc. Our example (with a normal dollar sign) uses three fonts: a math font, an italic font and an upright font. Often, these tables are read only by T<sub>E</sub>X, in some other cases a user might look at them, and sometimes they are designed to be modified. For instance, the slant value of the current font is in general ignored (except by commands like ‘\emph’). A very important table holds the meaning of commands: they may be predefined (like ‘\left’), defined in a format (like ‘\emph’), or defined by the user (there are a great number of mathematicians that define a command named ‘\RR’ for the set of real numbers).

<sup>8</sup>In the HTML version, the delimiters around 1=2 are not shown if the browser is not in MathML mode



An important table is the table of category codes. Each character has a category code, an integer between 0 and 15. For instance, ‘e’ has category letter, ‘é’ has category active (so that it is equivalent to ‘\’e’)<sup>9</sup>, backslash, open brace, close brace and dollar sign are of category 0, 1, 2, and 3 respectively. A command like ‘\left’ is formed by an introducer (any character of category 0, its value is irrelevant) and a sequence of letters (in the example, the space that follows is not part of the command, since its category code is 5; this space will be read again, and ignored, unless its category code changes; such a change can be triggered by the evaluation of the command); while a command like ‘\}’ or ‘\\$’ is formed of an introducer and a single non-letter character (when followed by spaces, these spaces do not disappear). Writing a T<sub>E</sub>X scanner is easy, the only difficulty is that category codes can change (for instance, in verbatim mode). In the example, the tokens are the following: \$<sub>3</sub> left [ <sub>12</sub> 1<sub>12</sub> =<sub>12</sub> right [ <sub>1</sub>, etc, s<sub>11</sub> }<sub>2</sub> ~<sub>13</sub> !<sub>12</sub>. Note that a closing brace, with default category code is shown as ‘}’<sub>2</sub>, while a command whose name is formed of a closing brace is shown as ‘}’<sub>2</sub>. If you say \let\foo\bar, and ask Tralics for the meaning of \foo, as in \show\foo, it knows the name ‘foo’ of the token and its value (this is a command code) and gets the name ‘bar’ from a lookup table, and adds the current value of the escape character in front. If this is the plus sign, you will see +foo=+bar.

Parsing an expression means finding for each command its arguments. In the example, the two active characters é and ~ take no argument. The \active command takes no argument either. The \emph command takes one argument<sup>10</sup>: a token, or a token list delimited by braces (i.e. characters with category codes 1 and 2), these braces are not part of the argument. The two commands \left and \right are special: they want a delimiter, in reality a pointer into a special slot in the current math font; the argument cannot be delimited by braces. The syntax of \catcode is more complicated. In fact, it is an instance of <codename><sup>11</sup>, and <codename><8-bit number> is something that can follow \the or be used as <internal integer>, while a <code assignment> is <codename> <8-bit number> <equals> <number>. We shall explain this in details later; the idea is that, depending on the context, the \catcode command returns a value stored in a table or modifies it. In the case of

```
\catcode'\$=\active
```

the <8-bit number> is the internal internal code of the dollar sign, expressed in the form ‘\\$', and the <number> is another character code<sup>12</sup>. Expanding a command means (roughly speaking) reading its arguments, and replacing it with the body of the command (where special markers like ‘#1’ have been substituted by the value of the arguments). In some cases, internal tables may be consulted, but they are never modified. Evaluating a command implies modification of some internal state variables (for instance, a character can be added to the current character list, or a complete paragraph split into lines, or a register modified). In this example, the dollar character becomes magically active: a dollar character is no more read as \$<sub>3</sub> but as \$<sub>13</sub>.

The syntax of \def and friends is <def><control sequence><definition text> where <definition text> is <parameter text><left brace><balanced text><right brace>. In the case of

```
\def$#1~{\catcode'\$=3 Hello, world}
```

the <control sequence> is the dollar sign (the object to be defined, a command or, as in this example, an active character), the <parameter text> is everything before the open brace, here ‘#1~’, and the <balanced text> is everything between the braces. Evaluation of ‘\def’ consists in storing in a table the <parameter text> and the <balanced text> (T<sub>E</sub>X stores also a special marker representing the <left brace>). What happens now when T<sub>E</sub>X sees a dollar sign? since this character is active, the definition given above applies. The <parameter text> explains how to read arguments. In this case, ‘#1~’ means that there is one argument, everything up to (but not including) a tilde character. In the case of ‘\$\left....\$. . . }~!’’, all characters are read (except the exclamation point). They are

<sup>9</sup>In reality, this depends on the input encoding. In Tralics, there are very few active characters.

<sup>10</sup>Since \emph is robust, it takes no argument, but the non-robust command associated to it reads the argument. In any case an argument is read.

<sup>11</sup>We use angle brackets for the formal syntax of T<sub>E</sub>X as explained in the T<sub>E</sub>Xbook, [4].

<sup>12</sup>In fact, you should use \active only in the scope of a \catcode, so you can forget that ‘\active’ typesets as ‘,!’

replaced by the body (no substitution is needed). After that,  $\text{T}_{\text{E}}\text{X}$  sees `\catcode` and evaluates it as before, so that the dollar sign becomes a math shift character again. The space after the digit 3 disappears, and we are left with ‘Hello, word!’. Note: the example is given in French, first in order to show how 8-bit characters can be used, and also because, in English, there is no tilde before an exclamation point. In general, when a macro reads a delimited argument and sees an empty line instead of the delimiter, it signals an error of the form *Runaway argument? Paragraph ended before \$ was complete*.

Translation difficulties. There are different kinds of  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  to HTML/XML translators. Some, like `gellmu`<sup>13</sup> use a syntax of their own; others, like `tex4ht` use  $\text{T}_{\text{E}}\text{X}$  as preprocessor, thus understand the full syntax; there are translators like `tth` or `hévéa` that use a fixed (and efficient parser), or like `latex2html` (written in Perl) that use pattern matching, and global substitutions instead of sequential evaluation. Neither of these is perfect. We explain in this paper how `Tralics` deals with a certain number of problems.

## 1.6 Some remarks on the Translation

In  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ , commands can be defined in five places:

1. In the Pascal source. In this case, the command is a primitive, for instance ‘`\def`’ or ‘`\left`’. No equivalent might exist in the target language (for instance, you cannot define anything in a XML document, the result of ‘`\left`’ is an attribute of the `<mfenced>` element that results of the translation of the group implied by `\left`. And what about `\dump`?)
2. In the format file. In this case, the command is defined in a file (for instance `latex.ltx`), analyzed by  $\text{T}_{\text{E}}\text{X}$  and stored on disk using a fast retrieval method. For instance, plain  $\text{T}_{\text{E}}\text{X}$ ,  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ , `ConTEXt` have their own format file. Both plain  $\text{T}_{\text{E}}\text{X}$  and  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  define a command named ‘`\item`’, in a different way, for the same purpose (the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  command must be used inside a special environment, the plain  $\text{T}_{\text{E}}\text{X}$  one can be used everywhere).
3. In a class file. A class file defines markup commands for a generic purpose (a book, an article, a presentation). For a book, you have a frontmatter, for an article, you can have a title page, in both you have sections, etc. The class defines also the current font, together with a lot of dimensions.
4. In a package. The difference between a class and a package was introduced by  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ . Each document uses a single class, and lots of packages (this document uses the ‘report’ class, and the ‘RR’ package for the look; it uses ‘hyperref’ for hyper-links, ‘amsmath’ for the mathematical examples, ‘fancyvrb’ for the verbatim examples, etc.). Note that the plain  $\text{T}_{\text{E}}\text{X}$  format provides a macro `\proclaim` for theorems, while the `amsmath` package provides a command `\newtheorem` for defining theorems.
5. In the  $\text{T}_{\text{E}}\text{X}$  source, or files included via `\input`. The  $\text{T}_{\text{E}}\text{X}$  source may redefine commands defined earlier; a package may redefine commands from the class, but should not redefine commands from other packages (i.e., the order in which packages are loaded should have no importance).

If one is to design a translator, the question is: which commands to translate? and how? Our idea is that all  $\text{T}_{\text{E}}\text{X}$  primitives should be understood (the difference between ‘`\dump`’ and ‘`\mydump`’ is that you get either an *Unimplemented* or *Undefined* command error), as well as all standard  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  commands; of course all user-defined commands are expanded. Concerning classes and packages, our model (the `Raweb`) looks like a report. There are too many classes and packages

<sup>13</sup>Generalized Extensible  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ -Like Markup

for implementing them all. In earlier versions of the software, commands `\documentclass {foo}` or `\usepackage {bar}` did some action if the class `foo` or the package `bar` were known, and were ignored otherwise; we mentioned in the first version of this document interest in PhD thesis and slides classes, but no progress have been made in this direction.

In the current version of the software, commands `\documentclass`, `\usepackage` and all these are fully implemented. However, instead of reading `foo.cls` and `bar.sty`, Tralics reads `foo.clt`, and `bar.plt`. As an example, consider the `makeidx` package. The difference between the `sty` and `plt` files is in the definition of the `\printindex` command. The semantics of this command is: insert the index here. In the first version of Tralics there was no index, but nowadays multiple indexes (including glossaries) are allowed; the effect of the command is to mark a position in the XML tree, if omitted the index will be put at the end of the document; as a consequence, this cannot be a user-defined command. On the other hand, the L<sup>A</sup>T<sub>E</sub>X code asks to read the content of the index file (produced by `makeindex`) via `\@input@`; this a command that, as `\@input`, is used by L<sup>A</sup>T<sub>E</sub>X to read files that may be created after a first run (files of type `aux`, `toc`, `bbl`, `ind`, etc.), and is not implemented in Tralics (you run Tralics only once).

Another feature is worth mentioning. We have shown above some syntax rules; the source document uses the command `\syntax`, that takes one argument and put brackets around it; we followed Knuth's notations and used the character `\langle`. This produces a math formula with a single character and my my favourite Web browser (in non-MathML mode) produces an ugly result, so that a simple less-than sign has been used for the HTML version. You can define a command `\iftralics` in the same way as `\ifpdf` (checking for instance that `\tralicsversion` is defined), and write conditional code (an example will be given later). But we used the opportunity provided by Tralics that, if your file is `myfile.tex`, `myfile.ult` is read before the T<sub>E</sub>X source (by Tralics, and not L<sup>A</sup>T<sub>E</sub>X) and this file may contain command redefinitions (of course, if you want to redefine a not-yet-defined command, some care must be taken).

Let's assume that `\foo` is a command defined in one of these L<sup>A</sup>T<sub>E</sub>X files. Should it be translated by Tralics? and how? Notice first that a user command is always expandable, and has to be expanded; only the input stream is affected by this operation that consists in reading arguments and replacing them by an instantiated body. On the other hand, evaluation may modify or use internal tables, which can be implemented differently in our translator than in T<sub>E</sub>X (for instance, in pdfT<sub>E</sub>X, there is a way to re-use an image). Whenever a command is defined in a style file<sup>14</sup>, we could use the L<sup>A</sup>T<sub>E</sub>X source. In some cases, for efficiency reasons, we implemented them in C++ (for instance, the whole `fp` package has been re-written).

A typical example is the `\it` command. This is a non official L<sup>A</sup>T<sub>E</sub>X command (see [2], or [6, section 7.3.6]): "it is legitimate for you to redefine them in a package or in the preamble according to your personal taste". In Tralics, there is no difference between `'\it'` and `'\normalfont\itshape'`. What these commands do in L<sup>A</sup>T<sub>E</sub>X is rather complex: essentially, some variables are set and `'\selectfont'` is called. These commands are robust: if you use `'\it'` in a chapter title, the toc file<sup>15</sup> will contain the name of the command, not the result of the expansion. Since Tralics does not write anything in a toc file, this feature is not implemented. Using font changes in a title is not recommended: look at the table of contents for this section. You will see that the word 'Translation' is in a different font, although it is the same in the main document. The command `\texorpdfstring` has been used, otherwise pdfT<sub>E</sub>X complains with *Token not allowed in a PDFDocEncoded string*, because font changes are forbidden in bookmarks.

Consider now a command like `\motscle`. This is defined by the Raweb as an environment (you can use it in Tralics only as an environment) and it expands to something like `'{\bf mots clés}~'`. We do not want Tralics to use this expansion. One reason can be that the post-processor might prefer something like `'\textbf{Mots clefs}~:'` (i.e., use an alternate spelling, use an

<sup>14</sup>Sometimes packages are called style files, because this was the original term, and the extension is `.sty`.

<sup>15</sup>L<sup>A</sup>T<sub>E</sub>X writes in this file everything needed to produce a Table of Contents; the content of the file can be inserted at the start or end of the end document.

initial capital, add punctuation, etc). Another reason is that, since 2003, the Raweb is in English, and the translation should be an English word. For this reason, the translation of `\motcles` is `<keywords>`. The raweb class file contains commands that should not be translated: for instance, there is a command `\ra@finpart` whose purpose is to add a period after the last keyword. This should not be done by the translator, but by the program that will typeset the XML result.

Some  $\TeX$  primitives are hard to translate. For instance the `\'` command is assumed to put an acute accent over a character. It is defined (via an indirection through font encoding tables) in terms of the `\accent` primitive. We could translate `\'` into a Unicode combining character (U+301, to be placed after the character, see [7, paragraph 7.7]). But in general, we have a construction like `\'e` which is defined in iso-latin1, this is the Unicode character “latin small letter e with acute”. In the same fashion `\'k a` translates to `'a`, this could produce `&aogon;`, but the actual translation by `Tralics` is `&#x105;`. In a previous version of `Tralics`, a construction like `\'\^` was illegal. In the current version, we use a double indirection table. You can typeset the name Hàn Thê Thành of the author of pdf $\TeX$ , the input is `Th\'\{\^e}`, the `Tralics` output is `Th&#x1EBF;`. If you know the Unicode character value, you can enter it (in `Tralics` as `Th^^^^1ebf` or as `Th\char1EBF`).

A construction like `\font\myfont=cmt10 at 13pt` defines a command `\myfont` that can be used as `\myfont \char217w`. The effect of this command is to pick up two characters from the font `cmt10`, scaled by some ratio (it depends on the “at” size of the font), and typeset them. Note that the font could specify a ligature, so that the result could be a single character. In any case, the result might look very different from `Üw` (for instance  $\LaTeX$  provides a font containing only lines or circles). In the current version of `Tralics`, translation does not depend on the current font. The only interest of changing the current font with `\font` is that you can access or modify the internal tables of the font (but no metric file is read at all by `Tralics`).

In order to translate font changes correctly, you should define a command, for instance `\myw`, that uses character `w`, of the font ‘`myfont`’; the command could be more elaborate (it could look at the slant of the current font and select a slanted version of the character); the command could then be redefined for `Tralics`, either using a Unicode character, for instance `\char"1E09` if you want a `C` with cedilla and acute, or something else if the character is not defined by Unicode.

## 1.7 Category codes and characters

A category code is an integer between 0 and 16, as explained in the  $\TeX$ book; it is an attribute of a character, as used by the tokenizer. The codes are used according to the following table.

1. A character of category code 1 serves as group delimiter (opening character), as well as delimiter for macros and token lists. It is a `{` as used in the rules explained above. By default, the left brace character `{` is the only one with category 1.
2. A character of category code 2 serves as group delimiter (closing character), as well as delimiter for macros and token lists. It is a `}` as used in the rules explained above. By default, the right brace character `}` is the only one with category 2. The end of ‘`verbatim`’ environment is handled by the following piece of code, that shows how to change the category codes of the three characters mentioned above, and use alternate characters for the same purpose.

```
\begingroup \catcode '{=0 \catcode '['= 1
\catcode']=2 \catcode '\{=12 \catcode '\}=12
\catcode'\=12 |gdef|@xverbatim#1\end{verbatim} [#1]end[verbatim]]
\endgroup
```

3. A character of category 3 serves a math shift character. By default the dollar character `'$'` is the only character in this category. The same character can be used to start and end a

formula. Two consecutive character of category code 3 are used to start or end display math; the characters need not be the same.

4. A character of category code 4 is used as delimiter in arrays; it indicates the end of a cell. A typical row in a L<sup>A</sup>T<sub>E</sub>X array is `a&b&c\`; in plain T<sub>E</sub>X, it would finish with `\cr` or `\crcr`. The `&` character is the only one of category code 4.
5. A character of category 5 is a end-of-line character. When such a character is seen, all remaining characters on the current line are ignored. After that, the reader behaves as if it had seen nothing, a space, or a `\par` token. In some cases, the `\par` token is invalid. In Tralics, this space character is special, in that it may print as a new line character, in T<sub>E</sub>X, it is a normal space. By default, only the carriage return character is of category code 5 (this character is inserted at the end of every line, instead of the line feed, carriage return, or both, that marks the end-of-line in the file).
6. A character of category 6 can be used as delimiter in a command, or a table preamble. By default, there is one character of this category, the `#`. A typical array preamble is `\indent#\hfil&\quad#\hfil\cr` (this is the first example of `\halign` in the T<sub>E</sub>Xbook. We shall not describe T<sub>E</sub>X arrays here).
7. A character of category code 7 is a superscript character; by default, it is the hat character. Do not confuse it with `\^` that produces an accent. Such a character can be used only in math mode. There is also the double hat construction, explained later in section 5.1: two identical characters of category code 7 can be used to read any 8bit character as in `^^ab`; in Tralics, a 16 bit character can be read using 4 such characters.
8. A character of category code 8 (by default the underscore character) can be used as subscript character; as in the case of superscript characters, you can use it only in math mode.
9. A character of category code 9 is ignored. Tralics ignores no character by default. On the other hand, it cannot put a null character in a string, so that the null character will not appear in the XML output (note: the same holds for `\char0` and `^^00`).
10. A character of category 10 behaves like a space. By default, space, tabulation, character 160 behave like this (note that the character 160 is no-break space, it should be equivalent to `~`).
11. A character of category code 11 is a letter. By default, only characters in the range ‘a’ to ‘z’ and ‘A’ to ‘Z’ are of category 11.
12. A character of category 12 is an ‘Other character’. All characters not listed here are of category 12, including all digits.
13. A character of category 13 is an active character. Currently, there is only one active character, the `~`. An error is signaled in the case where an active character is used, but undefined. Tralics defines `_`, `#` and `&`, to be the same as `\_`, `\#` and `\&`.
14. A character of category code 14, is a comment character: all characters remaining on the current line are discarded. By default `%` is a comment character.
15. A character of category code 15 is invalid. By default, all characters are valid.
16. A character of category code 16 is a ‘short verb’ character. This is a feature that does not exist in T<sub>E</sub>X. If you use `\DefineShortVerb` to make it a short verb, you should undefine the character before changing its category code. Unexpected results can follow if non-ASCII characters are of category code 16.

17. A character of category code 0 is used to create a command like `\foo` or `\;`; the associated token is `\foo` or `\`; the value of the character is irrelevant. By default, there is only one character of category 0, the backslash.

In version one of this document, we started with category code 0, decremented the ‘enumi’ counter by one, before the first item, so that the first item label was zero. However `Tralics` does not use this counter in a ‘enumerate’ environment. Hence, the easiest solution, for having the same labels in the Pdf and HTML version, was to move this item to the end of the list.

If a `TEX` file contains the following lines,

```
\show{ \show} \show$ \show& \show# \show^ \show_ %$
\expandafter\show\space \show a \show 1 %
\def\foo+{}
\foo{ \foo} \foo$ \foo& \foo# \foo^ \foo_ \foo a \foo1
```

then `Tralics` will print:

```
begin-group character {.
end-group character }.
math shift character $.
alignment tab character &.
macro parameter character #.
superscript character ^.
subscript character _.
blank space .
the letter a.
the character 1.
```

In any case, we have a prefix that depends on the category code, then the value of the character. All calls to the command `\foo` are wrong and signal an error. We show here the first error message, followed by the “got” part of the other error messages. It is important to remember that the command has to be followed by the right token, or the right character with the right category code.

```
Error signaled at line 20 of file txt6.tex:
Use of \foo doesn't match its definition;
got {Character { of catcode 1},
expected {Character + of catcode 12}.
got {Character } of catcode 2}
got {Character $ of catcode 3}
got {Character & of catcode 4}
got {Character # of catcode 6}
got {Character ^ of catcode 7}
got {Character _ of catcode 8}
got a
got {Character 1 of catcode 12}
```

## 1.8 Considerations about mathematics

Translating mathematical formulas is rather difficult: this is because mathematics are complex in both `TEX` and `MathML`. Basically, you start with kernels, add some decoration, and connect these things. Kernels can be letters like `x`, `y`, `z` or `A`, `B`, `C`, but you see very often Greek letters like  $\alpha$ ,  $\beta$ ,  $\Gamma$ ,  $\Delta$ , Hebraic characters like  $\aleph$ , old German, like  $\varphi$ . Decoration can be, like in  $x^y$ , a second kernel on the left, the right, above, below. People use also bars, dots, rings, arrows, etc. There are

different types of connectors: for instance you can say  $x = y$  or  $x \parallel y$ , using parallel bars, or  $x \times y$ ,  $x + y$  using crossing bars. Amstex was designed by Spivak for easy typesetting of tensors of the form  $T_{j_1 j_2 \dots j_m}^{i_1 i_2 \dots i_n}$ .

The first difficulty consists in representing all these symbols. In T<sub>E</sub>X, you use commands like `\alpha`, defined as `\mathchar"10B`, whose effect is to chose a character from a font (depending on the packages used; a big difficulty is to chose math fonts that go well with the main text font of the document). Some characters exist in bold version, or in italics version. A non trivial question is how to put everything in bold: you could use commands like `\boldx \boldequals \boldalpha`. It would be simpler to say `\bold{\$x=\alpha\$}`. One solution is to say ‘`\boldmath`’<sup>16</sup>, or equivalently ‘`\mathversion{bold}`’ before starting a math formula; the effect is to use a bold version of everything (in fact, of all characters that use one of the math families, but the number of families is so small that not all symbols use this mechanism). You can say ‘`\mathbf{x}`’ or ‘`\bm{x}`’ in a math formula. Only `\mathbf` is implemented in Tralics, it provides a bold upright font; on the other hand, the `\bm` command is defined in the `bm` package; provided that fonts are available, it should use a bold version of everything (at least for all characters for which a bold typeface exists).

Traditionally, uppercase letters were upright, lowercase letters were italics. This is the default for Greek letters, but T<sub>E</sub>X uses italics for Roman letters; digits are by default upright uppercase, but lower case digits (also known as “oldstyle numbers”) can also be used. There are some exceptions to these rules: an operator like `\sin` is typeset using upright font. As said above, `\mathbf` produces upright characters. The so-called “black board” or “double-struck” characters obtained by `\mathbb` are upright, for instance  $\mathbb{R}$ . An interesting point: lots of people prefer  $\mathbb{R}$ , using `\mathbbm`. Some people fake it as  $\mathcal{R}$ , using `\mathcal{R}`. You can always use `\mathrm` for an upright character, as in `\mathrm{E}=mc^2`. Some packages provide italic uppercase Greek letters, and upright lowercase. Then you can apply the Laplace operator to the Delta function like this:  $\Delta\Delta$ .

In MathML, characters should not be used directly. For instance, you should use `<mn>125</mn>` for a number, `<mi>Foo</mi>` for the variable Foo and `<mo>sin</mo>` for the operator sin. These elements have a `mathvariant` attribute, which indicates which variant to use. In the case `<mo>`, the default is upright; in the case `<mi>`, the MathML norms says: “The default `mathvariant` would (typically) be `normal` (non-slanted) unless the content is a single character, in which case it would be `italic`.” So we could translate `\mathbb{R}` as an `<mi>` element containing a normal R with a `mathvariant` attribute of value `doublestruck`. An alternate solution consists in using a double struck character R, and no `mathvariant` attribute. Here are all the possible variants, and for the character A its Unicode value: `normal`, `bold` (1D400), `italic` (1D434), `bold-italic` (1D468), `doublestruck` (1D358), `bold-fraktur` (1D56C), `script` (1D49C), `bold-script` (1D4D0), `fraktur` (1D504), `sans-serif` (1D5A0), `bold-sans-serif` (1D5D4), `sans-serif-italic` (1D608), `sans-serif-bold-italic` (1D63C), `monospace` (1D670). In the case of Greek letters, Unicode knows the following variants: `bold`, `italic`, `bold italic`, `sans-serif bold`, and `sans-serif bold italic`. And in the case of digits: `bold`, `doublestruck`, `sans-serif`, `sans-serif bold`, `monospace`.

Note: there are some holes in the table, starting at 1D400. For instance the Laplace symbol (U+2112) looks like  $\mathcal{L}$ , but the the translation of `\mathcal{L}` (that T<sub>E</sub>X shows as  $\mathcal{L}$ ), is not 1D49C+11 (this character does not exists). Another special case: there are some variants of the Greek letters  $\epsilon$  and  $\varepsilon$ ; which one is the default is unclear.

There are two complementary views of MathML: presentation and content. This is how you would convert  $a = b$  using content markup: `<reln> <eq/> <ci> a </ci> <ci> b </ci> </reln>`. And this is how Tralics converts the same using presentation `<mrow> <mi>a</mi> <mo>=</mo> <mi>b</mi> </mrow>`. A more complex example, in presentation mode:

```
<apply>
  <int/>
```

<sup>16</sup>An example of bold math is when typesetting the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> logo, see page 23.

```

<bvar><ci>x</ci></bvar>
<condition>
  <apply><in/><ci>x</ci><ci type="set">C</ci></apply>
</condition>
<apply><sin/><ci>x</ci></apply>
</apply>

```

You can clearly see that we apply the `<int>` operation to some quantity (in which `<sin>` is applied to  $x$ ), subject to some condition (in which `<in>` is applied to  $x$  and some set). If you consider the expression  $\int_0^{\infty} \sin(x) dx$ , it is translated by Tralics as

```

<msubsup><mo>&int;</mo> <mn>0</mn> <mi>&infin;</mi> </msubsup>
<mrow>
  <mo form='prefix'>sin</mo>
  <mo></mo>
  <mi>x</mi>
  <mo></mo>
  <mi>d</mi>
  <mi>x</mi>
</mrow>

```

Forget about the `<mrow>` element, this is added by Tralics using some heuristic rules that do not always work; their effect is to isolate the parentheses from the integral sign: the height of the parentheses should be normal. In the expression, one can see that the integral sign is a character considered as an operator (there no `<int>` element here); it has an exponent and an index. In the same fashion, `\sin` is translated as a `<mo>` element (with an attribute that says it precedes its argument). Nothing in the formula says that the argument is  $x$ .<sup>17</sup> Nothing says that ‘ $dx$ ’ is not the product of  $d$  and  $x$ , but a conventional way of indicating that  $x$  is the bound variable (the `<bvar>` above). In fact, Tralics cannot guess the *use* of the tokens, it knows only the layout: for instance, is  $\Gamma_{ij}^k$  a Christoffel symbol with three indices or a randomly Greek letter, with two indices raised to some power? And what about  ${}^3\text{He}$  or this footnote?<sup>18</sup> Note that such expressions are not part of content markup: in both cases, these things look like tensors and are to be produced with `<mmultiscripts>`. In what follows, we shall speak only of the presentation part of MathML.

Some features are difficult to implement. For instance, it is possible to group some equations in a single mathematical formula, and put a label (with a reference like 17) to the whole, as well as a label for each subequations (referenced as 17.a, 17.b, etc). It is also possible to split an equation on more than one line, with a single number for it:

$$\begin{aligned} \zeta(s) &= 1/1^s + 1/2^s + 1/3^s + 1/4^s \dots \\ &= 1/1^s + 1/3^s + \dots + 1/2^s + 1/4^s + \dots \end{aligned} \quad (1.17)$$

It is possible to add some lines of text between two equations

$$X = 1/2^s + 1/4^s + 1/6^s + 1/8^s + 1/10^s + 1/12^s + 1/14^s + 1/16^s + \dots \quad (1.18)$$

as in this example (equals signs are aligned)

$$= \frac{1}{2^s} (1/1^s + 1/2^s + 1/3^s + \dots) \quad (1.19)$$

What Tralics produces in this case is a single table (with a single equation number), and the intertext is just a new row, left aligned, that spans two columns (see `\intertext` in section 6.13).

<sup>17</sup>Apart from the kludge mentioned above, Tralics make no difference between the letter  $x$  and the parentheses around it; since parentheses are often omitted in the case of the sine function, the argument could as well be a parenthesized  $x$ .

<sup>18</sup>Guess: in the previous formula, was the H in or out of the math formula.



Horizontal spacing in math formulas is managed intelligently by T<sub>E</sub>X. In the case of  $a + b$  or  $a = b$ , there is some space on each side of the operator, and this space disappears when the formula becomes an exponent or an index. Here is an example.


$$\frac{a + b}{a = b}, \quad x_{a=b}^{a+b}.$$

Each operator has a type, and the value of horizontal spacing depends on the type of the tokens on the left and the right. Traditionally, parentheses are removed around the arguments of sine and cosine, and you say: sine squared of X instead of sine of X, the whole squared. Example:

$$\sin^2 x + (\sin x)^2.$$

In T<sub>E</sub>X, you can consider every expression (a simple atom, or a list in braces) as an operator, provided that it is preceded by its type. Types Over, Under, Acc, Rad, Vcent are obtained by construction (overline, underline, adding an accent, constructing a radical, vertical centering). Types Ord, Op, Bin, Rel, Open, Close, Punct, Inner are obtained by defining characters via `\mathchar`, or using the following commands `\mathord`, `\mathop`, `\mathbin`, `\mathrel`, `\mathopen`, `\mathclose`, `\mathpunct`, `\mathinner`. These commands are understood by Tralics, but in general ignored. In the case of an inline expression, a line break can only appear after an operator (like plus or equal), provided it is the main operator (in particular, if you put the whole expression in braces, no line break will occur). This is ignored by Tralics. Hence something like  $f(x^{a+b})$  cannot be broken<sup>19</sup>.

Note that exponents and indices use a smaller font size, and exponents in exponents use an even

smaller one: compare  $x^{y^z}$  with . The image was obtained by converting the math formula into XML, then in a png image (such kind of images will be put into HTML pages, because there are few HTML browsers that understand MathML. We used anti-aliasing, because this is supposed to increase readability). There are three sizes, and four styles (`\displaystyle`, `\textstyle`, `\scriptstyle`, and `\scriptscriptstyle`). The first two styles in the list have the same size (in MathML, there is an attribute `scriptlevel` that controls the size of the expression, and another one, `displaystyle`, that says if the expression is in display style or not). In fact, T<sub>E</sub>X has two substyles, cramped or not. Consider carefully the placement of exponents in the following example

$$\frac{a_2 + x^2}{a_2 + x^2} = \frac{x_3^2}{x_3^2}.$$

One problem (for a translator like Tralics) deals with the placement of arguments versus operators. For instance, if you want to put a dot over a letter, you can say `\dot{x}` (compare the math version  $\dot{x}$  and the text version  $\dot{x}$ ; in text mode, slants are taken into account). The MathML equivalent is an `<msup>` whose first element is the identifier x. If you want to put a prime after a letter you say `x'`, or `x{\prime}`. If you want to put an arrow over x prime, you have the choice between  $\vec{x}'$  and  $\vec{x}'$ . With the fonts used in this example, the first solution looks horrible. You can say `x_2^3` or `x^3_2`: the result is the same: a math item, formed of a nucleus, a superscript and a subscript. If you want something like  $\frac{2}{3}x$ , you have to use two items, the first one has an empty nucleus, the second has no scripts. You can say  $R_i^j{}_{kl}$ , using three items. The MathML translation should consist in a single `<mmultiscripts>` element. It is possible to enclose a formula by braces, brackets, etc, provided that the font contains the machinery needed for it. You can either use `\big`

<sup>19</sup>but Netscape does not hesitate to break after the plus sign, in the case of `<sup>`, observation made in 1999

and its variants, if you know the height of the formula, or `\left` and `\right`, or as in the example that follows, use an environment like `matrix` that uses whatever is best.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

The `\over` command (and friends) are discouraged by `amstex`: you get a message of the form *Package amsmath Warning: Foreign command \over; \frac or \genfrac should be used instead*. In fact the command takes two arguments, one before and one after. Example `x=\left( a+b \over c+d\right)^2+1` gives

$$x = \left( \frac{a+b}{c+d} \right)^2 + 1$$

The trouble is the following: Assume that we have a command `\myfrac` that typesets its arguments as  $a/b$  if the current style is `script` or `scriptscript`, and  $\frac{a}{b}$  otherwise. In an expression as above, the current style cannot be known before the `\over` is seen. For this reason, `TEX` introduced a command `\mathchoice` that takes four arguments, one for each style; after `TEX` has completely read the math expression, it takes, in a second pass, the relevant token list. This is complicated. This mechanism is partially implemented in the current version of `TRalics`.

If you do not like MathML, you can set the integer `\@nomathml` to a non-zero value. If it is positive, then most math commands are allowed outside math mode, with trivial translation. If it is negative, then math formulas are parsed as usual, but the resulting tree is output à la `TEX`, rather than producing a MathML formula. Example

```
\makeatletter
\@nomathml=1
\sqrt{\alpha+\beta ^4}
\@nomathml=-1
$\sqrt{\alpha+\beta ^4}$
```

The translation is

```
\sqrt \alpha +\beta ^4
<texmath type='inline'>\sqrt{\alpha +\beta ^4}</texmath>
```

## 1.9 Some subtleties of `TEX`

Assume that we have a number, say ‘1.3’, and want to convert this to a dimension, say ‘1.3pt’. If the number is in a command `\foo`, we can say `\dimen0=\foo pt`. On the other hand, how can we get 1.3 given that `\dimen0` holds 1.3pt? We may use `\the\dimen0`, and remove the last two characters. Hence we say `\expandafter\rem@pt\the\dimen0`. One trouble is that the category codes of the characters ‘pt’ produced by the `\the` command are 12, not 11, so that a definition like

```
\def\rem@pt#1pt{#1}
```

does not work. You could try to change the category codes of `p` and `t` in the definition, but these letters are part of the name of the command. Some black magic has to be used. The code shown here converts also ‘12.0pt’ to ‘12’.

```
\begingroup
\catcode'P=12
\catcode'T=12
\lowercase{
\def\x{\def\rem@pt##1.##2PT{##1\ifnum##2>\z@.##2\fi}}
\expandafter\endgroup\x
```

Implementing such a construction in a Perl script is not obvious. Consider then the following example:

```
\def\foo#1{#1x#1}\def\xbar#1{ $#1$}
\expandafter\foo\xbar y
\foo\xbar y
```

Consider line 2. After `\expandafter` has read the two tokens `\foo` and `\xbar`, current state is S, a space and the character ‘y’ are not yet read. Expansion of `\xbar` reads as argument the character y; the space before it is ignored. The expansion is then `\foo`  $\sqcup_5$   $\$_3$  y<sub>11</sub>  $\$_3$ . Hence the argument of `\foo` is the dollar sign, expanding it gives  $\$_3$  x<sub>11</sub>  $\$_3$  y<sub>11</sub>  $\$_3$ . Note that this gives an odd number of dollar signs.

Consider now the third line. Here the argument of `\foo` is `\xbar`. The expansion is `\xbar` x<sub>11</sub> `\xbar`, and a space and ‘y’ have to be read. The expansion of `\xbar` is  $\sqcup_5$   $\$_3$  x<sub>11</sub>  $\$_3$ . The second `\xbar` reads a space and ‘y’ as its argument. The expansion is  $\sqcup_5$   $\$_3$  y<sub>11</sub>  $\$_3$ . Note that `latex2html` complains with: *Unknown commands: xbarx*, because the string is rescanned, and a space should have been added. In the second version of our translator, which was a Perl script as `latex2html`, we solved this problem as follows. The commands `\` and `!` are replaced by `\00!` and `\01!`, special characters are replaced by `\3#!`, `\3&!`, `\3<!`, `\3>!`, and `\3~!`. A command like `\foo` is replaced by `\1foo!`, spaces after `\foo` are removed. Using such a mechanism, we can handle spaces correctly, as long as the category codes do not change.

The current version of `Tralics` uses the same representation as T<sub>E</sub>X for its tokens, namely an integer. A character token like  $\$_3$ , is represented by the integer  $c + N * C$ , where  $N$  is the number of characters,  $c$  the character value (here 36), and  $C$  the category code (here 3). In T<sub>E</sub>X, the constant  $N$  is 256, in `Tralics`, it is  $2^{16}$ . Let  $M = 16 * N$ ; then non-character tokens are integers  $x$  at least  $M$ , and  $x - M$  is the address in the table of equivalents. An active character is represented by  $c + N * C$ , with  $C = 16$ , and a single-character command is represented by  $c + N * C$  with  $C = 17$ ; multiletter control sequences are represented by integers larger than  $M + 2N$ , and  $x - M - 2N$  is the hashtable location of the token (it contains the name of the token).

If you say `\uccode‘\~='A` two integers are read; in these case, they are character constants; the value is obtained by taking the token modulo  $N$ . If you say `\uppercase{~}`, for each token in the list, something happens if the value is less than  $M + N$ ; if the value modulo  $N$  is  $c$ , and if the upper case equivalent of  $c$  is  $c'$ , a non-zero value, then  $x' = x - c + c'$  is used instead of  $x$ . Note that if  $C = x/N$  and  $C' = x'/N$ , then  $C = C'$ , this means that an active character is replaced by an active character, a character with category code  $C$  is replaced by a character with the same category code.

The behavior of a token is determined by its command code (and its subcode): if the token is  $c + N * C$ , then  $C$  is the command code, and  $c$  the subcode. Said otherwise, for a character token, the command code is the category code, and the subcode is the character value. In the other case, the token is non-constant, its meaning can change. The actual meaning is in the table of equivalents, it can be pushed on the save stack. For instance, the default value of `\count` is (1,90,0). Here the first integer is the definition level, (1 is the bottom level), 90 is the command code, namely *register*, and 0 is the subcode. The default value of `\pausing` is (1,91,324), where the second number is the command code *assignint* and the third number an address in the table of integers. Remember that, in the case of `\the\count0`, the `\count` command is evaluated for a value; in all other cases it is evaluated for side effects. In particular for a case like `\global\count3=17`. Assume that you say `\count3=17`; in such a case a number is read, namely 3, and checked for out-of-range. After that, the subtype is looked at (the commands `\count`, `\dimen`, `\skip`, or `\muskip` have the same command code, but different subtypes, they read something and store it somewhere). Note: if you say `\countdef\foo3`, then `\foo` has *assignint* as command code, and its subcode is 3, thus behaves exactly like `\pausing`. Moreover the action is the same as `\count`, after 3 has been read. In the same fashion, if you say `\chardef\bar3`, then `\bar` has *chargiven* as command code, and

its subcode is 3; the action is exactly the same as `\char`, after 3 has been read. In the case of *chargiven*, the action consists in putting the character in the dvi file (Tralics puts in the XML tree). In the case of *assignint*, a integer is stored or retrieved. For instance, `\foo=17`, and `\count3=17` read a number 17. This number will be stored somewhere in the eqtb table; the location is 3 slots after the start of the `\count` table. This slot contains (L,V), for instance (4,23). The first number is the definition level, and the second is the value. In the case L=0, this means that the object is undefined (in the case of a counter, this means zero). Otherwise this is the level. In the case `A{B{C}}`, A is at level 1, B at level 2, and C at level 3. The quantity *L* is never greater than the current level. In the case where the assignment is global, (L,V) is instantiated to (1,17). In the case where the current level is L, then V in (L,V) is replaced by 17. Otherwise, the old value is saved on the stack, and (L,V) is replaced by (*l*,17) (where *l* is the current level). When the group is closed, the old value is restored.

Note the following trick. Assume that `\A` increases some counter and puts the value in `\foo`, `\B` does the same, but changes globally `\foo`. Assume that you say `{\A\A\B\B\A\A...}`. The first `\A` sets `\foo` to 1 and saves (0,undef). The second `\A` sets `\foo` to 2. The first `\B` sets `\foo` to 3 at level 1, the second `\B` sets `\foo` to 4, level unchanged. The next `\A` saves the old value of `\foo` and sets `\foo` to 5, etc. As a consequence: every `\A` preceded by a `\B` will put an item on the save stack. When the stack is restored, the value to restore will be (*L*, *V*) and the current value (*L'*, *V'*). If *L'* is 1, nothing happens. Otherwise, (*L'*, *V'*) is replaced by (*L*, *V*). As a consequence the value after the group is the value of the last `\B`. This results in a waste of the save stack. For this reason, Knuth says: all assignments to the scratch registers whose numbers are 1, 3, 5, 7 and 9 should be `\global`; all assignments to the other scratch registers (0, 2, 4, 6, 8, 255) should be non-`\global`.

A silly question is: what happens if you say `{\let \endgraf \par \gdef\par{} \edef \foo {\endgraf} \Foo \def \endgraf {} \Bar }`. In the current version of Tralics, but this is also true for TeX, when you say `\def\foo`, the command code of `\foo` is changed to be *user-defined*, and the subcode is an address into a table containing the token list of the body. On the other hand, `\let \foo \bar` will use the command code and subcode of `\bar`, and copy this in `\foo`. In the original version of Tralics, the Perl version, we had two tables: the list of predefined commands, with their internal number, and the list of user defined commands with their body. In the code above, when `\Foo` is executed, then `\foo` is a user defined command, whose body contains `\endgraf`, whose meaning is the original `\par`. When `\Bar` is seen, the meaning of this token has changed. The essential reason why Tralics was re-written in C++ is to make this piece of code work.

## 1.10 Language options

Most TeX formats (plain, L<sup>A</sup>TeX, amstex) have been written by American people; nowadays, major developments are done in Europe (including the conTeXt format, and the hyperref package). However, lots of people use basic primitives for their French publications, and the situation is not simplified by the fact that there are two packages for writing French documents (one by late Bernard Gaulle, and one by D. Flipo).

In the original version of TeX you had to say ‘`\’e`’ for an e acute, and you had to say ‘`\c c`’ or ‘`\c{c}`’ for a c cedilla (which form being the best is in general unknown). For homogeneity reasons, Lamport recommends ‘`\’e`’. Because some accents are redefined by tables, or tabular environments, the solution that always works is ‘`\a’{e}`’. This is something strange, but a translator like Tralics has to cope with it (for instance, some authors of the Raweb use BibTeX files that are generated automatically from a data-basis, and this software systematically produces `\a` for accents). This makes texts rather uneasy to read, and not every spell-checker understands this (IsPELL for instance allows José or Jos\’e, but not both). By default (i.e. on Linux machine)

L<sup>A</sup>T<sub>E</sub>X understands iso-8859-1. This means that all characters used in France are recognized (except, œ, Æ, Ÿ, these characters cause also problems in HTML). Recently, another character was introduced, namely €. Nobody knows how to use it (according to [6, paragraph 7.8.7], published in april 2004, `\texteuro` is the official L<sup>A</sup>T<sub>E</sub>X way, it is translated by Tralics as `&#20AC;`). In case of doubt, you should use ‘euro’.

In order to emphasize words, you can underline them, use a different font, or mark them with quotes. In English, you would use quotes “like these” or ‘like these’, but never like "this". In France, guillemets « are used like this ». Note that the spacing is different from English, but the package should take care of everything. The forever question is: how to enter these funny characters in my keyboard made in Mexico<sup>20</sup>. One solution consists of typing two < in a row and hope for the best (we have either an active character, or a ligature). Note that Tralics translates `\verb+<<-->>+` as `<hi rend='tt'>&lt;&#x200B;&lt;&#x200B;-&#x200B;-&#x200B;&gt;&#x200B;&gt;&#x200B;</hi>` the funny characters have as only purpose to inhibit ligatures in the resulting XML<sup>21</sup>. The result might also be: `ij` and `ïï`. Depending on the packages, you should perhaps use `\guillemotleft`, `\guillemotright` or `\og`, `\fg`. Which method is the best is still unclear to me.

<sup>20</sup>or Thailand; it has an international layout, with a W. logo which is a registered trademark of M.C.

<sup>21</sup>This character is not generated when using option `nozerowidthspace`



# Chapter 2

## Expansion

One part of the work of T<sub>E</sub>X is to replace all user defined tokens by primitives; this is the main objective of the ‘expansion’ process. In this respect, there is little difference between T<sub>E</sub>X and Tralics. In this chapter, we review some constructions.

### 2.1 Defining new commands

A definition is typically of the form

```
\def\fooi{foo}
\def\fooui#1#2{#2#1}
\def\foouii+#1.#2=#3#{Seen#1#2#3.}
```

You may wonder why the commands are not called ‘\foo1’, ‘\foo2’ and ‘\foo3’. The reason is that, if digits have standard category codes, they are not of type letter, so that ‘\2foo’ is the command \2, followed by the letters ‘foo’ (the tokens are  $\boxed{2}$  f<sub>11</sub> o<sub>11</sub> o<sub>11</sub>) and ‘\foo2’ is the command \foo followed by the digit 2 (the tokens are  $\boxed{foo}$  2<sub>12</sub>). It is possible to create the token  $\boxed{foo2}$  via \csname foo2\endcsname, and it is also possible to change the category code of 2. This is in general a bad idea: If you say \setlength{\parindent}{\foo2+2cm}, it is impossible to design the \setlength command so that ‘\foo2’ is read as a command and ‘2cm’ as a dimension. On the other hand, if you say \def\foo2#1#2{#2#1}, T<sub>E</sub>X expects, after the second #, the character 2 with category code 12; if not it complains with: *Parameters must be numbered consecutively*. In Tralics, the message is a bit different, it says *Error while scanning definition of \foo2 expecting #2; got #{Character 2 of catcode 11}*.) Note how 2<sub>11</sub> is printed.

Before \def, you can put a prefix: it can be \long, indicating that the command accepts whole paragraphs as arguments; it can be \outer, indicating that the command cannot be the argument of another command; it can be \protected, indicating that the command should not be expanded in an \edef (this is an  $\epsilon$ -T<sub>E</sub>X extension); it can be \global. This last prefix can be put before any assignment, it says that the assignment is global (unless \globaldefs is non-zero). More than one prefix can be used, the order is irrelevant. After the \def comes the object to define (this is either an active character, or a command name), then what T<sub>E</sub>X calls (parameter text), and this is followed by the body. The body starts with the first opening brace (any character of category code 1) and ends with the first closing brace (any character with category code 2) that makes the body balanced against braces. These braces are not part of the body. The parameter text is an arbitrary sequence of tokens, but cannot contain braces. If it contains a # (in fact, any character of category code 6), it has to be the final character of the sequence, or be followed by the digits 1, 2, 3, up to 9, in order. If there is some text between #3 and #4 (or between #3 and the start of the body), this imposes a constraint on the third argument. If there is some text before #1, this

imposes a constraint on the command itself. In the body you can use `##`, this will be replaced by a `#`; you can also use `#1`, `#2`, etc., this will be replaced by the value of the first, second, etc., argument. As above, the `#` is any character of category 6, the digits are of category 12, you cannot access the second argument if only one is available. If you define `\foo2` as above,  $\TeX$  will signal a second error: *Illegal parameter number in definition of \foo2*.

Once you have defined the commands, you can use them. We give here an example, and the translation by `Tralics`

```
\fooi\fooi12\fooi+ok. {\itshape 3} =xyz{!}
foo21Seenok <hi rend='it'>3</hi> xyz.!
```

and also by  $\LaTeX$  `'foo21Seenok 3xyz.!'` Some explanations. The first command takes no argument, thus is easy to understand. The second command takes two arguments, its body is `'#2#1'` so that the expansion is the token list formed by the tokens of the second argument followed by the tokens of the first argument. In the case of `'\foo12'`, the arguments are `'1'` and `'2'` (a list of length one). In the case of `'\fooi {AB} {CD}'`, the arguments are `'AB'` and `'CD'`, a list of length two. This is because  $\TeX$  ignores initial spaces when reading undelimited arguments; in any case, an argument is well-balanced against braces (same definition as above for the body of a command). The shortest possible sequence of tokens is read (in the case of an undelimited argument, this sequence is never empty). If the result starts with an open brace and ends with a closing braces, these are removed, provided that the remaining token list is well-balanced; for instance, in the case `'\fooi{a}'`, the first argument is empty. If the command is not `\long`, then `\par` tokens are forbidden in the argument. In any case tokens that are defined to be `\outer` are forbidden in a parameter.

In the case of `\fooi+ok`, the situation is a bit more complicated. Fetching the argument is more involved than in the general case. The specification is: plus sign, argument, dot, argument, equals sign, argument, sharp sign. Note first that the `'+'` sign is not part of the command name, but is required after it whenever used. The first argument here is the shortest sequence (possibly empty) of tokens, that is a balanced list, and this is followed by the required token list (here, a single dot). Here it is `'_{\it_3}_'`; a pair of initial and final braces disappear, if possible. The `'#3'` says that the third argument is delimited by an open brace. This brace is left unread. Such a construction is rare: it occurs only four times in the  $\LaTeX$  sources, two example will be given later in section 2.10.

Consider the following example: `'\def\opt[#1]{}'`. If you say `'\opt[foo]'` or `'\opt[{foo}]'`, the argument is `'foo'`. If you say `'\opt[{}foo]'`, it is `'[foo]'`. It is important to know that braces are required if you want a closing bracket in the argument. In the case of `'\item[{\it foo}]'`, the braces are useless; the scope of the `\it` command is limited to `'foo'` because an additional pair of braces is added somewhere in the body of the `\item` command. The following example is non-trivial:

```
\def\@car#1#2\@nil{#1}
\def\@cdr#1#2\@nil{#2}
\if b\expandafter\@car\f@series\@nil\boldmath\fi
```

Both commands `\@car` and `\@cdr` read a normal (undelimited) argument, and a second argument delimited by `\@nil`, and return one of these. These commands are implemented in `Tralics` in the C++ kernel for efficiency. The third line shows a use of `\@car`, where the arguments are the expansion of `\f@series`; the main assumption is that this token list does not contain the `\@nil` token, which is a reserved command. The caller of the macro must also ensure that the list is not empty, for otherwise the first argument would be `\@nil`, and the end of the second argument would never be seen if the `\@nil` does not appear in the document text. Note that an error is signaled and scanning stops at the first `\par` token (or empty line) because the command is not outer.



Let's assume that `\f@series` expands to a non-empty list, for instance 'mc' (this means that the current font has medium weight and is condensed). Then '`\@car md\@nil`' expands to 'm'. The third line of our example uses `\@car` to get the first character of `\f@series`, and compares it to 'b' (the result is true if the current font is bold, extra bold, bold condensed, etc). This code is used for typesetting the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> logo in bold version as **L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>**. The commands `\if` and `\expandafter` will be explained later. Note that `\if` fully expands what follows the letter b. This means that you are in trouble if `\f@series` expands to an empty list, or if the first token is a command whose expansion may cause problem (perhaps because it has delimited arguments and `\@car` gobbled the delimiter), or is empty, or is a list that starts with the letter b.

The following example is from the T<sub>E</sub>Xbook:

```
\def\cs AB#1#2C$#3\$ {#3ab#1 c##\x #2}
\cs AB{\Look}C${And\$ }{look}\$ 5
```

If you feed this to Tralics<sup>1</sup>, you will get three errors (one because of the '##', and two undefined commands). In verbose mode, the transcript file of Tralics will contain the following

```
\cs AB#1#2C$#3\$ ->#3ab#1 c##\x #2
#1<-\Look
#2<-
#3<--{And\$ }{look}
```

One question is: should arguments be in braces or not? As seen elsewhere, some commands have a special syntax, and cannot be followed by braces (for instance, in the case of '`\catcode'\$'` the argument is the backtick followed by the dollar). In a case like `\$a \over b+c\$`, there are two arguments, one before and one after the command. An expression like `\$a\over b\over c\$` is a error. The error message says to add some braces, but they are used only for grouping. A similar error message is issued if you say `\$a^b^c\$`. But compare '`\$a^{b^c}d\$`' and '`\$a\over {b\over c}d\$`': the translation is  $a^b d$  and  $\frac{a}{\frac{b}{c}d}$ . In the case of `\sqrt \frac{1}{2}`, braces are inserted by T<sub>E</sub>X when converting `\frac` into `\over`; since Tralics replaces `\over` by `\frac`, no such braces are added and an error is signaled because of missing braces.

It is sometimes important to know which braces disappear or remain. As an example, you can say '`\def\ap{a}'` in order to get  $a'$ ; but if you say this '`\$x_\ap\not=x_{\ap}\$`', you get  $x'_a \neq x_a'$ . In fact, you cannot say that '`\ap`' is the argument of the underscore command; this is because T<sub>E</sub>X expands everything; in one case, it sees that the underscore is followed by the letter a, in the second case by a brace, hence a delimiter for a math list.

In general, you will be faced with the following problem: you say '`\def\foo#1{\xbar#1}'` and '`\def\xbar#1{\itshape #1}'`. Note the double braces: the outer braces delimit the argument (of `\def`, i.e., the body of `\xbar`), the inner braces delimit the scope of `\itshape`. When you say '`\foo{12}`' only the first letter is in italics, another level of braces is needed. This is what you can see in the transcript file of Tralics:

```
\foo #1->\xbar #1
#1<-12
\xbar #1->{\itshape #1}
#1<-1
{begin-group character {
+stack: level + 3 for brace
{\itshape}
{font change \itshape}
...
```

In this example, braces are missing in `\foo`, a remedy is to add a pair of braces in the argument, like '`\foo{{12}}`'. A comment in the T<sub>E</sub>X source says: Braces are effectively removed when

<sup>1</sup>The last character on the line is not read

they surround a single Ord without sub/superscripts, or when they surround an accent that is the nucleus of an Ord atom. This is the case in ‘`{\tilde x}^2^3`’, hence you get a *Double superscript* error; in this case adding additional braces has no effect; the only solution consists in adding something in the inner list (for instance a kern of width zero).

It is possible to define commands inside commands. For instance, you can say

```
\def\foov#1{\def\xbar##1{#1##1###1###1}}
```

When the scanner reads a token list, it handles ‘#’ signs (in fact, any character of category 6) in a special manner inside a definition. The token list of the previous line is `def foov #23 {1 def xbar #6 112 {1 125 #6 112 #6 125 #6 #6 112 }2 }2`. As you can see, there are three possibilities for a sharp sign: before the brace that defines the body, it is `#23`, and the digit that follows is omitted<sup>2</sup>, it is `125`, `225`, in the body when followed by 1, 2, etc<sup>3</sup>. It is `#6` when followed by a sharp sign. Said otherwise, a double sharp sign in a definition is equivalent to a normal one outside. Note the following trick.

```
\catcode'\^6
\def\foo#1^2{#1^1## #^ # ^}
\show\foo
```

A quantity like `125` is shown as `^1`, because the hat character appears as `^2` (i.e., the token `^25`) in the (parameter text) part of the definition. Hence T<sub>E</sub>X prints `\foo=macro: #1^2->^1^1## ^^ ## ^`. On the other hand, T<sub>r</sub>alics uses a different mechanism for macros: it remembers the number of arguments and the items between them, hence does not make the difference with a macro defined as ‘`\def\xbar^1#2{...}`’<sup>4</sup>. `\foo=macro: #1#2->#1#1## ^^ ## ^^`.

Assume now that you say ‘`\foov{17}`’. The result of the expansion is the token list shown above, with `125` replaced by `112 712`. When `\xbar` is defined the `#6` will read the character that follows, in this case `112`. The situation is as if you had said ‘`\def\xbar#1{17#1#17##1}`’. Evaluating `\xbar` may signal an error, because of the ‘##’ (no error is signaled in case the argument of `\xbar` is ‘`\gee`’, a command that ignores its first and third argument). If you call `\foo` with ‘25’ instead of ‘17’ as argument, you will get the following error *Illegal parameter number in definition of \xbar*<sup>5</sup>.

## 2.2 Defining commands in L<sup>A</sup>T<sub>E</sub>X

You can say

```
\newcommand*\fooi}{foo}
\newcommand*\fooui}[2]{#2#1}
\newcommand\foov[3][bar]{Seen#1#2#3}
```

The first two lines define the same commands as in the start of section 2.1. It is not possible to define `\foouii`. However, you can define `\foov`, a command that takes an optional argument. In fact, you call it like this ‘`\foov[X]YZ`’; the expansion will be ‘`SeenXYZ`’. You can put a pair of braces around the arguments, like ‘`\foov[{X}]{Y}{Z}`’, the result is the same. Braces are needed for the first argument in case you want a closing bracket in it. If the first argument is ‘`bar`’, you can omit the ‘`[bar]`’: for this reason, the argument is called optional. In L<sup>A</sup>T<sub>E</sub>X, `\foov` expands to `\@protected@testopt`, which is a command to make `\foov` robust (i.e., in some cases, the test for an optional argument is delayed); it then expands to `\foov`, which is a command that

<sup>2</sup>T<sub>E</sub>X uses `#13`, because no confusion with an active character can occur.

<sup>3</sup>T<sub>E</sub>X uses `15`, because no confusion with a character of category 5 can occur (See T<sub>E</sub>Xbook, exercise 7.3).

<sup>4</sup>Said otherwise `\ifx\foo\xbar` produces different results in T<sub>E</sub>X and in T<sub>r</sub>alics; these two commands take two arguments, and have the same body, the only difference is in the introducing character.

<sup>5</sup>This was accepted by T<sub>r</sub>alics until version 2.3: if a command takes one argument, the value of the second argument is the empty list.

takes three arguments. In Tralics, no auxiliary command is used. If you say ‘\show\fooiv’, Tralics will print the following on the transcript file.

```
\fooiv=opt \long macro: bar#2#3->Seen#1#2#3.
```

Commands defined by \newcommand are \long unless a star is used (they accept paragraphs as arguments.) The ‘opt’ before it shows that the command takes an optional argument. We show the value of this argument instead of #1 before the ->. The following is printed by L<sup>A</sup>T<sub>E</sub>X

```
> \fooiv=macro:
->\@protected@testopt \fooiv \fooiv {bar}.
```

Since being \long deals with reading parameters, in L<sup>A</sup>T<sub>E</sub>X, it is the auxiliary command \fooiv which is \long. This shows how to ask L<sup>A</sup>T<sub>E</sub>X for the meaning of the auxiliary command and its answer:

```
\expandafter\show\csname\string\fooiv\endcsname
> \fooiv=\long macro:
[#1]#2#3->Seen#1#2#3.
```

The philosophy of L<sup>A</sup>T<sub>E</sub>X is that a user should not randomly redefine commands. For this reason, you must use \newcommand (for an undefined command) or \renewcommand (for overwriting an existing command). In the same fashion, \renewenvironment is used to redefine an environment; we shall see later that an environment ‘foo’ is defined by two commands: \foo and \endfoo. You should never define \endfoo. This explains error messages of the form: *LaTeX Error: Command \endfoobar already defined. Or name \end... illegal, see p.192 of the manual.* In Tralics, we do not check that the command starts with ‘end’; the error message is *\newcommand: cannot define \foo; token is already defined.* You can use \providecommand, the syntax is the same. In this case, the definition is silently ignored if the command already exists. You can use \DeclareRobustCommand, this is defined by Tralics to be the same as \providecommand although the L<sup>A</sup>T<sub>E</sub>X behavior is different. You can say ‘\global\def\foo{’}, this is the same as ‘\gdef\foo{’}, it defines \foo globally. You cannot use the \global prefix for L<sup>A</sup>T<sub>E</sub>X commands. You can use \CheckCommand. This is like \newcommand, but it does not define the command; instead it defines a dummy command, then checks that the dummy command has the same definition as the real one and produces a warning in case of mismatch; this feature can be used before overwriting a command.

It is now time to explain that braces have two different purposes: as a delimiter for an argument list, and also for grouping: in the same fashion as the formula  $z(x + y)$  can be considered as  $z$  applied to  $x + y$  or the product of  $z$  and  $x + y$ . In the case of ‘\textit{12}’, the braces delimit the arguments, in the case of ‘{\itshape 12}’, the braces are used for grouping. In both cases, all characters up to the closing brace are in italics, but this property depends on the semantics of the operator, not the syntax. There is a big difference between these two use of braces: the tokenizer produces token lists that are always balanced (there are as many opening delimiters as closing delimiters, where delimiters are characters of category code 1 and 2). On the other hand, if you say ‘\let\bgroup={’, the \bgroup has the same meaning as an opening brace, hence triggers the start of a new group; but it is not an explicit character (such things are called “implicit characters” in the T<sub>E</sub>Xbook). When you say ‘\hbox...’ the opening brace can be implicit or explicit (in this case, braces are used both as delimiters and for grouping). Groups can also be defined by math shift characters (if you like empty lines in the source of a math formula, you can say ‘ $\let\par\relax \dots$ ’), or implicitly for a cell in a table, or via \left and \right in a math formula, or via \begingroup and \endgroup (they define a “semi simple group”).

One difference between plain T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X is the existence of named groups: instead of saying ‘\beginfoo’ and ‘\endfoo’, you say ‘\begin{foo}’ and ‘\end{foo}’. This is interpreted by L<sup>A</sup>T<sub>E</sub>X as

1. When \begin{foo} is seen,

- (a) a test is made to see if ‘foo’ exists: if it does not exist, an error is signaled and steps (1c) and (1d) are skipped (via a call to `\expandafter`):
  - (b) the command `\begin{group}` is executed (with space hacking);
  - (c) the name ‘foo’ is stored in `\@currentenv`;
  - (d) the command `\foo` is executed.
2. When `\end{foo}` is seen,
- (a) the command `\endfoo` is executed;
  - (b) the name ‘foo’ is compared with `\@currentenv`, an error is signaled in case of mismatch;
  - (c) the command `\endgroup` is executed (with more space hacking).

This mechanism is not symmetric. It is implemented in `Tralics` in a similar manner (but there are some differences that can be analyzed by a malicious user). The first remark is the following: on entry, you may get a message that says *LaTeX Error: Environment unknown undefined*, on exit you would get *LaTeX Error: \begin{document} ended by \end{unknown}*. The trick is that the `\endfoo` token (created by `\csname`) is never undefined (its default action is `\relax`). One important point is that the command used in step (1d) is `\foo`, not `\beginfoo`. In [6, example 7-3-1], there is an example of ‘bfseries’ as an environment; there is no command `\endbfseries`. Note that in step (1d), the token that comes after `\foo` is the token after ‘`\begin{foo}`’, and this means that `\foo` can grab its arguments; on the other hand the token after `\endfoo` in step (2a) is the start of the sequence that checks the environment name: thus `\endfoo` cannot read its argument (we shall see in a minute why steps (2a) and (2b) cannot be swapped). In the current version of `Tralics`, the “space hacking” is not implemented; we shall not discuss it here.

There are some tokens whose name start with ‘end’, you should no use these as environments. Consider `\begin{group}` and `\endgroup`, the commands explained above; consider `\input`, `\endinput`, these are `TeX` primitives for inputting from a file; consider `\beginL`, `\endL`, `\beginR`, `\endR`, the  $\epsilon$ -`TeX` extensions for left-to-right or right-to-left writing; consider `\citation` and `\endcitation`, these are `Tralics` commands for the bibliography; the command `\endsec` indicates the end of a section; the `\endlinechar` is a reference to an integer register that contains the character to be added at the end of each line. Commands `\endgraf` and `\endline` are aliases to `\par` and `\cr`.

This is how you can define new environments:

```
\newenvironment{x}[2]{#1BY\begin{y}#2AY} {by\end{y}ay}
\newenvironment{y}{Z}{z}
\begin{x}a b c \end{x}
```

This typesets as `aBYzbAY c byzay`. The `\begin` part reads two arguments. The `\end` part takes no argument; it could use the first argument of the `\begin`, provided that this one saves it in a command. In verbose mode, the following is printed by `Tralics` in the transcript file. We have removed all lines with ‘Character sequence’ and ‘Text’.

```
1 [185] \begin{x}a b c \end{x}
2 {\begin}
3 {\begin x}
4 +stack: level + 3 for environment entered on line 185
5 \x #1#2->#1BY\begin {y}#2AY
6 #1<-a
7 #2<-b
8 {\begin}
9 {\begin y}
```

```

10 +stack: level + 4 for environment entered on line 185
11 \y ->Z
12 {\end}
13 {\end x}
14 \endx ->by\end {y}ay
15 {\end}
16 {\end y}
17 \endy ->z
18 {\endgroup (for env)}
19 +stack: ending environment y; resuming x.
20 +stack: level - 4 for environment from line 185
21 {\endgroup (for env)}
22 +stack: ending environment x; resuming document.
23 +stack: level - 3 for environment from line 185

```

At lines 4, 10, 20 and 23, you can see that the current “level” changes (this is what T<sub>E</sub>X calls the “semantic level”). The default level is level one, our example was done at level two, the first environment is at level three, the second at level four<sup>6</sup>. When you see ‘level + 4’, it is because the level has just incremented; if you see ‘level - 4’ it means that the level will decrease. At lines 18 and 21, you see that Tralics uses a special ‘\endgroup’ token. Look closely at lines 13 and 19: when Tralics sees ‘\end{x}’, the current environment is ‘y’, it is only after evaluation of \endx that the environment is ‘x’ again; this example shows that steps (2a) and (2b) cannot be swapped. In Tralics the name of the environment cannot be modified by the user.

Because of the \begingroup command, everything, until the \endgroup, is local to this group; in particular \currentenvir will be restored. If you say something like

```
\begin{zfoo}\renewcommand\endzfoo{A}\end{zfoo}
```

the command associated to \end{zfoo} is locally redefined. In some cases, this is a big mistake: in Tralics, the start command can assume that the corresponding end command is executed or an error is signaled. In fact, the meaning of \endzfoo is stored on a special stack, and restored by \end{zfoo}. There is a big hack in L<sup>A</sup>T<sub>E</sub>X (and also in Tralics): since no text should follow the end of the document, there is no need to store on the stack every definition given between the start and end of the document; thus \document executes a \endgroup; logically, \enddocument should insert a \begingroup token; in L<sup>A</sup>T<sub>E</sub>X, this is not needed because step (2c) is never executed. In Tralics we re-insert a \begin, because we have to typeset the bibliography. (as a consequence, the start-line in the trace is the line that contains \end). Moreover, action cannot be completely trivial, because we have to re-insert all tokens saved by \AtEndDocument. We show here the transcript file, assuming that only one token has been saved, namely \empty. You can see the stack increase and decrease; you can see the \endinput that closes the current file; you can also see a second \enddocument command whose action is to pop the XML stack; it is marked ‘pop (module)’ for historical reasons.

```

[31] \end{document}
{\end}
{\end document}
+stack: level + 2 for environment entered on line 31
{\enddocument}
{\endallinput}
\empty ->
{\enddocument}
{Pop (module) 2: document_v div0_v div1_v}
{\endgroup (for env)}

```

<sup>6</sup>The command \currentgrouplevel defined by  $\epsilon$ -T<sub>E</sub>X returns one less than this value.

```
+stack: ending environment document; resuming document.
+stack: level - 2 for environment from line 31
++ Input stack empty at end of file
```

The last line of the transcript file shown above says that the current file was not inputted by another one. What happens if a file `foo.tex` contains `\input tralics-rr`, followed by some junk? Well, the purpose of the pseudo command `\endallinput` is to forget about everything. The transcript file would contain

```
++ End of file tralics-rr.tex
++ cur_file_pos restored to 0
++ Input stack -- 1 tralics-rr.tex
++ Input stack empty at end of file
```

Clearly, you cannot use a document environment in a document; if you try,  $\LaTeX$  complains with *LaTeX Error: Can be used only in preamble* (the preamble is everything before `\begin{document}`). The error message of `Tralics` is a bit more explicit: *Two environments named document*. If you put `\begin{it}` before `\begin{document}`,  $\LaTeX$  does not complain. The trouble is at the end: you will get an error of the form *LaTeX Error: \begin{it} on input line 9 ended by \end{document}*, followed by a  $\TeX$  warning: *(\end occurred inside a group at level 1)*. In `Tralics`, an error is signaled at the start: *\begin{document} not at level 0*. On page 220, you see statistics of the form ‘Save stack +1582 -1582’; this means that the semantic stack pointer has increased 1582 times, it has decreased the same number of times, so that the end of the document has been seen at level zero, no warning is issued in the case the two numbers are not the same.

The package `checkend` contains a magic command whose effect is to unwind the stack, signaling an error if unclosed items are seen. This command should only be used at end of document, in the end-document hook. The result of using the package produces a result like the following:

```
Error signaled at line 687 of file testkeyval.tex:
Non-closed \begingroup started at line 683.
Non-closed brace started at line 437.
Non-closed environment ‘it’ started at line 213.
```

## 2.3 Some small examples

Remember that `\foo` and `\;` are two commands who differ only in the following behavior: when the tokenizer sees a backslash followed by a semi colon (whose category code is not letter), it constructs a command whose name is formed by that character (and sets the internal state to a mode in which spaces are not ignored). On the other hand, if the backslash is followed by a letter, all letters are read (and the state is set such that following spaces will be ignored). By space, we mean here every character that has the category code of a space. A space after `\verb` is never ignored, but it is unwise to use this space as delimiter. In the case of `\foo`, the tokenizer allocates a slot on the hash table (unless `\foo` already exists). The possibility to change category codes dynamically is interesting (however, the implementation of `\verb` in `Tralics` uses no category code changes, and is more efficient). The two commands `\makeatletter` and `\makeatother` change the category codes of the at sign character `@`, to letter and other. For instance

```
\makeatletter
\def\foo@val{}
\def\foo#1{\def\foo@val{#1}\check@foo}
\def\usefoo{\foo@val}
\makeatother
```

In this example, we have two user commands: `\foo` that defines a variable, and `\usefoo` that uses it. The variable `\foo@val` has a reserved name; there is a command `\check@foo` that makes sure

that the argument is correct. The default category code of @ is 12; in most of the examples, we shall assume that it is 11, because these examples come from the L<sup>A</sup>T<sub>E</sub>X kernel or style files where the default category code is 11.

As explained above, ‘`\catcode‘\$=3’`’ changes the category code of the dollar sign. What follows has to be a character code (a number between 0 and 255) followed by an optional equals sign, followed by a valid category code (an integer between 0 and 15). Assume that you say `\def\A{25}`, followed by ‘`\catcode\A7.`’. In the case where standard category codes are in effect this is tokenised as `\catcode \A 712 .12`. But when a number is read, all tokens are expanded, until the end of the number is found (in the case where the number is formed by digits, one space character after the number will be read, if possible). In this case, T<sub>E</sub>X reads the digits 2, 5 and 7. It stops reading at the dot. This is an error (signaled by Tralics as *Bad character code replaced by 0: 257*). Then T<sub>E</sub>X reads an optional equals sign (there is none) and an integer (there is none). Hence you get a second error (*Missing number, treated as zero*). The result is that you have changed the category code of the null character to zero (like backslash). Since version 2.9, Tralics accepts 16bit characters, so that the number 257 is valid, and you changed the category code of the letter ‘latin small letter a with caron’ to zero.

If you want to put 7 in the category code of the character defined by the command `\A`, you should say ‘`\catcode\A=7~`’.<sup>7</sup> It is possible to make `\A` a reference to the character number 25, by using `\chardef`. Thus you can say ‘`\chardef\A25~`’ and ‘`\catcode\A7~`’. Note that, in the context of routines like `scanint`, a character number is a valid number; so that `\A` can be used as the number 25, wherever a number is required. In the sources of L<sup>A</sup>T<sub>E</sub>X you can see ‘`\chardef\active=13`’. You will also see ‘`\mathchardef\ccclvi=256`’; there is no difference between `\chardef` and `\mathchardef`, except that a character is in the range 0-255, while a math char can take larger values (less than 2<sup>15</sup>). You can use `\countdef\B26` (this will make `\B` as a reference to count register number 26), `\dimendef\C27` (this will make `\C` as a reference to dimension register number 27), `\skipdef\D28` (this will make `\D` as a reference to skip register number 28), `\muskipdef\E29` (this will make `\E` as a reference to muskip register number 29), and `\toksdef\F30` (this will make `\F` as a reference to token register number 30). There is no ‘`\boxdef`’. The reason is that, if you want to copy the value of counter 1 into counter 0, you say `\count0=\count1`. If you say `\count@=\B` this will put the value of the counter 26 into `\count@` (this is the counter 255). However, you say `\setbox0=\copy1` if you want to copy the content of box 1 into box 0: the syntax is not the same. Note that `\setbox0=\box1` copies and clears the box number one. When you use a command like `\chardef`, a line will be added to the transcript file, even in non-verbose mode, see section 6.13.

Commands can be defined via ‘`\let`’. You say `\let\A=\B`, where `\A` is a token that can be defined (active characters or commands; T<sub>E</sub>X does not care if the token is defined or not). It is followed by `<equals><one optional space>`. This means that T<sub>E</sub>X reads all space tokens; if the first unread token is an equals sign, it is read as well as the next token, provided that it is a space. If the equals sign is followed by two space tokens, only one is read. Instead of `\B`, you can put any token. After that, the current meaning of `\A` will be the current meaning of `\B`. For instance, if you say `\let\foo\bar\show\foo` you will get `\foo=macro:->\mathaccent "7016\relax`. In plain T<sub>E</sub>X, you would see a space instead of `\relax` (both a space and a `\relax` indicate the end of the number). In Tralics, you would see `\foo=\bar`, this is because `\bar` is a primitive, instead of a user defined command. If you say `\let\A=+`, then `\A` will behave like a + character (of category 12). In fact, this is called an implicit character, and sometimes an explicit character is required. For instance in the case `\parindent=-3.4pt`, the minus sign, the digits, the dot, and the two letters `pt` must be explicit characters. However, after

```
\let\bgroup={ \let\egroup=} \let\sp^ \let\sb=_
```

<sup>7</sup>Here, and in the lines that follow, you should replace the tilde by anything that stops scanning the number.

there is no difference between  $x\sp\bgroup a\sb b\egroup$  and  $x^{\{a_b\}}$ . The assignments shown here are made by `Tralics` when bootstrapping, and the command so defined should be considered primitives. A token list has to be well balanced against explicit braces. For instance

```
\def\foo{\catcode'\=0\egroup}
```

satisfies the requirements. The body of the command consists in  $\{_1 \boxed{\text{catcode}} \text{' }_2 =_{12} 0_{12} \boxed{\text{egroup}} \}$ . If you evaluate `\foo`, the `\catcode` command will read the four tokens that follow; it will modify the category code of the opening brace. All this happens inside a group opened by  $\{_1$  and closed by  $\boxed{\text{egroup}}$ , so that this is harmless. One use of `\let` is the following:

```
\def\fooA{a very long command}
\def\fooB{another very long command}
\def\xbar#1{\ifx 0#1\let\foo\fooA \else \let\foo\fooB\fi}
```

Here we use the fact that `\let` just moves a pointer.<sup>8</sup> This is faster than copying a list. In particular, consider

```
\def\xbar#1{\ifx 0#1\fooA \else \fooB\fi}
\def\xbar#1{\ifx 0#1\let\foo\fooA \else \let\foo\fooB\fi\foo}
```

The first line executes conditionally one of `\fooA` and `\fooB`. However, this command cannot read an argument (because `\fooA` is followed by `\else` and `\fooB` by `\fi`). In the second case, we define `\foo` conditionally, and it can read its arguments without problem.

You can use the following construct

```
\def\addtofoo#1{\let\oldfoo\foo\def\foo{#1\oldfoo}}
% example of use
\def\foo{A}\foo
\addtofoo{B}\foo
```

This typesets as `ABA`. Beware: the `\addtofoo` command can be used only once (the old value of `\oldfoo` has to be saved...). We shall see later how to replace in the definition above the `\oldfoo` by its value, using either tokens lists or `\edef`, using a method where `\oldfoo` is a temporary. This is another example:

```
\def\double#1#2{\let#1#2\def#2{#1#1}}
% example
\def\B{C}\def\C{to}\double\tmp\B
```

Here `\B` typesets as `'toto'`. In fact `\B` is defined as `'\tmp\tmp'`, where `\tmp` is the old definition of `\B`, namely a command that expands to `\C`. If you say `\def\C{ti}\B`, you will get `'titi'`. If in `\double` the `\let` is replaced by a `\def` as `\def#1{#2}`, the expansion of `\tmp` would have been `\B`, and `\B` would have been the same as `\B\B`. You see the problem? This could provoke a stack overflow, a parameter stack overflow, or even a program crash.

Let's mention the existence of `\futurelet\A\B\C`. It is the same as `\let\A\C\B\C`. The usefulness of such a construct will be explained later.

You can say `\expandafter\A\B`. In such a case, `TEX` reads the first token, saves it somewhere, calls expand if possible, re-inserts the saved token. Nothing special happens if the second token (here `\B`) cannot be expanded, because it is a non-active character, or a command like `\par` or `\relax`. But assume that `\A` is a command that uses one argument (for instance `\textit`) and `\B` expands to `'foo'`. If you use `\expandafter`, only the first letter will be in italics. Assume that `\foo` expands to a dollar sign. Then `\foo` is an empty math formula because `\foo` is not expanded, but `\expandafter\foo.\foo` is a display math formula with a dot. The main reason why tokens are not expanded after a dollar sign (when `TEX` looks for an other dollar sign) is that a test `\ifmode true\fi` should evaluate to true. You can use `\expandafter` if you want the test to be executed outside math mode. Note: if a table contains a template of the form `'##$'`, if the cell starts with

<sup>8</sup>The reference count of the token list of the body of `\fooA` or `\fooB` must also be increased.



`\ifmmode`, then the test is expanded (i.e. evaluated) before math mode is entered, because T<sub>E</sub>X is looking for an `\omit` token. As a consequence you should always put `'\relax'` before a test (this is not needed if a command is made “Robust”).

Look carefully at the following lines:

```

1 \def\toto{\titi!}\def\titi{\tata}\def\tata{\tutu}
2 \expandafter\expandafter\expandafter\def\toto{5}
3 \let\E\expandafter \E\E\E\def\toto{6}
4 \def\E{\expandafter} \E\E\E\def\toto{7}
5 \expandafter\def\toto{8}

```

On the first line we define three commands `\toto`, `\titi` and `\tata`. As we shall see, lines 2, 3 and 4 do not change the meaning of `\toto`, so that, on line 5, the expansion of `'\toto'` is `'\titi!'`. In this case, the effect of the `\expandafter` is to replace `'\toto'` by `'\titi!'`. Hence, line 5 defines a macro `\titi`, that has to be followed by an exclamation point, takes no argument, and expands to 8. Consider now line 2. The first `\expandafter` puts apart the `\expandafter` token; it expands the next token, which is `\expandafter`, and the expansion of this is: read the token that follows (here `'\def'`), and expand the token that follows. This is `'\toto'`, that expands to `'\titi!'`. If we pop back the two tokens, line 2 is equivalent to `'\expandafter\def\titi!{5}'`. This looks like line 5, so that it is the same as `'\def\tata!{5}'`. There is no difference between lines 2 and 3: the `\E` command behaves exactly like `\expandafter`. Consider now line 4. What T<sub>E</sub>X does is expand the first token. It is `\E`, it expands to `'\expandafter'`. Since the token can be expanded, it will. Thus T<sub>E</sub>X reads and remembers the token that follows. It expands the next token (the third `'\E'`). Its expansion is `'\expandafter'`. Hence, line 4 is equivalent to `'\E\expandafter\def\toto{7}'`. Now, the `\E` in this list has as effect to try to expand the second token; it is `\def`, which cannot be expanded. Hence this `'\E'` is useless. Line 4 is equivalent to `'\expandafter\def\toto{7}'`. And this defines `\titi`. We give here the trace of Tralics (it is a bit more complete than the trace of T<sub>E</sub>X):

```

\E ->\expandafter
{\expandafter \E \E}
\E ->\expandafter
\E ->\expandafter
{\expandafter \expandafter \def}
{\expandafter \def \toto}
\toto ->\titi !
{\def}
{\def \titi !->7}

```

A question is : how many commands with two characters can be defined in Tralics? The answer is 255 squared (all characters but the null character are allowed<sup>9</sup>). Of course, if you say `'\def\++{...}'`, this defines the `'\+'` command not the `'\++'`. You could imagine to change category codes (but, in a construction like `'\def\{}{...}'`, it is impossible to give a different role to the first and second opening brace). The solution is given by `\csname`, you can use it like this `'\csname1+1=2\endcsname'`. Note that this typesets nothing: when `\csname` manufactures a new control sequence name, it defines it as being `\relax` (the control sequence will exist, until the end of the job). You can hide the `\csname` command, like this

```

\def\nameuse#1{\csname #1\endcsname}
\nameuse{1+1=2}

```

If you want to define such a beast, you must use `\expandafter`.

```

\def\namedef#1{\expandafter\def\csname #1\endcsname}
\namedef{1+1=2}{true}

```

<sup>9</sup>Since Tralics2.9 uses 16bit integers for characters, this number is much larger, nearly 2<sup>32</sup>; it is much larger than the size of the hash table, so the real answer is the number of free slots in the hash table plus the number of already defined two character tokens

The two commands `\@namedef` and `\@nameuse` are defined by L<sup>A</sup>T<sub>E</sub>X and Tralics like `\namedef` and `\nameuse`.

You can also say `\namedef{++}#1{#1+#1}` followed by `\nameuse{++}{3}`. This should give 3+3. If you want a macro named `\{}`, you can say `\nameuse{\string{\string}}`, provided that `\escapechar=-1`. If you do not like this setting of `\escapechar`, you can define a command, say `\Lbra`, that expands to `{`<sub>12</sub> (an inactive opening brace character) using whatever method seems best. For instance

```
{\escapechar=-1 \xdef\Lbra{\string\{\}\xdef\Rbra{\string\}}
\namedef{\Lbra\Rbra}{Hey}
```

We explained above what happens when three `\expandafter` come in a row. Thus, it should not surprise you that the following command defines `\foo`.

```
\expandafter\expandafter\expandafter\def\nameuse{foo}{12}
```

A more realistic example of `\csname` is

```
\def\allocate#1{...}
\def\newcount#1{\allocate{ctr}\countdef#1\allocationnumber}
\def\newcounter#1{\expandafter\newcount\csname c@#1\endcsname}
```

There are ten such commands in L<sup>A</sup>T<sub>E</sub>X, `\newcount`, `\newtoks`, `\newbox`, `\newdimen`, `\newskip`, `\newmuskip`, `\newread`, `\newwrite`, `\newlanguage` are implemented in Tralics. The equivalent of `\allocate` takes as argument a type (for counters, dimensions, skip registers, muskip registers, box registers, token registers, input registers, output register, math families, language codes, insertions, etc) and allocates a unique number depending on the type, and puts it in `\allocationnumber`. Count registers between 10 and 19 are used for this purpose, and the user should not modify them. Command `\new@mathgroup` is not implemented because math groups are unused. Note that `\newsavebox` and `\newdimen` are the same as `\newbox` and `\newskip` since Tralics does not check redefinition of the command; the command `\newinsert` is not implemented (this requires a box register, a count register, a dimen register and a skip register; each unprocessed float in L<sup>A</sup>T<sub>E</sub>X uses a insert, this may trigger a *too many unprocessed floats* error). The command `\newhelp` is not implemented in Tralics, it allocates no counter.

For instance, if you say `\newcount\Foo`, the allocated number could be 110, if you say `\newskip\Bar`, the number could be 46. In the first case, the result is as if you had said `\countdef\Foo110`. In the case of `\newcounter{foo}`, the result is as `\newcount\c@foo111`. Note that there are only 256 count registers available in T<sub>E</sub>X<sup>10</sup>. You can use registers zero to nine as scratch registers (Do not forget that `\count0` contains the current page number), L<sup>A</sup>T<sub>E</sub>X uses registers 10 to 21 for its allocation mechanism. In the current version, the first free counter is 79. Some other counters are allocated by the class, and the package (in the transcript file, one line is printed for every call to `\allocate`, for instance: `\c@chapter=\count80`; in Tralics, the line looks like `{\countdef\c@foo=\count43}`).

A very important point is that all tokens between `\csname` and `\endcsname` are fully expanded. It is an error if a non-character token remains. Thus it is important to know which commands are expanded, and those that cannot be expanded. The exact rules are in the T<sub>E</sub>Xbook, chapter 20. As a rule of thumb, commands that do no typesetting and modify no internal table can be expanded. More precisely: user defined commands, conversions like `\string`, `\number`, conditionals like `\fi`, marks, and some special commands like `\csname`, `\expandafter`, `\the` can be expanded. A construction like `\csname\char'A\endcsname` is invalid.

If you say `\noexpand\foo`, the result is `\foo`, unexpanded. Example:

```
1 \def\F00{12}
2 %\csname\noexpand\F00\endcsname %bad
3 \edef\xbar{\noexpand\F00}
```

<sup>10</sup>many more in  $\epsilon$ -T<sub>E</sub>X, but they are dynamically allocated; in Tralics2.9, the limit is 4096

```

4 \noexpand\F00
5 \expandafter\textit\F00
6 \expandafter\textit\noexpand\F00
7 \count0=1\F00
8 \count0=1\noexpand\F00

```

Line two is an error: the no-expanded `\F00` is not a character. On line 3, the body of `\xbar` is `'\F00'`, it will be expanded later. The translation of line 4 is empty (the command `\F00` is temporarily seen as `\relax`, and `\relax` does nothing). Because of the `\expandafter`, the argument of `\textit` on line 5 is 1, on line 6 it is 12. On line 7, 112 is put in `\count0`, because `\F00` is expanded. On line 8, 1 is put in the register, and 12 is typeset. On lines 8 and 6, `\F00` is expanded twice, the first expansion being inhibited by the `\noexpand`.

Some quantities are never expanded, for instance `\lowercase` (this is black magic), `\def` (more generally all assignments), `\relax` (it does nothing, but stops scanning integers, dimensions, glue, etc), `\hbox`, `\par`<sup>11</sup>, `\left`, etc. There are cases when an expandable token is not expanded: ten cases in T<sub>E</sub>X, and four additional cases in  $\epsilon$ -T<sub>E</sub>X, these are described in section 6.12. Be careful with constructs like `\csname\endcsname`: L<sup>A</sup>T<sub>E</sub>X may signal an error involving `\unhbox`.

A command can be defined via `\edef` instead of `\def` (`\xdef` is the same as `\edef`, with an implicit `\global` prefix). All tokens, unless defined with `\protected`, in the body of the definition are expanded. Example:

```

\def\A{\B\C} \def\C{1}
\def\Bgroup{{\iffalse}\fi}\def\Egroup{\iffalse{\fi}}
{\let\B\relax \global\edef\D\bgroup{\A\noexpand\C\egroup}}
{\let\B\relax \global\edef\E\Bgroup{\A\noexpand\C\Egroup}}

```

In this example, we consider two groups, that define (locally) a command `\B` and (globally) two commands `\D` and `\E`. The difference between these two commands is that `\bgroup` is an implicit character: when evaluated, it behaves like an opening brace, but it cannot be expanded. On the other hand, `\Bgroup` expands to an open brace. The `\edef` expands tokens following an explicit opening brace. It stops reading after having found an explicit closing brace (resulting from the expansion of `\Egroup`, not `\egroup`). The expansion of `'\A'` is `'\B\C'`, this is expanded again. Since `\B` is `\relax`, it cannot be expanded, and is left unchanged. The expansion of `'\C'` is `'1'`, so that the full expansion of `'\A'` is `'\B1'`. The expansion of `'\noexpand\C'` is `'\C'`. Thus, the example is equivalent to

```

\global\def\D\bgroup{\B1\C\egroup}
\global\E\Bgroup{\B1\C}

```

You can put three `\noexpand` in a row followed by some token `X`. After the first expansion, the result is `\noexpand` followed by `X`, after the second expansion, the result is `X`. In the example that follows, the value of `\B` is `\xbar`.

```

\def\xbar{xbar}
\edef\A{\noexpand\noexpand\noexpand\xbar}
\edef\B{\A}

```

Consider a realistic example like this

```

\def\add#1#2{\edef#1{#1\do{#2}}}
\def\cons#1#2{\begingroup\let\@elt\relax\xdef#1{#1\@elt #2}\endgroup}

```

We can say something like

```

\def\A{\def\B{}} %init
\let\do\relax% just in case
\add\A x, \add\A y, \add\A z,

```

---

<sup>11</sup>Unless redefined by L<sup>A</sup>T<sub>E</sub>X

```
\cons\B{ab}, \cons\B{cd}, \cons\B{ef}.
\show\A\show\B
```

This gives two ways to add some tokens to a list. Because both commands use `\edef`, full expansion is in use; you have to be very careful if the tokens contain macros that can be expanded. For the case of `\add`, we assume that `\do` does nothing; for the case of `\cons`, the command resets `\@elt` to `\relax`. The body of `\A` will be `\do{x}\do{y}\do{z}` and the body of `\B` will be `\@elt ab\@elt cd\@elt ef`. Note the absence of braces: if you really need them, you should add them to the argument of the `\cons` command. The built-in command `\@cons`

The major problem with `\edef` is that it is not aware of assignments. Assume that `\def\@A\B{}`, and `\def\C{B \let\@A\D}`, `\def\E{C}` have been somehow evaluated. Consider now an `\edef` containing `\E`. This implies expansion of `\C`, hence of `'\let\@A\D'`. The `\let` command cannot be expanded. Hence `\@A` is expanded, and you get the following error: *Use of \@A doesn't match its definition* from inside `\C`. You have never heard of this command `\@A`, and never used `\C`! For this reason some commands are made robust: for instance `\hspace` expands to `'\protect\hspace'` (the second command here has a space at the end), and `\protect` is defined to be `\relax`, or `\noexpand`, and sometimes `\string`. This mechanism works only if you use `\protected@edef` instead of `\edef`. (Note: `\protect` behaves like `\string` inside `\protected@write`, which is a variant of `\write`).

## 2.4 Variables in T<sub>E</sub>X

By variable, we mean everything that the user can modify or watch changing. For instance, the current hash table usage is not a variable (it varies, of course, but the value is available only at the end of the run, in the transcript file). The current vertical list is updated whenever a paragraph is split into lines; you cannot access this list, however the `\output` routine gets the part of it that should be typeset on the current page in the box register 255. There are general purpose variables, and specialised ones: for instance `\spacefactor` makes sense only in horizontal mode, and the height of the material on current page (`\pagetotal`) can be used only between paragraphs (in fact, it is updated by T<sub>E</sub>X whenever a line is added to the page; you can consult, and even modify, this value at any time). There are variables that you cannot modify (the version number, for instance) or only once (the magnification), or in the preamble (i.e., L<sup>A</sup>T<sub>E</sub>X reads some variables at begin-document, changes done later to these variables are ignored).

Variables can be classified into two categories depending on their use: in some cases you need to put a prefix before `\foo` if you want to use it, in other cases the prefix is required for modification. For instance, if `\foo` is a user-defined command, you say `\let\foo`, or `\def\foo`, if you want to change the value, and simply `\foo` if you want to use it. In the same fashion `\font\tenrm` defines a font, and `\tenrm` is a use. On the other hand, if you say `\pageno=3`, this will set the current page number to 3 (this is plain T<sub>E</sub>X syntax, the L<sup>A</sup>T<sub>E</sub>X syntax will be explained later). If you say something like `\hskip-\fontdimen2\font`, the `\hskip` command is a prefix that says that the variable that follows will be used. In this case, this is some dimension from a font. Note that `\fontdimen` is a prefix so that `\font` does not define a new font, but refers to the current font. The meaning of the above piece of code is: insert horizontal space, whose amount is the opposite of the second parameter of the current font (i.e., normal interword space).

According to the T<sub>E</sub>Xbook, a `<font>` can be a command like `\tenrm` defined by `\font \tenrm =somefont`, of the null font `\nullfont`, or the current font `\font`, or a family member (`\textfont`, `\scriptfont`, or `\scriptscriptfont`, followed by a 4bit integer). In the case of `\hyphenchar` or `\skewchar`, a `<font>` follows the command. This gives a reference to an integer, the hyphenchar or skewchar of the font (if this integer is not a valid character, the font has no hyphenchar or skewchar). In the case of `\fontdimen`, there is an integer P, a font, and this defines a reference to a dimension. The integer P must be positive and not greater than the number of parameters

in the font (initialised by T<sub>E</sub>X to the number of parameters in the font metric file, 7 for a normal font, 13 for math extension, 22 for math symbols, see T<sub>E</sub>Xbook, appendix F). You can get an error: *Font somefont has only 7 fontdimen parameters*. In Tralics, the value is zero if P is out-of-range. In T<sub>E</sub>X, the last loaded font table can be dynamically increased: if you assign a value at position  $P > M$ , this will increase M. In Tralics, this is possible for all fonts, if  $P < 10^5$ .

The value of a variable can be

- an integer (32bit, signed, with magnitude less than  $2^{31}$ ). The value can be restricted in some cases (to 0-255 if it is an index in a table of registers, to 0-255 if it is a character, etc).
- a dimension, often expressed in pt, (an integer number of times the small unit sp). Normally, the maximum value is  $2^{14}$ pt, but T<sub>E</sub>X does not always check for overflow.
- a glue. Called rubber length in L<sup>A</sup>T<sub>E</sub>X. It is like a dimension with a stretch part, and a shrink part.
- a muglue. Like glue, but only one unit of measure is allowed: mu (math unit).
- a token list. This is a list of tokens (as always, well balanced against explicit braces).
- a font.
- a box (a box contains characters, rules, boxes, penalties, glue, whatsit, etc, but no commands). In Tralics a box contains XML stuff.

You can say ‘\afterassignment\foo\count0=3’; in this case, the command \foo is pushed on a special stack, and popped after assignment is complete. There is only room for one token on this special stack. For instance, if you write the following:

```
\def\fooA{\relax}\def\fooB{\relax}\def\fooC{\relax}\def\fooD{\relax}
\afterassignment \fooA\afterassignment\fooB
\fooC\count0=1\fooD
```

the transcript file of Tralics will contain (in verbose mode)

```
[9] \afterassignment \fooA\afterassignment\fooB
{\afterassignment}
{\afterassignment: \fooA}
{\afterassignment}
{\afterassignment: \fooB}
```

At this point, the after assignment stack contains \fooB. The order of evaluation is now the following: \fooD is expanded; this gives \relax, which terminates scanning of the number; it will be read again, after evaluation of \fooB:

```
[10] \fooC\count0=1\fooD
\fooC ->\relax
{\relax}
{\count}
+scanint for \count->0
\fooD ->\relax
+scanint for \count->1
{after assignment: \fooB}
\fooB ->\relax
{\relax}
{\relax}
```

You can use the \showbox command for displaying the content of a box. This is a little example. It uses \everyhbox and \afterassignment. Note the order in which these tokens are inserted.

```
\everyhbox{3}
\def\foo{12}
```

```
\afterassignment\foo\setbox0=\hbox{4}
\showbox0
```

This is what T<sub>E</sub>X prints in the log file:

```
> \box0=
\hbox(6.4151+0.0)x19.99512
.\T1/cmr/m/n/10 1
.\T1/cmr/m/n/10 2
.\T1/cmr/m/n/10 3
.\T1/cmr/m/n/10 4
```

The first line of the trace starts with `\hbox` or `\vbox`, followed by the dimensions (height, depth, width; the unit is ‘pt’ by default), optionally followed by ‘shifted 27.1’ if the the box is shifted, and by ‘glue set 0.19’ if the glue has to be stretched or shrunk. After that, you will see the content of the box, one line per item (no more than `\showboxbreadth` lines are printed per box), each item is preceded by a context (a sequence of  $N$  dots at depth  $N$ , tokens at depth greater than `\showboxdepth` are not shown). In the box, you can see things like ‘`\penalty -51`’ or ‘`\kern 28.45274`’ or ‘`\glue 3.0 plus 1.0`’ or ‘`\glue(\baselineskip) 2.28015`’ (this last glue is inserted automatically by T<sub>E</sub>X, it knows where it comes from, so that the name can be printed), `\special{...}`, `\write4{\indexentry...}`. The interesting point in the last object is that we have a list of tokens that will be evaluated later (when the page is shipped out). Tralics does not put `\kern`, `\penalty`, neither `\glue` in a box. The `\special` command is not implemented; finally `\write` is never delayed. In our example, the box contains four items, which are characters (T<sub>E</sub>X shows a command that contains the name of the font; in our example, the font is something like ‘`ecrm1000`’).

In Tralics, you would see the same characters, but no font and no size. On the other hand, you can say something like

```
\everyxbox{Test}
\setbox0=\xbox{foo}{1\xbox{bar}{2} %
  \AddAttToLast{x}{1}\AddAttToCurrent{y}{2}3}
\showbox0
```

and you will see

```
<foo y='2'>Test1<bar x='1'>Test2</bar> 3</foo>
```

Note the two commands that were used to add attributes to the current XML elements, and the last constructed one. We have added another command, `\XMLaddatt` that takes as optional argument the id of the element to which the attribute value pair should be added. This is an integer; if omitted, the current element is used. You can use `\XMLlastid` or `\XMLcurrentid` (there are references to variables, you must use `\the` if you want the value). If you want to overwrite an existing attribute pair, you must use a star. The previous example can be written like this:

```
\everyxbox{Test}
\setbox0=\xbox{foo}{1\xbox{bar}{2} %
  \XMLaddatt[\the\xMLlastid]{x}{1}\XMLaddatt[\the\xMLcurrentid]{y}{22}%
  \XMLaddatt[\the\xMLlastid]{x}{11}\XMLaddatt*{y}{2}3}
\showbox0
```

If `\foo` is any command then `\show\foo` will show its value. Here are some examples

```
\def\Bar#1#{#1} \show\Bar
\let\foo\par \show\foo
\renewcommand\foo[2][toto]{#1#2} \show\foo
\let\foo=1 \show\foo
\let\foo=_ \show\foo
```

```
\let\foo=\undef \show\foo
\show\bgroup
```

This is what Tralics prints (it differs slightly from the L<sup>A</sup>T<sub>E</sub>X output)

```
\Bar=macro: #1#->#1.
\foo=\par.
\foo=opt \long macro: toto#2->#1#2
\foo=the character 1.
\foo=subscript character _.
\foo=undefined.
\bgroup=begin-group character {.
```

In the case of a variable, you can say `\the\foo`, the result is a token list that represents the value of `\foo` (if `\foo` is a token list, `\the\foo` is the value of `\foo`, otherwise, it is a list of characters). The command `\showthe` will show the value, i.e. print on the terminal the token list returned by `\the`. Example

```
\def\Show#1{\the#1\showthe#1}
\widowpenalty=3 \Show\widowpenalty
\parindent1.5pt \Show\parindent
\leftskip = 1pt plus 2fil minus 4fill \Show\leftskip
\thinmuskip = 3mu plus -2fil minus 4fill \Show\thinmuskip
\count0=17 \Show{\count0}
\dimen0=17pt \Show{\dimen0}
\skip0=17pt plus 1 pt minus 2pt \Show{\skip0}
\muskip0=17mu plus 1 mu minus 2mu \Show{\muskip0}
\Show{\catcode'\A}
\Show{\lccode'\B}
\Show\inputlineno
\font\xa=cmr10 at 11truept
\fontdimen6\xa = 11pt \hyphenchar\xa='\-
\Show{\fontdimen6\xa}
\Show{\hyphenchar\xa}
\chardef\foo25
\Show\foo
\Show\xa
\toks0={\foo = \foo} \def\foo{foo}
\Show{\toks0}
```

This is what Tralics prints on the screen.

```
\show: 3
\show: 1.5pt
\show: 1.0pt plus 2.0fil minus 4.0fill
\show: 3.0mu plus -2.0fil minus 4.0fill
\show: 17
\show: 17.0pt
\show: 17.0pt plus 1.0pt minus 2.0pt
\show: 17.0mu plus 1.0mu minus 2.0mu
\show: 11
\show: 98
\show: 79
\show: 11.0pt
\show: 45
\show: 25
\show: cmr10
\show: \foo= \foo
```

The typeset result is:  $31.5pt0.0pt0.0mu1717.0pt17.0pt$  plus  $1.0pt$  minus  $2.0pt17.0mu$  plus  $1.0mu$  minus  $2.0mu11987911.0pt$  45 25cmr10 foo= foo<sup>12</sup>.

In the case of `\the\foo`, `\showthe\foo`, `\advance\foo`, `\multiply\foo`, `\divide\foo`, the token that follows the first command is fully expanded.

## 2.5 All the variables

All variables (exceptions will be given later) are in the table of equivalents: this table contains the current meaning of quantities that are saved/restored by the grouping mechanism of T<sub>E</sub>X. In T<sub>E</sub>X this table is divided into six parts; in Tralics, the layout is slightly different, for instance, because T<sub>E</sub>X makes a heavy using of glue (each space character produces a glue item), while Tralics ignores them completely. This big table contains the following objects

1. the current equivalent of single character control sequences (for `~` as well as `\~`);
2. the hash table (in Tralics, there are two such tables, if the command `\foo` produces `<bar gee='true'>`, the three strings 'bar', 'gee' and 'true' are in a special table).
3. all glue parameters.
4. all quantities that fit on 16 bits.
5. all integers.
6. all dimensions.

The glue parameters are the following (unused by Tralics, initialised to 0, unless stated otherwise).

- `\lineskip`: interline glue if `\baselineskip` is infeasible.
- `\baselineskip`: desired glue between baselines.
- `\parskip`: extra glue just above a paragraph.
- `\abovedisplayskip`: extra glue just above displayed math.
- `\belowdisplayskip`: extra glue just below displayed math.
- `\abovedisplayshortskip`: glue above displayed math following short lines.
- `\belowdisplayshortskip`: glue below displayed math following short lines.
- `\leftskip`: glue at left of justified lines.
- `\rightskip`: glue at right of justified lines. L<sup>A</sup>T<sub>E</sub>X uses `\leftskip` and `\rightskip` for commands like `\centering`, `\raggedright` etc. Unused by Tralics. On the other hand there is `\nocentering`, whose effect is the same as setting both `\leftskip` and `\rightskip` to zero.
- `\topskip`: glue at top of main pages.
- `\splittopskip`: glue at top of split pages.
- `\tabskip`: glue between aligned entries.
- `\spaceskip`: glue between words.
- `\xspaceskip`: glue after sentences.
- `\parfillskip`: glue on last line of paragraph.
- `\thinmuskip`: thin space in math formula.
- `\medmuskip`: medium space in math formula.
- `\thickmuskip`: thick space in math formula.
- `\itemsep`: defined by L<sup>A</sup>T<sub>E</sub>X. Rubber space between successive items in a list.

<sup>12</sup>This might be unexpected, and differs from the L<sup>A</sup>T<sub>E</sub>X output. In fact, the command `\the` is expanded before the key word 'fill' is completely read, hence before the assignment of the glue, but `\show` shows the value after the assignment. The difference with L<sup>A</sup>T<sub>E</sub>X is the initial value of `\thinmuskip`.



- `\labelsep`: defined by L<sup>A</sup>T<sub>E</sub>X. The space between the end of the label box and the text of the item in a list.
- `\parsep`: defined by L<sup>A</sup>T<sub>E</sub>X. Rubber space between paragraphs within an item.
- `\fill`: defined by L<sup>A</sup>T<sub>E</sub>X. Holds 0pt plus 1fill. You should not modify it.
- `\smallskipamount`, `\medskipamount`, `\bigskipamount`. Quantities defined by Tralics in the same way as L<sup>A</sup>T<sub>E</sub>X, but unused.
- `\floatsep`, `\textfloatsep`, `\intertextsep`, `\dblfloatsep`, `\dbltextfloatsep`: glue inserted by L<sup>A</sup>T<sub>E</sub>X between float and other material.
- `\hideskip`. Holds -1000pt plus 1 fill. This is used in T<sub>E</sub>X to implement `\hidewidth`, a construction not implemented in Tralics.
- `\z@skip`. You should leave this quantity unchanged
- `\normalbaselineskip`, `\normallineskip`. Use by L<sup>A</sup>T<sub>E</sub>X for font switching.
- `\listparindent`, `\topsep`, `\partopsep`. Some parameters used by L<sup>A</sup>T<sub>E</sub>X.
- `\@tempskipa`, `\@tempskipb`: temporary skip

The token parameters are the following (initially empty; unused by Tralics unless stated otherwise):

- `\parshape`: for funny paragraphs (not really a token list).
- `\output`: user defined output routine.
- `\everypar`: tokens inserted by T<sub>E</sub>X at start of every paragraph.
- `\everymath`: tokens inserted by T<sub>E</sub>X and Tralics at the start of every non-display math formula.
- `\everydisplay`: tokens inserted by T<sub>E</sub>X and Tralics at the start of every display math formula.
- `\everyhbox`: tokens inserted by T<sub>E</sub>X and Tralics at the start of every `\hbox`.
- `\everyvbox`: tokens inserted by T<sub>E</sub>X and Tralics at the start of every `\vbox`.
- `\everyxbox`: tokens inserted by Tralics at the start of every `\xbox`.
- `\everyjob`: tokens inserted by T<sub>E</sub>X and Tralics at the start of every job. This must be defined by the format (for T<sub>E</sub>X), otherwise it is useless; in Tralics, you must put `everyjob = "\something{like this}"` in the configuration file.
- `\everycr`: tokens inserted by T<sub>E</sub>X after every `\cr` or non redundant `\cr`.
- `\errhelp`: tokens that will be printed by T<sub>E</sub>X in case of user-error.
- `\everyeof`: tokens inserted by  $\epsilon$ -T<sub>E</sub>X at each end-of-file.
- `\@temptokena`: scratch register.

The integer parameters are the following. These parameters are zero, unless stated otherwise.

- `\pretolerance`: badness tolerance before hyphenation (initialised to 100 by Tralics).
- `\tolerance`: badness tolerance after hyphenation (initialised to 200 by Tralics).
- `\linepenalty`: amount added to the badness of every line in a paragraph.
- `\hyphenpenalty`: penalty for break after discretionary hyphen.
- `\exhyphenpenalty`: penalty for break after explicit hyphen.
- `\clubpenalty`: penalty for creating a club line at a bottom of a page.
- `\widowpenalty`: penalty for creating a widow line at top of page.
- `\displaywidowpenalty`: ditto, just before a display.
- `\brokenpenalty`: penalty for breaking a page at a broken line.
- `\binoppenalty`: penalty for breaking after a binary operation in a math formula.
- `\relpenalty`: penalty for breaking after a relation in a math formula.
- `\predisplaypenalty`: penalty for breaking just before a displayed formula.

- `\postdisplaypenalty`: penalty for breaking just after a displayed formula.
- `\interlinepenalty`: additional penalty for a page break between lines.
- `\doublehyphendemerits`: demerits for consecutive broken lines.
- `\finalhyphendemerits`: demerits for a penultimate broken line.
- `\adjdemerits`: demerits for adjacent incompatible lines.
- `\mag`: magnification ratio, times 1000.
- `\delimiterfactor`: ratio for variable-size delimiters.
- `\looseness`: change to the number of lines in a paragraph.
- `\time`: current time of day. Number of minutes since midnight, computed by Tralics at start of run.
- `\day`: current day of the month (between 1 and 31).
- `\month`: current month of the year (between 1 and 12).
- `\year`: current year of our Lord. The initial values of `\time`, `\day`, `\month`, `\year`, are printed in the transcript file by Tralics in the following format: 2006/10/24 10:18:10
- `\showboxbreadth`: maximum items per level when boxes are shown (when Tralics shows the content of a box, it always shows everything).
- `\showboxdepth`: maximum level when boxes are shown (when Tralics shows the content of a box, it always shows everything).
- `\hbadness`: hboxes exceeding this badness will be shown.
- `\vbadness`: vboxes exceeding this badness will be shown.
- `\pausing`: pause after each line is read from a file. In Tralics there is no interaction with the user.
- `\tracingonline`: show diagnostic output on terminal. In verbose mode, this variable, and some other ones are set to a non-zero value, as explained in section 6.6.
- `\tracingmacros`: show macros as they are being expanded.
- `\tracingstats`: show memory usage if  $\TeX$  knows it.
- `\tracingparagraphs`: show line-break calculations.
- `\tracingpages`: show page-break calculations.
- `\tracingoutput`: show boxes when they are shipped out.
- `\tracinglostchars`: show characters that aren't in the font.
- `\tracingcommands`: show command codes.
- `\tracingrestores`: show equivalents when they are restored.
- `\uchyph`: hyphenate words beginning with a capital letter.
- `\outputpenalty`: penalty found at current page break.
- `\maxdeadcycles`: bound on consecutive dead cycles of output.
- `\insertpenalties`. Is the sum of all penalties for split insertions on the current page.
- `\spacefactor`: According to the  $\TeX$ book: “the exact amount of glue inserted by a space depends on `\spacefactor`, the current font, and the `\spaceskip` and `\xspaceskip` parameters as described in Chapter 12.”
- `\hangafter`: hanging indentation changes after this many lines.
- `\floatingpenalty`: penalty for insertions heldover after a split.
- `\globaldefs`: override `\global` specifications.
- `\fam`: current family.
- `\escapechar`: escape character for token output. Initialised by Tralics to backslash.
- `\defaultthyphenchar`: value of `\hyphenchar` when a font is loaded.
- `\defaultskewchar`: value of `\skewchar` when a font is loaded.
- `\endlinechar`: character placed at the right end of the buffer when reading a new line. Initialised by Tralics to CR (ascii 13).
- `\newlinechar`: character that prints as a LF. Initialised by Tralics to LF, but not used.

- `\language`: the current set of hyphenation rules. For Tralics, 0 means English, 1 means French, 2 means German, and 3 stands for any other language.
- `\lefthyphenmin`: minimum left hyphenation fragment size.
- `\righthyphenmin`: minimum right hyphenation fragment size.
- `\holdinginserts`: do not remove insertion nodes from `\box255`.
- `\errorcontextlines`: maximum intermediate line pairs shown. In Tralics, the context of an error is not shown.
- `\tracingassigns`, `\tracinggroups`, `\tracingifs`, `\tracingscantokens`, `\tracingnesting`. These are extensions of  $\epsilon$ -T<sub>E</sub>X, that control what is printed in the transcript file.
- `\tracingmath`: controls what is printed when Tralics interprets a math formula.
- `\predisplaydirection`, `\lastlinefit`, `\savingdiscards`, `\savingshyphcodes`. These are other  $\epsilon$ -T<sub>E</sub>X extensions, described in section 6.12
- `\FPseed`. This is defined only when the FP package is loaded.
- `\TeXXeTstate` controls  $\epsilon$ -T<sub>E</sub>X bidirectional printed, unused by Tralics.
- `\@nomathml`. If this number is non-zero, math formulas are not converted into MathML expressions.
- `\notrivialmath`. This controls how some *trivial* math formulas should be translated as text.
- `\hyphenchar`, `\skewchar`. The command should be followed by a font reference, and this is a reference to the hyphenchar or skewchar of the font.
- `\interlinepenalties`, `\clubpenalties`, `\widowpenalties`, and `\displaywidowpenalties`. Commands defined by  $\epsilon$ -T<sub>E</sub>X, described in section 6.12. Like `\parshape`; the command reads an integer  $n$ , and returns the corresponding value in the slot, it can also read  $n$  integers and store them.
- `\@mathversion`. If the value of the counter is positive, then a bold variant is used for math characters, if possible. The command `\mathversion` reads an argument; it sets the counter to one if the argument is ‘bold’, to zero otherwise.
- `\@tempcnta`, `\@tempcnta`: Scratch counter.
- `\interfootnotelinepenalty` contains penalty added by L<sup>A</sup>T<sub>E</sub>X in footnotes.
- `\interdisplaylinepenalty` contains penalty inserted by L<sup>A</sup>T<sub>E</sub>X between lines of equations/

The following quantities are read only variables. They are integers, unless stated otherwise.

- `\lastpenalty` returns the value of the last item on the current list, if it is a penalty (always zero in Tralics).
- `\lastkern` returns the value of the last item on the current list, if it is a kern. This is a dimension, 0pt in Tralics.
- `\lastskip` returns the value of the last item on the current list, if it is glue. This is a dimension, 0pt in Tralics.
- `\lastnodetype` is an  $\epsilon$ -T<sub>E</sub>X extension containing the type of the last item on the current list; always zero in Tralics.
- `\inputlineno` contains the current input line number.
- `\badness` contains the current badness (always zero in Tralics).
- `\XMLlastid` contains the unique identifier of the most recently created XML element. Defined only by Tralics.
- `\XMLcurrentid` contains the unique identifier of the current XML element. Defined only by Tralics.
- `\currentgrouplevel` contains the current grouping level (index in the semantic stack); it is an  $\epsilon$ -T<sub>E</sub>X extension.
- `\currentgroupstype` contains the type of the current semantic group; it is an  $\epsilon$ -T<sub>E</sub>X extension, explained in section 6.12.

- `\currentiflevel`, `\currentifttype`, `\currentifbranch` are  $\epsilon$ -TeX extensions described in section 6.12. The variables contain information about the condition stack.
- `\eTeXversion` is defined by  $\epsilon$ -TeX, contains its revision number.
- `\fontcharwd`, `\fontcharht`, `\fontchardp`, `\fontcharic`. These  $\epsilon$ -TeX extension commands read a font identifier, and an integer (character position). They return a property of the character in the font, always 0 in Tralics.
- `\parshapelength`, `\parshapeindent`, `\parshapedimen`. Commands that read an integer and give properties of the current paragraph shape.
- `\numexpr`, `\dimexpr`, `\glueexpr`, `\muexpr`: these commands read an expression, that can be a number, a dimension, a glue or a math dimension, with an extended syntax. They are  $\epsilon$ -TeX extensions described in section 6.12.
- `\gluestretchorder`, `\glueshrinkorder`, `\gluestretch`, `\glueshrink`. These commands read some glue, and extract a part of it. They are  $\epsilon$ -TeX extensions described in section 6.12.
- `\gluetomu`, `\mutoglu`: These commands read some glue, or math glue, and convert. They are  $\epsilon$ -TeX extensions described in section 6.12.

The counters defined in Tralics are the following. The counters are not used unless specified otherwise, but you can say `\renewcommand\thepage{...}`, this is not an error.

- `page`. This is `\count0`. Tralics initialises it to 1.
- `enumi`, `enumii`, `enumiii`, `enumiv`: for enumerations. Unless specified otherwise, the enumeration counter is updated, but the value is not used.
- `part`, for parts of a book.
- `chapter`, `subsection`, `section`, `subsection`, `paragraph`, `subparagraph`. Each counter depends on the preceding one.
- `FancyVerbLine`, this is the default counter used by the `verbatim` environment for counting lines.
- `footnote`. Each call to the `\footnote` command increments this counter, but the value is not used.
- `mpfootnote`. Minipage footnotes.
- `bottomnumber`, `topnumber`: used by latex for float placement.
- `totalnumber`, `dbltopnumber`:

The dimension parameters are the following:

- `\parindent`: indentation of paragraphs.
- `\mathsurround`: space around math in text.
- `\lineskiplimit`: threshold where `\baselineskip` switches to `\lineskip`.
- `\hsize`: line width in horizontal mode.
- `\vsize`: page height in vertical mode.
- `\maxdepth`: maximum depth of boxes on main pages.
- `\splitmaxdepth`: maximum depth of boxes on split pages.
- `\boxmaxdepth`: maximum depth of explicit vboxes.
- `\hfuzz`: tolerance for overfull hbox messages.
- `\vfuzz`: tolerance for overfull vbox messages.
- `\delimitershortfall`: maximum amount uncovered by variable delimiters.
- `\nulldelimiterspace`: blank space in null delimiters.
- `\scriptspace`: extra space after subscript or superscript.
- `\prelispaysize`: length of text preceding a display.
- `\displaywidth`: length of line for displayed equation.

- `\displayindent`: indentation of line for displayed equation.
- `\overfullrule`: width of rule that identifies overfull hboxes.
- `\hangindent`: amount of hanging indentation.
- `\hoffset`: amount of horizontal offset when shipping pages out.
- `\voffset`: amount of vertical offset when shipping pages out.
- `\emergencystretch`: reduces badnesses on final pass of line-breaking.
- `\z@`: You should forget that this is a variable, and use it only as the constant zero.
- `\p@`: contains 1pt.
- `\evensidemargin`: defined by L<sup>A</sup>T<sub>E</sub>X, left margin for even pages.
- `\oddsidemargin`: defined by L<sup>A</sup>T<sub>E</sub>X, left margin for odd pages.
- `\leftmargin`: defined by L<sup>A</sup>T<sub>E</sub>X. Space between the left margin of the page and the left margin of the text. Depends on the list level.
- `\rightmargin`: defined by L<sup>A</sup>T<sub>E</sub>X. Similar to `\leftmargin`, but for the right margin.
- `\leftmargini`: defined by L<sup>A</sup>T<sub>E</sub>X. Value of left margin for a list at level one.
- `\leftmarginii`: defined by L<sup>A</sup>T<sub>E</sub>X. Value of left margin for a list at level two.
- `\leftmarginiii`: defined by L<sup>A</sup>T<sub>E</sub>X. Value of left margin for a list at level three.
- `\leftmarginiv`: defined by L<sup>A</sup>T<sub>E</sub>X. Value of left margin for a list at level four.
- `\leftmarginv`: defined by L<sup>A</sup>T<sub>E</sub>X. Value of left margin for a list at level five.
- `\leftmarginvi`: defined by L<sup>A</sup>T<sub>E</sub>X. Value of left margin for a list at level six.
- `\itemindent`: defined by L<sup>A</sup>T<sub>E</sub>X. Extra indentation added to the horizontal indentation of the text part of the first line of an item in a list.
- `\labelwidth`: defined by L<sup>A</sup>T<sub>E</sub>X. Nominal width of the box containing the label of an item.
- `\fboxsep`: defined by L<sup>A</sup>T<sub>E</sub>X. Space left between the edge of the box and its content produced by `\fbox` or `\framebox`.
- `\fboxrule`: defined by L<sup>A</sup>T<sub>E</sub>X. Width of line produced by `\fbox` or `\framebox`.
- `\arraycolsep`: defined by L<sup>A</sup>T<sub>E</sub>X, contains half of the width of the default horizontal space in a math array.
- `\tabcolsep`: defined by L<sup>A</sup>T<sub>E</sub>X, contains half of the width of the default horizontal space in a text array.
- `\arrayrulewidth`: defined by L<sup>A</sup>T<sub>E</sub>X, contains the width of vertical rules in an array.
- `\doublerulewidth`: defined by L<sup>A</sup>T<sub>E</sub>X, contains the distance between two vertical rules in an array.
- `\normallineskiplimit`: default value of `\lineskiplimit`.
- `\epsfxsize`. Used for including images.
- `\epsfysize`. Used for including images.
- `\unitlength`. Used for the picture environment. Initialised by Tralics to 1pt.
- `\textwidth`. Width of the text. Initialised by Tralics to 427pt. More or less 15cm.
- `\textheight`. Height of the text. Initialised by Tralics to 570pt. More or less 20cm.
- `\columnwidth`: this is defined by L<sup>A</sup>T<sub>E</sub>X as containing the current line width; in a two-column document this is the half of `\textwidth` minus `\columnsep`, otherwise it is `\textwidth`. L<sup>A</sup>T<sub>E</sub>X copies this value in `\hsize` in some cases (for instance, when switching between one column and two columns).
- `\marginparwidth`, `\marginparsep`, `\marginparpush`: three parameters defined by L<sup>A</sup>T<sub>E</sub>X for placement of marginal notes.
- `\topmargin`, `\headheight`, `\headsep`: these three dimensions are used by L<sup>A</sup>T<sub>E</sub>X for controlling the header of a page.
- `\footskip`: this dimension (not glus) is used by L<sup>A</sup>T<sub>E</sub>X for controlling the footer of a page.
- `\columnsep`, `\columnseprule`: this is defined by L<sup>A</sup>T<sub>E</sub>X as containing the distance between columns and width of the vertical rule between columns in a multi-column document.

- `\linewidth`: this is defined by L<sup>A</sup>T<sub>E</sub>X as containing the current line width (typically, this is `\columnwidth` minus the margins introduced by list environment). In Tralics, these two commands are undefined, except that you can use it as a unit of measure inside the optional argument of `\includegraphics` for ‘height’ and ‘width’: the value is 15cm.
- `\@tempdima`, `\@tempdima`, `\@tempdima`: three scratch registers.
- `\paperheight`, `\paperwidth`: contains the size of the paper on which the document will be printed; is A4 by default in Tralics, namely 297 and 210mm.
- `\jot`: a small quantity, in fact 3pt.
- `\maxdimen`: a large quantity, in fact the largest dimension that can be stored is 2<sup>14</sup>pt minus one sp.

The registers are the following

- `\count xxx`: table of 256 “count” registers.
- `\dimen xxx`: table of 256 “dimen” registers.
- `\skip xxx`: table of 256 “skip” registers.
- `\muskip xxx`: table of 256 “muskip” registers.
- `\toks xxx`: table of 256 token lists.
- `\box xxx`: table of 256 box registers.
- `\wd xxx`: Width of box N. If you ask for the value of the width, you will get zero. If you modify the width, nothing happens.
- `\ht xxx`: Height of box N.
- `\dp xxx`: Depth of box N.
- `\delcode xxx`: table of 256 delimiter code mappings. Unused by Tralics.
- `\catcode xxx`: table of 256 category codes. In Tralics, `\{}``$``&``#``^``~``%` have standard category codes. All letters have category code 11. Space, tabulation, non-breaking space (character 160) have category 10. All other characters have category 12.
- `\sfcode xxx`: table of 256 spacefactor mappings. Not used by Tralics.
- `\lccode xxx`: table of 256 lowercase equivalents.
- `\uccode xxx`: table of 256 uppercase equivalents. Since Tralics2.9, uppercase equivalent of `ÿ` is `Ÿ` (hex FF and 178), and vice-versa.
- `\mathcode xxx`: table of 256 math mode mappings. The math code of a character is ignored, except that a value of 32768, that makes the character active in math mode.

Since version 2.9 of Tralics, all characters have 16 bits, so that the number of characters 256 should be replaced by 2<sup>16</sup>. i.e. 65536, in the sizes above. Moreover, the number of other registers (from `\count` to `\box` above) has been increased to 4096.

Some quantities are meaningful when T<sub>E</sub>X makes lines into pages. The dimension `\pagegoal` contains the current page height (minus the size of all potential insertions). The current page height has a natural value in `\pagetotal` and a shrink part in `\pageshrink`, the stretch part is in `\pagestretch`, its ‘fil’ part is in `\pagefilstretch`, its ‘fill’ part in `\pagefillstretch` and its ‘filll’ part in `\pagefilllstretch`. The depth of the box is a constant dimension, in `\pagedepth`. Whenever the output routine is called, T<sub>E</sub>X increases the value of the integer counter `\deadcycles`; an error is signaled if the value is too big, it is reset to zero by `\shipout`. In `\prevdepth`, you can find the depth of the most recent box on the current vertical list, in the integer `\prevgraf` the number of lines in the most recent paragraph that has been completed or partially completed. Of course, all these value are zero in Tralics.

In plain T<sub>E</sub>X, you can use `\nointerlineskip` and `\offinterlineskip`. These commands change the value of `\prevdepth`. They are ignored by Tralics.

## 2.6 Using the variables

There are three routines defined in Tralics, named `scanint`, `scandimen` and `scanglue` that read a integer, a dimension and glue. Assume that `\count0` is 1, `\parindent` is 3pt, and you say `\skip\count0=2pt plus \parindent\relax`. The transcript file of Tralics will contain

```
[346] \skip\count0=2pt plus \parindent \relax
{\skip}
+scanint for \count->0
+scanint for \skip->1
+scanint for \skip->2
+scandimen for \skip->2.0pt
+scandimen for \skip->3.0pt
{scanglue 2.0pt plus 3.0pt\relax }
{changing \skip1=0.0pt into \skip1=2.0pt plus 3.0pt}
{\relax}
```

The exact rules will be given later. The following happens here: After `\skip` there is an integer, an optional equals sign, then glue. After `\count` there has to be an integer. Thus, `scanint` reads an integer for `\count`, and an other one for `\skip`. A glue item is formed of a dimension (the natural width), optionally followed by ‘plus’ and a dimension (the stretch part), optionally followed by ‘minus’ and a dimension (the shrink part). In this case, there is no stretch part, because of `\relax`. The second dimension comes from the variable `\parindent`; the first dimension is explicit: the integer part of the dimension is read by `scanint`.

An integer can be explicit or implicit: an implicit integer comes from a command (it can be a variable like `\date`, or a constant like `\active`). In all other cases, the number can be followed by one optional space. In general, the number will be given as a non-empty sequence of digits, like 01239; you can specify digits in base 16 as “FF, this is 255, in this case, letters between A and F (uppercase, category 11 or 12) are allowed. You can specify digits in base 8 as ‘177, this is 127. You can also specify a digit as a character: ‘A is 65. You can say ‘\A, this is also 65; note that a backslash is needed in cases like ‘\%. Only one letter is allowed in the command name, digits and quotes must have category 12.

An integer or a dimension can be preceded by a sign. This is a sequence of arbitrary length, formed of spaces or `+12` or `-12` signs. If the number of minus signs is odd, this changes the sign of the result. Hence if you say `\count0=++'77` and `\count1=-\count0`, this will put 63 in `\count1`.

A dimension can be implicit or explicit. You can say `\count0=\dimen0`: in this case the value of the dimension in sp units is put in the count register. You can say `\dimen0=\skip0`: the shrink and stretch part of the glue is ignored. You can also say `\count0=\skip0` (guess what happens). It is not possible to convert (directly) an integer to a dimension or glue. An explicit dimension is formed of a factor and a unit of measure. The factor can be an integer (hence `-'77pt` is a valid dimension), or a decimal number (given in base ten, like 1.5, or 1,5). Units can be `pt`, `pc`, `in`, `bp`, `cm`, `mm`, `dd`, `cc`, `sp`. The case is irrelevant: `Pt`, `pt`, `PT` and `pT` are valid, the category code may be anything (it cannot be active, because everything is fully expanded). Units shown above can be preceded by `true` (note that Tralics ignores magnification, thus the ‘true’ prefix). Units can also be `em` or `ex`. These values depend on the current font. Tralics always assumes that the font is a ten point one. A unit of measure can also be an integer, a dimension, or glue. For instance `\dimen0=1\count0` will multiply the value of `\count0` by one. This is the dual to `\count0=\dimen0`. You can say `\parindent=1.2\parindent` if you want to increase it by 20%.

A glue is formed of three parts: a dimension, a stretch part, and a shrink part. The stretch part can be a dimension (it can use special units like ‘fil’, ‘fill’ and ‘filll’, these are called infinite, of first, second and third order). You can say `\skip0=0pt plus 1fil`. For some strange reasons, after `fil` you can put a second L, and a third one. As is the case with other units like

ex or em, the case is irrelevant. Spaces are ignored after the L. Moreover, T<sub>E</sub>X continues scanning for an L after having found ‘filll’; if found, it signals the following error: *Illegal unit of measure (replaced by filll)*. In the case of `\skip0=2\skip0`, the equals sign is followed by a dimension: there is a factor 2, and a unit (the fixed part of `\skip0`). As a consequence, this multiplies by two the fixed part of the glue, and sets the shrink and stretch part to zero (unless the code above is followed by ‘plus’ or ‘minus’).

Note: if you say `\chardef\foo=123\foo`, then `\foo` is made equal to 123: the first thing that `\chardef` does it to make `\foo` the same as `\relax`, so that scanning stops after digit 3. On the other hand in the case of `\count0=3\ifnum...` the conditional is evaluated while reading the number, thus before the assignment is complete. In particular, if the test compares `\count0` with something else, the value might be different from three. Assume that `\count0` and `\count13` contain the value 7. What happens if you say: `\count0=2\ifnum\count0=\count13\fi4`? It will put 2 in `\count0` and typeset 4. In fact, after the digit 3 is sensed, the `\fi` token terminates the `\ifnum`. It does so by inserting a `\relax` token, and a second `\fi` token. The effect of `\relax` is to finish reading the number. Thus `\ifnum` can compare the two values. If these two values are different, the expansion of the conditional is empty, and 24 is put in `\count0`. But the test is true, and T<sub>E</sub>X reads again the inserted `\relax`: it has as effect to stop scanning of the number 2. After that the inserted `\fi` is read. The transcript file of T<sub>R</sub>alics might look like the following. Since version 2.9, the transcript file contains also assignments. So you can see the order: when the `\fi` is seen, the last `\count`, hence the RHS of the equality, is not yet evaluated and a `\fi` token is inserted, preceded by a `\relax` token; these are evaluated later; the `\relax` token is seen by `\count`, and left unchanged. After that, we have the number 13, hence the value of `\count13`, hence the truth value of the test. Now, the body of the conditional is read; it consists solely of the `\relax`. This one is seen by the first `\count`, that has the value needed by the assignment. After the assignment is complete, the `\relax` is considered again: it is read, and the inserted `\fi` is evaluated.

```
[2677] \count0=2\ifnum\count0=\count13\fi4
{\count}
+scanint for \count->0
+\ifnum989
+scanint for \count->0
+scanint for \ifnum->7
+\fi989
+scanint for \count->13
+scanint for \ifnum->7
+iftest989 true
+scanint for \count->2
{changing \count0=7 into \count0=2}
{\relax}
+\fi989
Character sequence: 4 .
```

A token list is a like a command without arguments. You can say `\foo={ABC}` if you want to put something it it, and `\the\foo` if you want to use the list. The equals sign is optional. You can insert a `\relax` between the equals sign and the opening brace. In the example that follows, you can see that, after the optional equals sign, you can put as many spaces or `\relax` tokens as you like; tokens are expanded, as long as no brace is found. The last line of the example shows that the token that follows `\the` is expanded (if `\the` itself is expanded). Thus, the last line adds some tokens at the end of the list. Note the space in `\A`: without it, T<sub>E</sub>X would see something like `\the\toks0\the\toks0`, and the second `\the` is expanded by the scanint routine, so that this inserts in `\toks0` the content of `\toks01` followed by a sharp sign.



```

\def\myrelax{ \relax}
\def\A{\toks0 }
\A=\relax\myrelax{1#}
\A=\expandafter{\the\A \the\A}\showthe\toks0

```

The `\showthe` command prints ‘1##\the \A’, but only a single # is in the list.

We have seen on page 33 how to use `\cons` to add some tokens to a command via `\edef`. The code that follows adds tokens to a list. The command is called `\addto@hook` in Tralics and is long, but the body is the same.

```

\def\addtohook#1#2{#1\expandafter{\the#1#2}}
\newtoks\foo
\addtohook{\foo}{\do{A}}\addtohook{\foo}{\do{B}}\addtohook{\foo}{\do{C}}

```

The command `\newtoks` defines its argument as a reference to a token register, for instance `\toks23`. Whenever you use `\addtohook` with `\A` as first argument, it is like the assignment `\A=\expandafter{\the\A...}` shown in the previous example. Other example

```

\T\expandafter{\L}
% \xdef\L{... \the\T}
\xdef\L{\catcode\the\count@=\the\catcode\the\count@\relax\the\T}

```

Let’s assume that `\L` is a parameterless command, and `\T` a reference to a token register. The first line puts the value of `\L` in `\T`. The second line explains what we do in the third one. Remember that `\xdef` expands everything in the body. All tokens are fully expanded (except that the result of `\the` is not expanded). As a result, this will put some tokens in front of `\L`. Let’s explain which tokens. We assume that `\count@` is a reference to some counter, that the counter contains 65, this is the ASCII code of the upper case letter A, and we assume that the category code is 11. The first token is `\catcode`, it cannot be expanded, it will be left unchanged. The second token is `\the`. It can be expanded, the result is the value of the counter, the two characters 65. The equals sign cannot be expanded. Then comes `\the`; this expands what follows. The `\catcode` command reads a number. Because of `\the`, it reads two digits 6 and 5, and looks at the `\relax`. Note: this `\the` is useless, this example revealed a bug in Tralics. This is the log of Tralics. The last line indicates the value of `\L`:

```

[18] \xdef\L{\catcode\the\countx=\the\catcode\the\countx\relax\the\T}
{\xdef}
{\the}
{\the \countx}
\the->65.
{\the}
{\the \catcode}
{\the}
{\the \countx}
\the->65.
+scanint for \catcode->65
\the->11.
{\the}
{\the \T}
\the->\catcode 48=12\relax .
{\def \L ->\catcode 65=11\relax \catcode 48=12\relax }

```

There are some advantages in putting items in a box. For instance, if it takes a long time to translate a piece of text that will be used several times, it can save some time. A second possibility is to create a box in a given context and use it in another one (this can be used for instance to put verbatim material in a section title; not in the toc, because the toc is obtained by reading characters from a file, but the box can be used for page headings). Finally, one can put some text

in a box, measure the size of the box, and do some action according to the size of the box; it is not possible to measure a box in `Tralics` because no typesetting is done. Note that there is a limited number of boxes (there is a limit on the number of token registers, but you can always put your token list in a macro; in the same fashion, it is always possible to store integers and dimensions into token lists, i.e., in commands). Note that, if you want to implement arithmetics on big numbers, if you represent a number  $x = \sum x_k B^k$  as a sequence of commands, try to access to  $x_k$  via `\csname x\the\k\endcsname`, and parse this as an integer, then you get something inefficient. It is much more efficient to say `\fontdimen\k\x` (there is a `TeX` file by Denis Roegel that computes thousands of digits of  $\pi$  using font tables as auxiliary memory).

## 2.7 Counters

The most useful registers are counters. Rather than saying `\count16=0`, at the risk of destroying variables used by other packages, you should use named counters, together with an allocation scheme. We have seen that `\newcount\foo` does that. In `LaTeX`, we can do more. If you say `\newcounter{foo}[bar]` then a counter `foo` is defined that depends on `bar`. Let's assume<sup>13</sup>, for simplicity, that the allocation mechanism allocates count register 17. Then `\c@foo` is a reference to `\count17`. It is assumed that no package defines a command that starts with `c@`, or `p@` or `cl@`, so that `\c@foo`, `\cl@foo`, and `\p@foo` are reserved for the counter `foo`. In `LaTeX`, there is a command `\value` that takes one argument and expands to `\csname c@#1\endcsname`. The same command exists in `Tralics`, but it signals an error in the case where `\c@foo` is not a reference to a count register. You can say `\value{foo}=10`, this will put 10 into the counter, you can say `\the\value{foo}`, this will typeset the value of the counter. You should not use this low-level `TeX` syntax. In fact, if you say `\value{foo}=10\the\value{foo}` this will put 103 into the counter (assuming that it contained 3). Compare this with `\parindent=10\parindent` where there is an implicit multiplication.

Assignment should be done via `\setcounter{foo}{10}`. This is the same as `\global\value{foo}=10\relax` (plus a check that `foo` is a counter). The `\relax` has as effect to stop scanning the number. The `\global` makes the assignment global. In the same fashion, `\addtocounter{foo}{4}` is the same as `\global\advance\value{foo}4\relax`. You can say something like `\parindent=\value{foo}\xbar`, this puts in `\parindent` the value of `\xbar` (let's assume it is a dimension) multiplied by the the value of the `foo` counter. If you want to typeset the value of the counter, you say `\number\value{foo}`. You can also use `\romannumeral` or `\Romannumeral` (this last command is not defined by `TeX`) instead of `\number` (it has to be followed by a number, for instance `\value...`). The following commands take as argument the name of a counter, and typeset the value: `\arabic` (it gives 7), `\roman` (it gives vii), `\Roman` (it gives VII), `\alph` (it gives g), `\Alph` (it gives G), `\fnsymbol` (it gives \*\*). The following commands: `\@arabic`, `\@roman`, `\@Roman`, `\@alph`, `\@Alph`, `\@fnsymbol` are used internally by `LaTeX`. They are defined in `Tralics` for compatibility reasons. Hence `\number\value{foo}` is the same as `\@arabic\c@foo` and the same as `\arabic{foo}`; using `\arabic` is the best choice.

Three operations are defined: `\advance` that increments a counter (or a dimension, or a glue), `\multiply` that multiplies it by an integer, and `\divide` that divides it by an integer. In the case of integer division, `TeX` divides the absolute values, and adds the required sign to the quotient (the remainder is not computed). The following piece of code puts in `\count0` the number of hours and in `\count2` the numbers of minutes (quotient of remainder of the division of `\time` by 60).

```
\count0\time
\divide\count0 60
```

<sup>13</sup>In earlier versions, the value returned by the allocator was stored in an internal integer; nowadays, a count register is used (the same as in `LaTeX`). The counter 17 contains the current allocation number for `\newwrite` and should be used with care; the smallest number returned by `\newcount` is at least 40.

```

\count2=-\count0
\multiply\count2 60
\advance\count2 \time

```

You can say `\newlength\foo`. This allocates a new skip register. You can use `\setlength` and `\addtolength`, in the same way as `\setcounter` and `\addtocounter`. However, assignments are local. Using plain T<sub>E</sub>X syntax, you can say:

```

\dimen0=2mm\dimen1=0.2cm
\advance\dimen0 by-\dimen1
\count0=\dimen0

```

Note that `\dimen@`, `\dimen@i`, and `\dimen@ii` are aliases for `\dimen0`, `\dimen0` and `\dimen2`, these quantities are defined but not used by the L<sup>A</sup>T<sub>E</sub>X kernel (but they are used by packages). All registers with number less than ten can be used freely, others should use the allocation mechanism. Example

```

\newcounter{foo}\newlength\lenA\newlength\lenB
\setlength{\lenA}{2mm}
\setlength{\lenB}{0.2cm}
\addtolength\lenA{-\lenB}
\setcounter{foo}{\lenA}

```

After this operation, the counter `foo` contains 5. This means that the difference between 2mm and 0.2cm is 5sp (two thousands of a micrometer). Note: Tralics uses exactly the same algorithms as T<sub>E</sub>X, hence produces the same results.

Appendix A.3.1 of [6] describes the `calc` (package) package. It allows to write commands like that:

```

\newcommand{\printtime}{%
  \setcounter{hours}{\time/60}%
  \setcounter{minutes}{\time-(\value{hours}*60)}
  \thehours h \theminutes min}
\def\today{\ifcase\day\or
  1st\or 2nd\or 3rd\or 4th\or 5th\or
  6th\or 7th\or 8th\or 9th\or 10th\or
  11th\or 12th\or 13th\or 14th\or 15th\or
  16th\or 17th\or 18th\or 19th\or 20th\or
  21st\or 22nd\or 23rd\or 24th\or 25th\or
  26th\or 27th\or 28th\or 29th\or 30th\or
  31st\fi~\ifcase\month\or
  January\or February\or March\or April\or May\or June\or
  July\or August\or September\or October\or November\or
  December\fi\space \number\year}

```

The time is `\printtime`, `\today`.

In this case, the result of Tralics could be: ‘The time is 16h 37min, 7th December 2004.’

You can do operations on integers like this:

```

\newcounter{Ac}
\setcounter{Ac}{(1+2)*(3+4)-20}          %% \theAc=1
\addtocounter{Ac}{(1*2)+(3*-4)+(34/7)}  %% \theAc=-5

```

and on dimensions:

```

\newlength{\Bc}
\setlength{\Bc}{(1cm+2cm)*(3+4)-200mm}  %%\the\Bc=28.4526pt
% exact results should be 1.0pt
\setlength\Bc{\the\Bc*\ratio{25.4pt}{722.7pt}}  %%\the\Bc=0.99985pt

```

```

\Bc=1in \setlength\Bc{\the\Bc * 100 / 7227}           %%\the\Bc=0.99998pt
\Bc=1in \setlength\Bc{\the\Bc * \real{ 0.01383700013837}} %%\the\Bc=1.00018pt
\Bc=1cm \setlength\Bc{\the\Bc / \real{28.452755}}     %%\the\Bc=0.99985pt
\Bc=1cm \setlength\Bc{\the\Bc * \ratio{254pt}{7227pt}} %%\the\Bc=0.99985pt
\Bc=1in \setlength\Bc{\the\Bc / \ratio{7227pt}{100pt}} %%\the\Bc=1.00018pt
\Bc=1IN \setlength\Bc{\the\Bc / \ratio{7227PT}{100PT}} %%\the\Bc=1.00018pt

```

In L<sup>A</sup>T<sub>E</sub>X, there is a command called `\stepcounter`. Its effect is to increment a counter, and reset all counters that depend on it (see example below). There is also `\refstepcounter` whose purpose is to define the current label. This is not implemented in Tralics (see later for how `\label` works). The idea is that, for a counter ‘foo’, the printed value of the label is defined by ‘`\p@foo\thefoo`’. Here `\thefoo` is normally ‘`\arabic{foo}`’, but the quantity can be redefined. For instance, the book class has `\renewcommand\thesection{\thechapter.\@arabic\c@section}` (the article class has no chapter, and does not redefine `\thesection`). Both book and article classes say: `\renewcommand\thesubsection{\thesection.\@arabic\c@subsection}`.

Here we define some counters, and make them depend on other counters.

```

\newcounter{toto}           \setcounter{toto}{10}
\newcounter{titi}[toto]    \setcounter{titi}{20}
\newcounter{tata}[titi]    \setcounter{tata}{30}
\newcounter{tutu}[toto]    \setcounter{tutu}{40}

```

Here we call `\stepcounter`. The typeset result should be 11101=11101.

```

\stepcounter{toto} % kills titi, tutu
\stepcounter{tata} %% \thetata=31,
\stepcounter{titi} %% \thetata=0 % titi=1
\stepcounter{tutu}
\thetoto\thetiti\thetata\thetutu=11101

```

The magic is accomplished by the following command:

```

\def\@addtoreset#1#2{\expandafter\@cons\csname c1@#2\endcsname {#{#1}}}

```

The first argument is the counter to define (for instance ‘tutu’), and the second argument is the dependent counter (for instance ‘toto’). The `\@cons` command is defined like on page 33. It modifies the command `\c1@toto` by adding `\@elt{tutu}`. If you say `\stepcounter{toto}`, then L<sup>A</sup>T<sub>E</sub>X executes ‘`\let \@elt \@stpelt \csname c1@#1\endcsname`’. Here is a part of the transcript file of Tralics that shows what happens (you won’t see the `\csname`, because characters needed for `\c@toto` and `\c1@toto` are read and expanded only once by Tralics.)

```

[720] \stepcounter{toto}
\stepcounter->\global \advance \c@toto 1\relax {\let \@elt \@stpelt \c1@toto }
{\global}
{\global\advance}
+scanint for \c@toto->1
{globally changing \count45=10 into \count45=11}
{\relax}
{begin-group character {}}
+stack: level + 3 for brace entered on line 720
{\let}
{\let \@elt \@stpelt}
{changing \@elt=undefined}
{into \@elt=\@stpelt}
\c1@toto ->\@elt {titi}\@elt {tutu}
\@elt->\global \c@titi 0\relax
{\global}
{\global\c@titi}
+scanint for \c@titi->0

```

```

{globally changing \count46=20 into \count46=0}
{\relax}
\@elt->\global \c@tutu 0\relax
{\global}
{\global\c@tutu}
+scanint for \c@tutu->0
{globally changing \count48=40 into \count48=0}
{\relax}
{end-group character }}
+stack: killing \@elt
+stack: level - 3 for brace from line 720
[721] \stepcounter{tata}
\stepcounter->\global \advance \c@tata 1\relax {\let \@elt \@stpelt \cl@tata }
{\global}
{\global\advance}
+scanint for \c@tata->1
{globally changing \count47=30 into \count47=31}
{\relax}
{begin-group character {}
+stack: level + 3 for brace entered on line 721
{\let}
{\let \@elt \@stpelt}
{changing \@elt=undefined}
{into \@elt=\@stpelt}
\cl@tata ->
{end-group character }}
+stack: killing \@elt
+stack: level - 3 for brace from line 721

```

## 2.8 Fonts

One of the question we can ask is: what does `\it` do? As explained above, this is an unofficial command, thus could be implemented to do anything. Let's assume that it is defined in L<sup>A</sup>T<sub>E</sub>X 2.09 compatibility mode. It is then possible to explain what happens, but it is harder to explain what Tralics should do. A software like `latex2html` (that we studied carefully when implementing the first version of Tralics in Perl) uses a lot of energy in order to translate font changes properly. It is however very difficult to tell it that `\french` is a similar command (in fact, what we wanted is more than just finding the scope of the `\french`, we also wanted French syntax rules to apply, we wanted dashes instead of bullets in lists, etc.). In this paragraph, we shall explain all the gory details concerning fonts (however, look at [6] for what is in a `.fd` file).

One big table in T<sub>E</sub>X is the table of fonts: there are  $N$  fonts with  $N$  characters in them (currently  $N = 256$ , and this is a small limit, in  $\Omega$ , this value is  $2^{16}$ ; the dvi format specifies  $N = 2^{32}$ ). A book like [6] uses lots of fonts indirectly, via inclusion of PostScript files. Note that metric files designed for  $\Omega$  cannot be read by T<sub>E</sub>X. The hyphenation algorithm considers as a word only sequences of characters from the same font (hence 256 characters per font is a hard limit). A metric file contains all that it needed for T<sub>E</sub>X to typeset a character; it does not contain glyphs. Essentially, it contains three tables, indicating for each character its height, its depth and its width. There are two other tables, the lig/kern table, and the kern table, that indicate, for instance in the case VA that some negative space should be used to make the characters narrower, and in the case of fi to use a single glyph instead of two. There is another table (useful only for math mode) that explains how to construct, for instance, braces of various sizes. Finally, there are some parameters. One parameter is the design size (the design size of a ten point font is 10pt), other parameters are the slant, the width of a space (this is glue), the two values of `ex` and `em`,

and extra space. Math fonts have extra parameters, see [4, appendix G]. A font has two integer parameters: hyphen char, and skew char. These values are not in the metric file: when the font is loaded  $\TeX$  uses the values of `\defaultshyphenchar` and `\defaultskewchar`. Note: `Tralics` does not read TFM files, it sets all parameters to zero.

You load a font by saying `\font\preloaded=cmr7 scaled \magstep4` or `\font\foo=cmr10 at 12pt`. Such a construction will read a file `cmr7.tfm` or `cmr10.tfm` and apply a scale factor (a factor 2 in the first case, and 1.2, in the second case). A font like `ecrm` exists in size 5, 6, 7, 8, 9, 10, 10.95 (`magstephalf`), 12, 14.4, 17.28, 20.74, 24.88 (`magstep` 1, 2, 3, 4, and 5 respectively), 29.86 and 35.83. There are some slight differences between `cmr10` at 12pt and `cmr12` (see the  $\TeX$ book for details). You can simply say `\font\tenrm=cmr10`. After that you use it like this `{\tenrm test}`. This gives: test. You can use `\fontdimen1\tenrm` like any dimension. For instance, using `\the` to typeset the value, we get 0.0pt for the slant, 3.33333pt plus 1.66666pt minus 1.11111pt for the interword space, 4.30554pt for the ex-height, 10.00002pt for the quad,<sup>14</sup> 1.11111pt for the extra space. Parameters for the current font are: 0.0pt for the slant, 3.33333pt plus 1.66666pt minus 1.11111pt for the interword space, 4.3055pt for the ex-height, 10.0pt for the quad, 1.11111pt for the extra space. If you say

```
\fontencoding{T1}\fontfamily{cmr}\fontseries{m}%
\fontshape{n}\fontsize{10pt}{12pt}
\selectfont
```

you specify all font parameter, and you switch (from the font named ‘`ec-lmr10`’) to the default ten point font with T1 encoding, namely ‘`ecrm1000`’. The default font in this document uses ‘`lmr`’ as family. The parameters are now: 0.0pt for the slant, 3.33252pt plus 1.66626pt minus 1.11084pt for the interword space, 4.3045pt for the ex-height, 9.99756pt for the quad, 1.11084pt for the extra space. As you can see, they are not exactly the same. However, the glyphs are similar. The current font name can be printed via `\fontname\font`.

The commands shown above are provided by  $\LaTeX$ . The effect of `\selectfont` is to take all values (stored by the other commands) and create a font name (say `\tenrm` for simplicity, see example below for a real name), check the font, and make it the current font. Printing a character like e-acute can depend of the encoding (in some cases the character is in the font, in other cases a combination of two characters is needed). As a consequence, checking the font means to inform some commands of an encoding change. In the example above, the quantity 10pt is the size of the font, but the value 12pt is the baseline skip, changing it means changing some other parameters (for instance the value of `\strut`). An important task of `\selectfont` is to associate to the font name `\tenrm` a real name (say `cmr10`) and call the `\font` command. The real name is computed according to rules defined in a font definition file, for instance `t1cmr.fd`, that depend only on the encoding and family; there are rules that say how to deal with the case where the desired series, shape or size are unavailable. All these commands are implemented in `Tralics`. The size and encoding is currently ignored. We shall describe below some commands that change the series and shape of the current font (for instance `\bfseries`, `\itshape`) that are easily related to parameters of `\selectfont`. Interpreting the argument of `\fontfamily` is a bit more complicated: for instance `pcr` is interpreted as `cmtt` (the name `cmtt` will be explained below, while `pcr` refers to a Courier font). There is another bunch of font commands, implemented in `Tralics`, that provoke an *Unimplemented NFSS command* error; for instance `\DeclareTextAccent` is a command that takes three arguments A, E and N, and says that accent A in encoding E is at position N in the font.

An important characteristic of a font is how glyphs are represented: For  $\TeX$ , this is irrelevant, since the dvi file contains only the metrics. However, the reader will see some black and white pixels (of ink on a sheet of paper, or dots on a screen, or points on a wall projected by a beamer). All fonts designed by Knuth are produced by the `metafont` program that produces both the metrics and

<sup>14</sup>This is a bit more than ten points.

the glyphs as bitmaps (in the form of `gf` file, usually packed as `pk` files). If the resolution of these bitmaps is different from that required by the printing device, some interpolation, extrapolation is required (this is sometimes called ‘antialiasing’, it may involve colored pixels instead of black and white). In general, people print a `dvi` file by converting the first into PostScript format; in a PostScript or pdf file, a font can be specified via different formats, Type1, Type3, TrueType etc. The simplest format is Type3, namely bitmaps. Some software like Acrobat Reader prefer Type1 (a format in which characters are defined by small programs). There is no direct way to produce a Type1 file from a metafont file, so that not all T<sub>E</sub>X fonts exist in Type1. For instance, the computer modern fonts (in version OT1) have been translated but not the T1 version (said otherwise, `cmr10` exists in Type1 format, but not `ecrm1000`). On the other hand, most commercially available fonts are not produced by metafont, hence cannot be used directly by T<sub>E</sub>X. In this document, we experiment the Latin Modern font family; it is very similar to Computer Modern.

In modern distributions, the engine behind L<sup>A</sup>T<sub>E</sub>X is pdfT<sub>E</sub>X, so that producing pdf instead of `dvi` is as easy; in this case, the engine needs the glyphs. Since it is no more restricted to informations found in the metric files, funny effects can be achieved. An extension of T<sub>E</sub>X, called XeT<sub>E</sub>X, produces spectacular results; as in the case of  $\Omega$ , the result can be a variant of the `dvi` format, called `xdv` or `odvi`.

In the case of a format like plain T<sub>E</sub>X, fonts are used according to the following scheme. First you define fonts like `\tenrm`, in three sizes (thus, you define `\sevenrm`, `\fiverm`), and different variants (say `\teni`, `\tensy`, `\tenex`, `\tensl`, `\tentt`, etc). Then you say `\textfont0=\tenrm`, `\scriptfont0=...`, `\scriptscriptfont0=...`: this defines family zero. You do the same for family 1, 2, 3, etc. We shall see later how certain math symbols use a specified family, in other cases the family specified by the `\fam` variable is used (there are only 16 families available). The size of a symbol is defined by the current style (`displaystyle`, `textstyle`, `scriptstyle`, or `scriptscriptstyle`). Then you say `\def\it{\fam4\tenit}`. Thus `\it` has two effects: one is to switch to `\tenit`, the second one is to set `\fam` to 4. Now, you can define a command `\twelvepoint` that modifies all the fonts values, using larger values. Guess what happens for a definition like `\def\it{\tenit\fam4}`.

In the case of a format like L<sup>A</sup>T<sub>E</sub>X, the situation is different. There are some high level commands like `\large`, that are defined like `@setsize \large {14pt} \xiipt\@xiipt` (note: infinite recursion may be possible), and the `\xiipt` command is like the `\twelvepoint` command mentioned above. This is rather complicated. The situation became worse when people tried to replace computer modern fonts by other fonts. We shall describe here only the user interface of the NFSS (new font selection scheme).

There is a clear distinction between `\textit` and `\mathit`: they are to be used in text mode or math mode only; the command `\it` chooses one of them. Guess how `\mathit` is defined. In fact, it switches to some family (the number is not hard-code as the 4 above), to which a font is associated. This may be `OT1/cmr/m/it/10`; an important point is that the size may vary (depending on the current math style of the current font size), but the encoding is fixed: if the current encoding is T1, a different font is used in lath mode and in text mode.

We already mentioned that a important characteristic of the font is the encoding: We met OT1 (Original encoding by Knuth) and T1 (“Cork” encoding, similar to latin 1). There is an obsolete OT2 encoding for cyrillic, and new ones: T2A, T2B, T2C. The companion mentions over twenty standard font encodings. In the example of `\showbox` above, T<sub>E</sub>X told us that the current font was `\T1/cmr/m/n/10`. The first two letters indicate the encoding. There are different families of fonts. Assume that you use Computer Modern fonts (you do this by selecting a package; after that, your whole document will be in computer modern, unless you use fonts selected via `\font` or `\selectfont`). There are six sub-families: Roman, Sans, Typewriter, Fibonacci, Funny roman, and Dunhill. The name of these families are: `cmr`, `cmss`, `cmnt`, `cmfib`, `cmfr`, `cmdh`. The default family in this document is `cmr`. You can chose another family via the commands `\rmfamily`, `\ttfamily` and `\sffamily` (no command is provided for the other families). The commands `\textrm`, `\textsf` and `\texttt` take an argument and typeset it using the family. The commands

`\rm`, `\sf`, `\tt` do the same, but they reset the series to medium, and the shape to normal. The series of a font can be: bold, bold extended, semibold, etc. In L<sup>A</sup>T<sub>E</sub>X you have `\mdseries` and `\bfseries` (you have also `\textmd` and `\textbf`, which are commands that take an argument; you have also `\bf` that selects roman family, bold series, normal shape). The shape can be: normal, italic, slanted, upright italic, small caps, etc. In L<sup>A</sup>T<sub>E</sub>X we have `\upshape`, `\itshape`, `\slshape`, and `\scshape` (and as, usual, `\textup`, `\textit`, `\textsl` and `\textsc`; there is also `\it`, `\sl`, `\sc`). There are two commands `\em` (a declaration) and `\emph` (that takes an argument) that use upright shape if the current font has a slant, and italics shape otherwise. These rules explain the `cmr/m/n` part in the font. In fact, the ‘`cmr`’ part comes from the command `\rmdefault`, but these commands are not implemented in Tralics. The command `\textnormal` takes an argument as is the equivalent of `\normalfont`.

There are two parameters that define the size of the font. First, document class options indicate the size used by `\normalsize`. In our example it is 10pt. There are ten commands that change the font size. In increasing order they are `\tiny`, `\scriptsize`, `\footnotesize`, `\small`, `\normalsize`, `\large`, `\Large`, `\LARGE`, `\huge`, and `\Huge`. There is a command `\selectfont`; its purpose is to combine everything, the result will be `\T1/cmr/m/n/10`. There is another process that converts this to the font name `ecrm1000`, using font definition files.

In math formulas, you see things like  $\alpha'$  and  $\ddot{e}$ , but never ‘á’ and ‘ë’. If you want an acute accent you use `\acute`, if you want a double dot accent you say `\ddot`. In fact, the textfont used for math is very often a 7bit font, without accented letters. If you want  $x^{\grave{e}me}$  you should say `x{\grave{e}me}`, or perhaps `x{\hbox{\grave{e}me}}` (this gives  $x^{\grave{e}me}$ , letters are too big). Note that Tralics may translate this as  $x^e$ ; if you do not like it, either set the `notrivialmath` counter to zero, or an empty group in the formula before the hat. A solution is `x\textsuperscript{ième}`, `xième`. In French, you say  $1^{er}$ ,  $1^{re}$ ,  $1^{ers}$ ,  $1^{res}$ ,  $2^e$ ,  $3^{es}$ , etc., via `1\ier`, `1\iere`, `1\iers`, `1\ieres`, `2\ieme`, `3\iemes`. In English, you say  $1^{st}$ ,  $2^{nd}$ ,  $3^{rd}$ ,  $4^{th}$ .

## 2.9 Spaces

In T<sub>E</sub>X spaces are ignored after a command like `\foo`, and a sequence of spaces is treated as a single one. The exact rule is the following. There is a variable whose values can be N (start of line), or M (middle of line) or S (when spaces are skipped). Whenever a line is read, T<sub>E</sub>X removes every space character at the end of the line. It inserts the value of `\endlinechar` (provided this is a valid character, an integer between 0 and 255). The state is N. Spaces are ignored if the state is S or N; if the state is M, a space produces a space token, and the state is changed to S; in this sentence a “space” is any character whose category code is 10. If T<sub>E</sub>X sees an end-of-line character (category 5), it ignores all other characters on the current line. If the state is N (line was empty), the tokeniser returns a `\par` token, if the state is M it returns a space token, otherwise the character is ignored. Note: in Tralics, the space token produced by an end-of-line is a line-feed character, this is to keep line breaks in the XML translation. If T<sub>E</sub>X sees a backslash (or any character of category code 0), it reads a command; the state will be S if the character is a letter or a space, it will be N otherwise. If T<sub>E</sub>X sees anything else, the state will be M.

For instance, if you say ‘`x{ }{ }{ }{ }{ }y`’ the tokeniser sees 5 spaces. If you say `\def\A{ }` and `\def\B{ \A\A\A }`, then the body of `\A` contains a space as well as the body of `\B`. Full expansion of `\B` contains three spaces and `x\By` contains four spaces. The command `\space` is defined like `\A` above.

Spaces discarded by the tokeniser do not appear in the translation. However, spaces produced by the tokeniser can be ignored in some cases. A typical example: a command can take a space as argument, and ignore the argument. For instance `\` is a command that ignores spaces that follow it using explicit scanning (i.e. `\futurelet`). We already mentioned that spaces between arguments are generally ignored. Spaces can be ignored because you say `\ignorespaces`: the



effect of this command is to expand what follows, until a non-expandable token is seen. If it is a space, it is ignored, and the process continues. A space can be ignored because of a syntax rule (for instance, before an equals sign in an assignment). In L<sup>A</sup>T<sub>E</sub>X you can see things like that `\end{x}` `\end{y}` `\end{z}`, each ‘end(xxx)’ being on a line by itself: this produces a space, and the L<sup>A</sup>T<sub>E</sub>X environment mechanism is clever enough to remove these spurious spaces. It is also possible to remove a space from typeset material via `\unskip`.

Spaces are ignored in math mode. The reason is that spaces are used to separate words, and there are no words in math formulas. There are operators, and these operators know how much white space to use. In the case of  $x + y = z$ , on each side of the plus sign there is some glue, the value comes from `\medmuskip`, it is 2.22 plus 1.11 minus 2.22; on each side of the equals sign there is `\thickmuskip`, namely 2.77 plus 2.77 (the unit is pt).<sup>15</sup> After the zed, there is a kern of value 0.4398. Note: the plus sign is followed by a penalty of 700, the equals sign by a penalty of 500. Plain T<sub>E</sub>X defines

```
\thinmuskip=3mu
\medmuskip=4mu plus 2mu minus 4mu
\thickmuskip=5mu plus 5mu
```

In Tralics, constant values are used (expressed in terms of em units; one em is 18mu, in the example above one em is 10pt). You can say `\:`, `\>` and `\;`. This produces a space (thin, medium, thick) using the values given above. You can also use `\!`, this is the negative of thin space. The translation of `$A\B\C\D\!E$` is:

```
<mrow><mi>A</mi><mspace width='0.166667em' />
  <mi>B</mi><mspace width='0.222222em' />
  <mi>C</mi><mspace width='0.277778em' />
  <mi>D</mi><mspace width='-0.166667em' /><mi>E</mi></mrow>
```

The `\space` command expands to a single space token. It may disappear in all cases where the syntax says that a space is optional (because in general these rules imply expansion); in a case like `\let\foo\space`, tokens are not expanded, and `\foo` is made equivalent to the current value of `\space`. The `\_` command cannot be expanded. It starts a paragraph (if used in vertical mode). It inserts some white space whose value is the same as if the current space factor were 1000. You can use it after an abbreviation like `Mr.` in order to indicate that the dot is not an end of sentence marker. You can also use it after a command like `\TeX` if you want to leave some space. In math mode, Tralics interprets it as a space of width 6pt. The `~` character is usually active, its expansion is `\nobreakspace`. This is defined in Tralics to translate to `&nbsp;`; . You can say `\quad` or `\qqquad`. This inserts some space (the width is one or two em). If you say `\hskip 1cm`, this will append some glue (in Tralics, it will generate a sequence of `&nbsp;`; whose width is more or less 1cm). Note: in the current version, entity names are no more generated, hence `&nbsp;`; is replaced by the Unicode character U+A0, and we assume that the width of this character is one forth of a quad. In math mode, both the tilde character and `\nobreakspace` will give 3.33pt; inside an URL, the result is a tilde character. If you say `\kern 1cm` this will append a kern (like glue, but the size is fixed). This is ignored by Tralics. A normal space produces glue (the value of the glue depends on some font parameters; it can also depend on the current space factor). A glue may disappear at a line break. Kerns will not. In L<sup>A</sup>T<sub>E</sub>X, you use `\hspace` instead of `\hskip`. You can use `\hspace*`, in this case, spaces at start of line are not ignored. Note the syntax `\hspace{2cm}` vs `\hskip2cm\relax`.

```
A\space\space B\ \ C\quad\qqquad etc
a\hskip2cm b\hspace{3cm}etc.
x\vskip2cm\vspace{2cm}etc.
```

Translation is (we have replaced nobreak space by tilde)

<sup>15</sup>these values were rounded; exact values are 2, 4 and 5 times 10pt/18, and 10pt/18 is a mu.

```

<p>A B C~~~~~etc
a~~~~~b~~~~~etc.
x</p>
<p spacebefore='56.9055pt'>y</p>
<p spacebefore='56.9055pt'>etc.
</p>

```

When  $\text{\TeX}$  wants to split a paragraph into lines of equal width, it will have to stretch and shrink the glue that appears on the line; it will remove interword glue at break points. An item of glue has the form  $x + y - z$ , where  $x$ ,  $y$  and  $z$  are dimensions ( $y$  and  $z$  can be expressed in terms of fil, fill and filll), all three values can be positive or negative. We can express this as: we have a vector of size 9:  $x_0$  is the regular part of the glue,  $x_1$ ,  $x_2$ ,  $x_3$  and  $x_4$  are the stretch component (in units of pt, fil, fill, and filll, only one of these components can be given),  $x_5$ ,  $x_6$ ,  $x_7$  and  $x_8$  are the shrink components (in units of pt, fil, fill, and filll, only one of these components can be given). When two pieces of glue are added, all components are added. The convention is that  $x_2$  is much larger than  $x_1$ , so that the sum of  $x_1$  and  $x_2$  is  $x_2$  (said otherwise if we add `1pt plus 2pt` and `3pt plus 4fil`, the result is `4pt plus 4fil`). Such simplifications are not done when  $\text{\TeX}$  computes the sum of all glue items in a paragraph (as a result, addition is associative). The command `\hfil` is equivalent to `\hskip0pt plus 1fil`, the command `\hfill` is equivalent to `\hskip 0pt plus 1fill`, the command `\hfilneg` is equivalent to `\hskip 0pt plus -1fil`, the command `\hss` is equivalent to `\hskip 0pt plus 1fil`. It is an error to use infinite shrinkage, like `\hss`, in a paragraph,  $\text{\TeX}$  complains with: *Infinite glue shrinkage found in a paragraph*. However you can say `123\hbox to1cm{\hss xxxxxx\hss}456`, the result is `123xxxxx456`, said otherwise, the text is centered, no overfull neither underfull box is signaled.

The commands `\vfil`, `\vfill`, `\vfilneg`, `\vss`, behave in the same fashion, in vertical mode, adding vertical space. `\Tralics` translates `\hfil`, `\hfill`, `\hfilneg`, and `\hss` as `\leavevmode` followed by an element `<hfil>`, that has the same name as the command. It translates `\vfil`, `\vfill`, `\vfilneg`, and `\vss` in the same fashion, by using `\par` instead of `\leavevmode`. The three commands `\bigskip`, `\medskip` and `\smallskip` are used to insert vertical space between paragraphs, of size 12pt, 6pt and 3pt respectively (in  $\text{\LaTeX}$ , this is some glue that the user can modify, however, `\Tralics` ignores the shrink and stretch parts of the glue inserted by `\hskip`, `\vskip`, `\hspace` and `\vspace`.) These four commands read an argument (in  $\text{\LaTeX}$ , `\hspace` and `\vspace` accept an optional star, that translates to an empty vertical or horizontal rule, `\Tralics` ignores the star). In the case of an horizontal space, `\leavevmode` is executed, then `~` are produced (one every 4 pt, a negative dimension produces nothing). In the case of a vertical space, the current paragraph is terminated; if after that the mode is vertical, a new paragraph is started, it has an attribute `spacebefore` with as value the dimension. In  $\text{\LaTeX}$ , the behavior is different (see appendix A.1.5 of [6]). In math mode, you can also use `\mskip` and `\mkern`, these command use  $\mu$  as unit, where 18 $\mu$  is one em. Since `\Tralics` does not know the value of an em, it uses 10pt, so that the dimension is first divided by 18, then multiplied by 10. Example

```

a\vfil\vfill\vfilneg\vss
b\hfil\hfill\hfilneg\hss
c\bigskip d\smallskip e\medskip f
$\mskip3\mu\mkern2\mu \mskip 18\mu$

```

Translation is

```

<p>a</p>
<vfil/><vfill/><vfilneg/><vss/>
<p>b<hfil/><hfill/><hfilneg/><hss/>c</p>
<p spacebefore='12.0pt'>d</p>
<p spacebefore='3.0pt'>e</p>
<p spacebefore='6.0pt'>f
<formula type='inline'><math xmlns='http://www.w3.org/1998/Math/MathML'>

```

```

<mrow><mspace width='1.66656pt' /><mspace width='1.111pt' />
<mspace width='10.0pt' /></mrow></math>
</formula>
</p>

```

In T<sub>E</sub>X, there is no command that starts a paragraph. The `\leavevmode` command is implemented as `\unhbox\voidb@x`, where `\unhbox` starts a new paragraph if needed, and produces nothing, provided that its argument is the void box; the paragraph may contain the current indentation and the value of `\everypar`. This is a primitive in Tralics, the value of `\everypar` is unused. Both commands `\indent` and `\noindent` make sure the current mode is horizontal, the first one inserts the current indentation (an empty box with the width of `\parindent`). In T<sub>E</sub>X, you can use `\indent` anywhere in a paragraph. In Tralics, the translation of

```

a\noindent b \indent c
{\centering a\noindent b \indent c\par d}
{\raggedright a\noindent b \par\indent c\par d}

```

is

```

<p>a</p>
<p noindent='true'>b</p>
<p rend='center' noindent='false'>c
a</p>
<p rend='center'>b</p>
<p rend='center'>c</p>
<p rend='center'>d
a</p>
<p noindent='true' rend='flushed-left'>b</p>
<p noindent='false' rend='flushed-left'>c</p>
<p rend='flushed-left'>d</p>

```

The rules are the following: if `\indent` or `\noindent` appear in an empty paragraph, that is not centered, and that has no `noindent` attribute, one is set. Otherwise a new paragraph is started. It will have a `noindent` attribute, unless the paragraph is centered. The value of `\parindent` is never considered.

The translation of `\par` is a bit complicated. Nothing happens inside a `\hbox`, in `\term`<sup>16</sup>, or if the current mode is not horizontal. The current XML element should be a `<p>`. A final space is removed from it. It will be popped from the stack. This restores the mode to the value of the previous mode. It restores the current XML element to the parent of the `<p>`. A newline character is added to it. There is an exception: in cases like `\noindent\par`, or `\bigskip\par`, or `\!\par`, the `\par` command was ignored until version 2.5 (pl7). The behavior is now: if the paragraph is empty, but there are attributes, then the `<p>` is removed, and attributes are added to the next `<p>` element.

The translation of `\!` depends on the context. The command can be followed by an optional star, and an optional dimension. Inside a cell, this indicates the end of the cell as well as the the end of the row. You can say `\newline`, this is like `\!` without optional argument and array test. In vertical mode, L<sup>A</sup>T<sub>E</sub>X complains with *There's no line here to end*, but Tralics ignores the command. Inside a title, the command is ignored. Otherwise, the behavior is like `\noindent`; if an optional argument is given, it behaves like `\vskip`. For instance, the translation of

```

a \!b \![2cm] c \newline[3cm]d \noindent e \vskip 4cm f

```

is

<sup>16</sup>There is no `\term` command in Tralics. Until version 2.11.6, the `\par` command was ignored if the current stack frame is named 'term', in other words inside the `motcle` environment.

```

<p>a</p>
<p noindent='true'>b</p>
<p noindent='true' spacebefore='56.9055pt'> c</p>
<p noindent='true'>[3cm]d</p>
<p noindent='true'>e</p>
<p spacebefore='113.81102pt'>f</p>

```

Many people do not know that `\` takes an optional argument, and try to use different tricks in order to avoid errors triggered by `\\`. We have seen for instance

```
\def\ligne{\protect{\mbox{}}\mbox{} \indent}}
```

Remember that `\protect` is like `\noexpand`, it is not a  $\LaTeX$  command that takes an argument! More strange cases can be found in [3].

The commands `\nolinebreak`, `\nopagebreak`, `\pagebreak`, and `\linebreak` are defined by  $\LaTeX$  to take an optional argument, an integer between 0 and 4. They insert some penalty, but depend on the mode, like `\hspace` and `\vspace`. They are ignored in *Tralics*. The command `\break`, `\nobreak`, and `\allowbreak`, are defined by  $\LaTeX$ , they insert some penalty (zero, plus or minus infinity). They are ignored by *Tralics*. The commands `\fillbreak`, `\goodbreak`, `\eject`, `\smallbreak`, `\medbreak`, `\bigbreak` are defined by  $\LaTeX$  to terminate a paragraph and insert some penalty. In *Tralics*, they behave like `\par`. Note. The last chapter of the second part of this document explains that, when converting XML to Pdf, special rules must be used when hyphenating URLs: ambiguities can be avoided when text is split slashes. For this reason, *Tralics* inserts a `<allowbreak>` element in these cases, and when the command `\allowbreak` is used as well.

## 2.10 Conditional expansion

In the previous paragraphs we have shown how to define a macro `\foo` that expands to `\bar` and a macro `\bar` that expands to `gee`.<sup>17</sup> Can a translator replace all `\foo` by `\bar` and all `\bar` by `gee`? the answer is obviously no; first because, if you say `\something\bar`, the argument will be (after expansion) `gee`, while in the case of `\something gee` it will be `g`; there is a second problem, that occurs in `latex2html`: if you replace `\bar` by its value, you get `\somethinggee`, and this is wrong, if you reparse it<sup>18</sup>; some commands can be randomly redefined (for instance, at first use) like this:

```
\def\NFSS{NFSS (New Font Selection Scheme)\global\def\NFSS{NFSS}}
```

The last reason is conditional expansion. Our original translator (written in Perl) has some troubles in these cases.

In this section, we shall consider cases where expansion depends on the context. We have already seen the commands `\noexpand` for delayed execution and `\expandafter` that changes the order of expansion, in section 6.12 we will describe `\protected` which inhibits expansion in a `\edef`. We shall analyze three commands: `\Color`, `\Map` and `\Loop`.

<sup>17</sup>It is unwise to say `\def\bar`, because `\bar` is already defined for use in math mode, and who knows which commands rely on this definition.

<sup>18</sup>The remedy is to add a space after the command name. The trouble is when the command results from partial expansion of an active character, case where spaces should be preserved. This is one reason why active characters cannot be made robust in  $\LaTeX$ . The package `fixtx2e` corrects a bug of this type for the expansion of `\@`: a space disappears if the command is written on a file and reread.

### 2.10.1 Constructing commands dynamically

Using colors in T<sub>E</sub>X is not completely trivial, one reason is that there are different color models, more or less adapted to the task (printing on paper, on transparencies, or using a video projector). The color package proposes

```
\def\textcolor#1#{\@textcolor{#1}}
\def\@textcolor#1#2#3{\protect\leavevmode{\color#1{#2}#3}}
```

Note that the brace character that indicates the start of the body of `\textcolor` is preceded by a sharp sign. This means that the argument of the command is everything before the brace. In a case of `\textcolor {green} {text}`, it is empty. The `\color` command takes two arguments (the color model, empty in the example, and the color); it changes the current color, which is magically restored at the end of the group. One of the reasons why colors are not implemented in Tralics is also the scope of the command is unclear. Assume that we have two commands ‘`\enrouge`’ and ‘`\envert`’ that take an argument and typeset it in red and green; they could be defined as

```
\def\enrouge#1{\textcolor{red}{#1}}
\def\envert#1{\textcolor{green}{#1}}
```

We explain here how to solve the following problem. We want to define a command `\Color` that takes two arguments, a color and text; if the color is ‘rouge’ or ‘vert’ it should call `\enrouge` or `\envert`. Otherwise, some default action is specified (an error could be signaled, in the following, we assume that the color should be ignored). One solution to this problem uses tests, as explain in the next section. This means that we have to change the macro if a new color (for instance ‘`\enbleu`’ for blue) is added to the list. The following works

```
\def\color#1{\csname en#1\endcsname}
```

The only drawback with this method is that it might produce unexpected results in the case where the command defined by `\csname` already exists (try ‘`\color{d}{document}`’).

There are many commands that use `\csname`. The problem mentioned above can be avoided if the command contains a non-letter character. For instance, when the counter `foo` is defined, the command `\p@foo` is created, and this command is used whenever the counter is printed. No package should define commands starting with `p@`. In some cases the construction can be

```
\csname\string\color @#1\endcsname
```

This constructs a command with a backslash in its name, and can be created only via `\csname`, thus offers a good protection.

### 2.10.2 Iterating over lists

In this paragraph we explain how to apply a command to all items in a list. The list could be defines as follows

```
\def\mylista{\do{A1}\do{B2}\do{C3}}
\def\mylistb{{A1}{B2}{C3}}
\def\mylistc{A1,B2,C3}
```

The last line is an example of CVS (comma separated values). The L<sup>A</sup>T<sub>E</sub>X command `\@for` can be used to apply a command to every item, and `\@tfor` should be used in the second case. Here is an example.

```
\makeatletter
\let\BreakTfor@break@tfor\let\Lfor\@for
\makeatother
\def\List{ }\def\thelist{12,3,4,5,6} % list is expanded here
\Lfor\Elt:=\thelist\do{\edef\List{\List\Elt}\if\Elt4\BreakTfor\fi}
```

We give here the transcript file produced by Tralics. The same algorithm is used as in  $\text{\LaTeX}$ . Arguments of  $\text{\@for}$  are respectively an element name, the colon-equal separator, the list to work on (it will be expanded), the  $\text{\do}$ -separator, and the code to be applied. On lines 4 and 5 you see the expansion: there is a call to  $\text{\@forloop}$ , taking as arguments the expanded list where two dummy items have been added, the end marker  $\text{\@@}$ , the element name and the code. The command is optimised in the case where the list is empty, or has a single element; in the general case, you will see assignment of  $\text{\Elt}$  (lines 7-8) and expansion on lines 9, 10, 11. Note that  $\text{\@iforloop}$  is used; you can see on lines 36, 37, 38 the expansion of  $\text{\@iforloop}$ , which is a simple recursive function. Other assignment of  $\text{\Elt}$  can be seen on lines 22 and 34. On lines 47 and 48 you can see the expansion of  $\text{\@breaktfor}$ . What you do not see is that this command gobbles all tokens inserted by  $\text{\@for}$  and friends (namely, everything up to the  $\text{\@@}$  token, the element name and the code). Caveat: the expansion of the  $\text{\LaTeX}$  command with the same name is a double  $\text{\fi}$ .

```

1 [6] \Lfor\Elt:=\thelist\do{\edef\List{\List\Elt}\if\Elt4\BreakTfor\fi}
2 {\@for}
3 \thelist ->12,3,4,5,6
4 \Lfor<- \@forloop 12,3,4,5,6,\@nil ,\@nil \@@ \Elt {\edef \List {\List \Elt }
5 \if \Elt 4\BreakTfor \fi }
6 {\@forloop}
7 {changing \Elt=undefined}
8 {into \Elt=macro:->12}
9 \@forloop<- \edef \List {\List \Elt }\if \Elt 4\BreakTfor \fi \def \Elt {3}
10 \edef \List {\List \Elt }\if \Elt 4\BreakTfor \fi \@iforloop 4,5,6,\@nil
11 ,\@nil \@@ \Elt {\edef \List {\List \Elt }\if \Elt 4\BreakTfor \fi }
12 {\edef}
13 \List ->
14 \Elt ->12
15 {changing \List=macro:->}
16 {into \List=macro:->12}
17 +\if1
18 \Elt ->12
19 +iftest1 false
20 +\fi1
21 {\def}
22 {changing \Elt=macro:->12}
23 {into \Elt=macro:->3}
24 {\edef}
25 \List ->12
26 \Elt ->3
27 {changing \List=macro:->12}
28 {into \List=macro:->123}
29 +\if2
30 \Elt ->3
31 +iftest2 false
32 +\fi2
33 {\@iforloop}
34 {changing \Elt=macro:->3}
35 {into \Elt=macro:->4}
36 \@iforloop<- \edef \List {\List \Elt }\if \Elt 4\BreakTfor \fi \relax
37 \@iforloop 5,6,\@nil ,\@nil \@@ \Elt {\edef \List {\List \Elt }\if
38 \Elt 4\BreakTfor \fi }
39 {\edef}
40 \List ->123
41 \Elt ->4
42 {changing \List=macro:->123}
43 {into \List=macro:->1234}

```

```

44 +\if3
45 \Elt ->4
46 +iftest3 true
47 {\@break@tfor}
48 \BreakTfor<- \fi
49 +\fi3

```

### 2.10.3 Mapping a command

We consider here the following task. We have a list, like `\mylista` above, and we want to apply a command, say `\foo` to every element of the list. The solution we propose here is faster than the previous one; remember that, in the case of `\@for`, for every element, the unread part of the list, together with five additional tokens, the element name and the body, all these tokens are read, and pushed back in the stream. Our solution is as simple as

```

\def\Map{\let\do}
\def\Map#1#2{\let\do#1#2}
\def\Map#1#2{\def\do{#1}#2}

```

Then we say `\Map\textit\mylista`. This produces *A1B2C3*. This is however a bit unsatisfactory: in some cases the list delimiter is different from `\do`, an example is given above: at the start of a chapter, we want to reset all counters that depend on the chapter counter, in this case `\@elt` is used as delimiter. We could imagine a map-with-argument macro, that would take as argument the `\do`. But this is nothing else than `\let`! Our definition is so simple that people just say ‘`\let\do\@makeother\dospecials`’, see for instance 2.12. The difference between the first two versions of `\Map` is that the second command takes arguments, hence removes an additional level of braces. If you say `\Map\foo{\do{A}}`, the command `\foo` is executed in a group in the first case (and it is a mistake to put braces around it).

In the third case, the first argument `#1` can consist in more than one token. For instance, if you say `\def\foo#1#2{\_#1#2}` then `\Map{\foo A}\mylista` gives ‘AA1 AB2 AC3’. Note that there are too many spaces in this example: the last space in `\foo` is spurious.

### 2.10.4 Creating a list via pattern matching

Consider

```

% \newcommand{\fooiv}[3][bar]{Seen #1 #2 #3}
\def\fooivaux[#1]#2#3{Seen #1 #2 #3}
\def\fooiv{\@ifnextchar[{\fooivaux}{\fooivaux[bar]}}

```

The commented line is interpreted by L<sup>A</sup>T<sub>E</sub>X in the same fashion as the two other lines (except that the internal name is a bit more complicated than ‘`\fooivaux`’). We shall explain later how ‘`\@ifnextchar`’ works<sup>19</sup>. We are interested here in how L<sup>A</sup>T<sub>E</sub>X converts the ‘`[3]`’ into ‘`[#1]#2#3`’. Since the number of arguments is between zero and nine, a short sequence of conditionals could be used. Instead, the following code is used by L<sup>A</sup>T<sub>E</sub>X:

```

1 \long \def \@yargdef #1#2#3{%
2   \ifx#2\tw@
3     \def\reserved@b##11{####1}}%
4   \else
5     \let\reserved@b\@gobble
6   \fi
7   \expandafter

```

<sup>19</sup>The actual code uses `\kernel@ifnextchar`, because `amsmath` redefines sometimes `\@ifnextchar`

```

8     \@yargd@f \expandafter{\number #3}#1%
9   }
10  \long \def \@yargd@f#1#2{%
11    \def \reserved@a ##1##2##{%
12      \expandafter\def\expandafter#2\reserved@b ##1#1%
13      }%
14    \l@ngrel@x \reserved@a 0##1##2##3##4##5##6##7##8##9##1%
15  }

```

In the case of ‘`\newcommand\fooooo[3]{foo}`’ the `\@yargdef` command is called with three arguments, the first is `\fooooo`, the command to be defined, then comes ‘`\@ne`’ (some randomly chosen token), then ‘`3`’ (the number of arguments) and finally ‘`{foo}`’, the body of the command to be defined. This argument is not read, but the code relies on the fact that it starts with an opening brace. The objective is to produce ‘`#1#2#3`’. In the case of ‘`\newcommand\fooi{foo}`’, arguments are the same with 0 as third argument, the objective is to produce the empty string. In the case of `\fooiiv`, the second argument is ‘`\tw@`’, this is something different from ‘`\@ne`’, the objective is similar, but a bit different: we want ‘`[#1]#2#3`’.

In order to make things easier to understand, we shall proceed to the following simplifications: let’s forget about the percent signs (their purpose is to suppress unwanted space). Let’s forget about ‘`\long`’ (is it really needed?) and ‘`\l@ngrel@x`’ (this is something that adds conditionally a ‘`\long`’ token before the definition). Let’s simplify the names: we write ‘`\Ra`’ and ‘`\Rb`’ instead of ‘`\reserved@a`’ and ‘`\reserved@b`’. We also write ‘`\ydef`’ and ‘`\yaux`’ instead of ‘`\@yargdef`’ and ‘`\@yargd@f`’. Finally, we replace the arguments by X, Y, Z, and ‘`##`’ by a simple ‘`#`’. Hence we get

```

\def \ydef XYZ{
  \ifx Y\tw@
    \def\Rb#11{[#1]}
  \else
    \let\Rb@gobble
  \fi
  \expandafter \yaux \expandafter{\number Z}X
}
\def \yaux XY{
  \def\Ra #1X#2#{\expandafter\def\expandafter Y\Rb #1X}
  \Ra 0#1#2#3#4#5#6#7#8#9#X
}

```

Let’s start the analysis with the lines 7 and 8. Because of the two ‘`\expandafter`’ tokens, the first token to be expanded is ‘`\number`’. This means that Z is replaced by its numeric value. Said otherwise, the number of arguments can be ‘`03`’, or ‘`\^^C`’, or even ‘`\value{page}`’ if the page number is not too big. In Tralics, only explicit numbers are allowed (You will get a message like *Only one token allowed; I will assume that the command takes no argument.*) In general, lines 7 and 8 are equivalent to `\yaux{Z}X`.

Let’s now explain lines 2 to 6. We are in a simple case of a conditional (the commands `\@ne` and `\tw@` are normally equivalent to 1 and 2, they compare unequal), so that line 3 is executed in case of an optional argument, and line 5 otherwise. In the last case `\Rb` is a command that takes an argument and ignores it; otherwise `\Rb` is a command that takes an argument, delimited by the character ‘`1`’, ignores it, and the expansion is ‘`[#1]`’ (four tokens). Remember that we want ‘`[#1]#2#3`’, that is a good starting point.

Consider now lines 11 and 12. In order to simplify explanations, we replace X by Z and Y by X (i.e. use the argument names of the outer function). We shall denote by U and V the arguments of `\Ra`. Thus `\Ra` is

```
\def\Ra UZV{\expandafter\def\expandafter X\Rb UZ}
```



The question now is what are the values of U and V? In order to answer this question we shall write line 14 in a different way. Let  $s(n)$  be the sequence  $\#1\#2\dots\#n\#$ , and  $S(n)$  the sequence  $\#n\dots\#9$ . The content of line 14

```
\Ra 0#1#2#3#4#5#6#7#8#9#Z
```

can be interpreted as  $\backslash\text{Ra}$ , 0,  $s(n-1)$ ,  $n$ ,  $S(n+1)$ ,  $\#Z$ , whenever  $n$  is a digit between 1 and 9. Said otherwise, whenever  $Z$  is a digit between 1 and 9, the first argument U of  $\backslash\text{Ra}$  is  $0s(n-1)$  (the second argument is ignored, it is everything up to the first brace, the one that delimits the body). Obviously, in the case where  $Z$  is the digit 0, U is empty. We leave it as an exercise to the reader to see what happens in the case where  $Z$  is a sharp sign<sup>20</sup>. In all other cases, U is the sequence  $0s(9)$ . The important point is that, whatever  $Z$ , T<sub>E</sub>X will not read beyond the opening brace of the body.

Assume now that we want to construct a normal command (case  $\backslash\text{Rb}$  is gobble). It always gobbles a zero (if  $Z$  is zero, U is empty, and  $Z$  is gobbled). Thus  $\backslash\text{Rb UZ}$  expand to: nothing if  $Z$  is 0,  $s(n-1)n$  if  $Z$  is a digit between 1 and 9, and  $\#1\#2\#3\#4\#5\#6\#7\#8\#9\#Z$  otherwise. This yields an error *You already have nine parameters* which is adequate in case  $Z$  is a number larger than nine. Consider now the case of an optional argument. Here  $\backslash\text{Rb}$  is a bit different: it reads the ‘0#1’ part and replaces it by ‘[#1]’. You will get a *Runaway argument?* error (or some other strange behavior) in case  $Z$  is ‘0’ because pattern matching fails (of course, you should never try to make optional the first argument of a function that takes none).

## 2.10.5 A variant of the previous problem

In the previous paragraph we have shown how to convert an integer, say 3, into a sequence  $\#1\#2\#3$ . One trouble with sharp signs is that you have to double them, and if you define a command in the body of the other one, they must be doubled again. Thus we state our problem as: given an integer  $N$  between 1 and 9, construct  $\backslash\text{sharp1}\backslash\text{sharp2}\dots\backslash\text{sharpN}$ . After that, we can evaluate the ‘ $\backslash\text{sharp}$ ’ command<sup>21</sup>, replacing it by ‘#’. One solution (original L<sup>A</sup>T<sub>E</sub>X code) uses a loop from  $N$  down to  $Y$  (with  $Y = 1$  in the case of normal argument,  $Y = 2$  otherwise<sup>22</sup>). Some variants will be discussed later on. The current L<sup>A</sup>T<sub>E</sub>X code uses pattern matching, as explained above, this leads to the following solution

```
\def\ydef#1#2#3{%
  \def\tmp##1#2##20{##1#2}%
  \def\Sharp{\noexpand\Sharp}%
  \edef\etmp{\tmp\Sharp1\Sharp2\Sharp3\Sharp4\Sharp5\Sharp6\Sharp7%
    \Sharp8\Sharp90}%
  \def\Sharp{#####}% needs 8 #
  \edef\etmp{\etmp}%
  \expandafter\def\expandafter#1\etmp{#3}}%

\ydef{\acmd}{3}{\string \acmd\space called with #1, #2, and #3.}
Test \acmd A{BC}D.

Test \acmd called with A, BC, and D..
```

<sup>20</sup>Since  $Z$  is the result of expansion of  $\backslash\text{number}$  it starts with a digit or a minus sign, so that this case will not happen.

<sup>21</sup>There is a  $\backslash\text{sharp}$  command, that typesets as #, hence be careful.

<sup>22</sup>This explains the names  $\backslash\text{@ne}$  and  $\backslash\text{tw@}$ .

## 2.10.6 Loops

A silly question is: can we do loops without conditionals? The answer will be given later. We assume here that our loop will be of the form: while  $N$  is not too big, do something and increment  $N$ . This mechanism needs modifying a table (the location of  $N$ ) hence is not pure expansion. In our example, we will write `\sharp\the\count0`, and hope that this will evaluate to `#3` later on, assuming that `\count0` contains `3` now. How that can be implemented is left as an exercise to the reader. See also section 2.11.1. We shall explain later all the silly details concerning conditionals in  $\TeX$ , all we need to know is that you can test  $a < b$  and  $a > b$ , but neither  $a \leq b$  nor  $a \geq b$ . Here is our code:

```
\def\code{\advance\count0 by 1 \sharp\the\count0}
\def\Loop{\ifnum\count0<\count1 \code\Loop\fi}
```

Assume that `\count0` holds 0 and `\count1` holds 3. In this case the test is true, `\code` is evaluated, then `\Loop`. The effect of evaluating `\code` is to increment the counter and produce `\sharp1`. The loop terminates after `\sharp3` has been produced. Notice that recursion is not terminal (but it would be in most computer languages): when the test is found false, there are four `\if` tokens not yet evaluated. This example is atypical, in that the counter is modified before its use; exchanging the `\sharp...` and the `\advance...` part implies changing initial and final value (1,4 instead of 0,3).

Our `\Loop` command is not generic, in that the name of the counters are built-in. Thus Knuth proposes the following:

```
\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body \let\next\iterate \else\let\next\relax\fi \next}
\let\repeat\fi % this makes \loop...\if...\repeat skippable
```

```
\loop \ifnum \count0<\count1 \sharp \the\count0 \advance\count0 by 1\repeat
```

Note that the last line contains an `\ifxx` where the associated `\fi` is the `\repeat` at the end of the line. Thus, in the case where the `\loop` command is *not expanded* this line is well-balanced regarding conditionals. In the case where `\loop` *is expanded*, the value of the `\repeat` token is irrelevant, it just serves as delimiter, and the `\fi` has to be found in `\iterate`. In order for `\iterate` to work, the `\body` should expand to an incomplete conditional, without `\else` part. It conditionally sets `\next`, and evaluates it after `\fi`; this trick makes the recursion terminal.

An alternate version is given by  $\LaTeX$ , as follows<sup>23</sup>

```
\def\loop#1\repeat{\def\iterate{#1\relax\expandafter\iterate\fi}%
\iterate \let\iterate\relax}
```

Adding `\let\iterate\relax` at the end of the definition has no real importance; but it causes no harm either. Note the `\expandafter` trick: if the test in the loop is false, neither `\expandafter` nor `\iterate` are expanded, if the test is true, `\fi` is evaluated before `\iterate`. Thus recursion is terminal. One difference with the  $\TeX$  method is that the body of the loop is put in `\iterate` rather than in a auxiliary command. The interesting point is the `\relax`. Guess what happens in this case:

```
\def\bloop#1\repeat{\def\iterate{#1\expandafter\iterate\fi}\iterate}
\count0=0 \count1=4
\bloop \ifnum \count0<\count1 \the\count0 \advance\count0 by 1\repeat
```

If you use  $\LaTeX$  in verbose mode, you can see that the test is true, true and false, where you expect it to be true four times. The printed result is `0` (hence the question: what did the second iteration do?). Using `\Tralics`, you will get more information.

<sup>23</sup>The kernel contains two versions, one of them being `\long`.

```

1 +\ifnum6
2 +scanint for \count->0
3 +scanint for \ifnum->0
4 +scanint for \count->1
5 +scanint for \ifnum->4
6 +iftest6 true
7 {\the}
8 {\the \count}
9 +scanint for \count->0
10 \the->0.
11 Character sequence: 0.
12 {\advance}
13 +scanint for \count->0
14 {\expandafter \iterate \fi}
15 +\fi6
16 \iterate ->\ifnum \count 0<\count 1 \the \count 0 \advance \count 0 by 1\expandafter \...
17 +\ifnum7
18 +scanint for \count->0
19 +scanint for \ifnum->0
20 +scanint for \count->1
21 +scanint for \ifnum->4
22 +iftest7 true
23 {\the}
24 {\the \count}
25 +scanint for \count->0
26 \the->0.
27 +scanint for \count->10
28 {\advance}
29 +scanint for \count->0
30 {\expandafter \iterate \fi}
31 +\fi7
32 \iterate ->\ifnum \count 0<\count 1 \the \count 0 \advance \count 0 by 1\expandafter \...
33 +\ifnum8
34 +scanint for \count->0
35 +scanint for \ifnum->10
36 +scanint for \count->1
37 +scanint for \ifnum->4
38 +iftest8 false
39 +\fi8

```

Lines 16 and 32 are a bit too long; there are two token `\iterate\fi` that are replaced by `\...`

As you can see, all tests have a serial number. On lines 2–5, you can see why the first test is true: it is because the numbers 0 and 4 are compared. On lines 18–21, you see why the second test is true, and on lines 34–37, you see why the last test is false; in fact, `\count0` contains ten. On line 27, you see something strange. Explanations: Assume that you say ‘`\advance \Foo4`’, where `\Foo` is a reference to some counter. In this case, the trace of Tralics will contain `+scanint for \Foo->4`, and everybody understands this. If you replace `\Foo` by `\count0`, the trace will contain `\count`; it will also contain a line for the zero in `\count0`. Hence, the number that appears in line 27 is the value read by the `\advance` in line 12. What happened is the following: after ‘by’ we have seen the digit ‘1’. In the case of `\loop`, the next token would be ‘`\relax`’, and this stops scanning of the number. But here, we have ‘`\expandafter`’, which is expandable and expanded, as a consequence, this finishes the first conditional. After that comes the test; it is true, because we did not increment our counter yet. Then comes ‘`\the`’ which is expandable. This reads ‘`\count0`’, as well as the space after it. The expansion of ‘`\the...`’ is the digit zero; so far, we have read 10, and continue reading. The next token is ‘`\advance`’ and this is not expandable. Hence `\advance`

has read everything up to the next `\advance`. Is it needed to explain what happens next? Let's just notice that, at line 39, `Tralics` (and also `TEX`) are still reading tokens for the second `\advance`. Since version 2.9, `Tralics` prints an additional line, between line 27 and 28, of the form `\count0` changed from 0 to 10.

## 2.11 Conditionals in T<sub>E</sub>X

We shall discuss in this section the following commands

- `\if AB` (comparison of two characters, character code),
- `\ifcat AB` (comparison of two characters, category code),
- `\ifint A=B` (comparison of integers),
- `\ifdim A=B` (comparison of dimensions),
- `\ifvmode`, `\ihmode`, `\ifmmode`, `\ifinner` (test of the current mode),
- `\ifvoid A`, `\ifhbox A`, `\ifvbox A` (test box register status),
- `\ifx AB` (comparison of two tokens),
- `\ifeof N` (test if characters can be read from a channel),
- `\iftrue`, `\iffalse` (constant tests),
- `\ifcase` (this is called ‘switch’ in some other languages),
- `\ifdefined \cmd` (check if `\cmd` is defined),
- `\ifcsname string \endcsname` (check if `string` is the name of a command),
- `\iffontchar \font chr` (check if `chr` exists in font),
- `\or`, `\else`, `\fi` (must be used after if),
- `\unless` (prefix negating the result).

### 2.11.1 Syntax of the conditionals

A conditional has the form `\if test true-code \else false-code \fi`. The `\else` part is optional; conditionals can be nested, and this nesting is independent from anything else. The command `\unless` (provided by  $\epsilon$ -T<sub>E</sub>X) can be used before the if-like command (except `\ifcase`), its effect is to reverse the truth value of the test. Conditionals are expanded: this means that conditionals are evaluated inside a `\edef`, you can use `\noexpand` to delay evaluation, and `\expandafter` to change the order of expansion.

An important point is the following: if you define a command `\ifthenelse` with three arguments, that evaluates the first argument as a boolean, and expands conditionally to the second or third argument, then these two arguments must be balanced, and category codes are fixed. In the case of `\if`, there is no such limitations: if the test is found false, then all tokens are read at high speed until finding a `\else`, and normal processing occurs, or until finding a `\fi`, that indicates the end of the conditional; if the test is true, and if there is an `\else` part, all tokens between `\else` and `\fi` are read at high speed. Consider for instance this piece of code

```
\ifnum \A=\B do-nothing \else {\let\fi\relax\C}\fi
```

Assume that the test is false; this means that the else part is evaluated. Locally `\fi` is redefined to do nothing, and `\C` is evaluated. Let's assume that `\C` does nothing special (it could typeset ‘Hello, world!’). In this case the `\fi` after the brace terminates the conditional. Assume now that the test is true. Skipping over the `\else` part at high speed just means compare the actual value

of a token with ‘\if’ or ‘\fi’: in the first case, the if-counter is incremented, in the second case it is decremented, in all other cases the counter is left unchanged; reading stops when the counter is zero. Here, the conditional is terminated by the first ‘\fi’. This means that you have to be very careful: the end of the conditional can change, depending on whether the test is true or false. When we say: ‘compare the actual value of the token’, this means that the name is irrelevant, only the meaning is used, for instance ‘\repeat’ has the same value as ‘\fi’, and `\loop... \if... \repeat` is well balanced.

All constructions indicated above have a then-part and an else-part, except `\ifcase`: this command reads a number (see section 2.6 for details) and you can specify action for the case zero, the case one, the case two, using `\or` as separator, and an optional `\else` for other cases. Any other use of the `\or` command will signal a *Extra \or* error. For instance, we can solve the problem of constructing `\sharp1... \sharp N` as follows (assuming ‘\N’ holds the value of *N*)

```
\ifcase \N \error{You cannot use zero here}
\or \sharp1
\or \sharp1\sharp2
\or \sharp1\sharp2\sharp3
\or \sharp1\sharp2\sharp3\sharp4
\or \sharp1\sharp2\sharp3\sharp4\sharp5
\or \sharp1\sharp2\sharp3\sharp4\sharp5\sharp6
\or \sharp1\sharp2\sharp3\sharp4\sharp5\sharp6\sharp7
\or \sharp1\sharp2\sharp3\sharp4\sharp5\sharp6\sharp7\sharp8
\or \sharp1\sharp2\sharp3\sharp4\sharp5\sharp6\sharp7\sharp8\sharp9
\else \error{Argument must be non-negative, at most nine}
\fi
```

The simple conditional ‘\if AB ... \else ... \fi’ compares two characters A and B, it shares some features with `\ifcat`. It expands tokens, using the following rules

- In the case `\noexpand\cmd`, where `\cmd` is a command that be expanded, it is as if you had said `\relax`.
- In the case `\noexpand X`, where X is an active character, the token that will be considered is X, without expansion.
- In the case `\noexpand Y`, that does not match any of the previous cases, the `\noexpand` is discarded.
- In the case `\cmd`, where `\cmd` can be expanded, it will be expanded.
- In the case X, where X is an active character, it will be expanded.
- In the case `\prim`, where `\prim` is a primitive that cannot be expanded, it is as if you have given character of numeric code `-1`, and category code `-1`.
- In the case of an implicit character `\cmd` (one obtained via `\let\cmd Y`, where Y is a character), the numeric code and the category code of the character is considered.
- In the case of an explicit character Y, the numeric code and the category code of the character is considered.

The command `\if` compares the two numeric codes, and `\ifcat` compares the category codes. If you say something like

```
\catcode ‘\A=3
\def\fooi{A}
\catcode ‘\A=11
\def\fooui{A}
\if\fooi\fooui H\fi \ifcat\fooi\fooui\else e\fi
```

```

\if\bgroup{l\fi \ifcat\egroup}l\fi \if\relax\par o\fi
\if01\else,\fi \ifcat01 w\fi \if\par1\else o\fi
\if\noexpand\fooi\relax r\fi \if\fooi Ald\fi \if!!!\fi

```

this should typeset as ‘Hello, world!’.

You must be very careful using a construction like ‘\if\A\B...’, because of the following

- If \A takes no argument, expands to ‘foo’, the test is false, whatever \B.
- If \A is defined by ‘\def\A\B{00}’, the test is true, whatever \B.
- If \A is defined by ‘\def\A\C{ }’, an error is signaled.
- If \A is a command that takes an argument, the argument will be ‘\B’, and what happens depends on what the command does with the argument.
- If ‘\A’ expands to ‘xxyy’, the test is true, whatever \B, and a part of \A is typeset before ‘\B’.
- If ‘\A’ expands to ‘xyyx’, the test is false, ‘\B’ is ignored.
- If ‘\A’ and ‘\B’ expand to nothing, tokens that follow will be examined.
- If ‘\A’ expands to nothing, ‘\if\A\B...’ behaves like ‘\if\B...’, see above.

Plain T<sub>E</sub>X provides an \outer macro<sup>24</sup> \newif that takes an argument \iffoo (whose name starts with the two letters if) and makes it a new conditional; the ifthen package provides the more L<sup>A</sup>T<sub>E</sub>Xish syntax \newboolean{foo}. This means that \iffoo true-code \else false-code \fi becomes valid, and evaluates false-code. You can say ‘\footrue’ and the condition becomes true (true-code is evaluated) or ‘\foofalse’ and it becomes false (false-code is evaluated). The \global prefix is allowed before the command. The ifthen package provides \setboolean{foo}{true} where the second argument is case insensitive. These commands could be implemented as

```

\def\footrue{\def\iffoo{\if00}}
\def\foofalse{\def\iffoo{\if01}}

```

The trouble with this definition is that, when ‘\iffoo’ is read at high speed, it is not recognized as a conditional (it is a user defined command), see discussion about ‘\ifhph’ in [4, Chapter 20]. For this reason, the commands \iftrue and \iffalse were added to T<sub>E</sub>X, they evaluate respectively to true and false, and the following lines work (because \let is used instead of \def):

```

\def\footrue{\let\iffoo\iftrue}
\def\foofalse{\let\iffoo\iffalse}

```

You can use ‘\ifnum’ or ‘\ifdim’: in both cases a numeric quantity, an operator, and another numerical quantity are read. Three operators are recognized: less than, greater than and equal to. In the case of ‘\ifnum’, both quantities have to be numbers, otherwise dimensions. Note that glue is converted to a dimension (and possibly a number), by ignoring the shrink and stretch part. If you want to compare two items of glue, you must split them into components and check them in order. The example that follows shows also that math glue must first be converted into ordinary glue. All the commands shown here are fully expandable; without the \relax, this piece of code gives three errors (and T<sub>E</sub>X is still trying to see if the ‘fill’ is not a ‘fill!’).

```

\muskip0=36mu plus 18mu minus 1fill\relax
\ifnum\glueshrinkorder\mutoglue\muskip0=2\else\bad\fi
\ifdim\glueshrink\mutoglue\muskip0=1pt\else\bad\fi
\ifdim\gluestretch\mutoglue\muskip0=18pt\else\bad\fi

```

Here is an example that uses no extension.

```

\count0=0 \count1=1 \dimen0=1pc \dimen1=12pt
\skip0=1cm minus3fill \skip1=1mmplus 2fill

```

<sup>24</sup>There is no clear reason why this command should be \outer, hence L<sup>A</sup>T<sub>E</sub>X uses a normal macro instead

```

\tracingall
\ifnum \count0<\count1
  \ifdim \dimen0=\dimen1
    \ifdim \skip0>\skip1 ok \fi\fi\fi

```

This is the trace of Tralics. Note that for L<sup>A</sup>T<sub>E</sub>X, all lengths allocated by `\newlength` are “rubber” length, i.e. associated to a `\skip` register. Such quantities are automatically converted into rigid length (however, if you replace in the example ‘`\skip1`’ by ‘`1mmplus 2fill`’, then only a rigid dimension is read, the ‘`plus 2fill`’ is not part of the condition).

```

+\ifnum26
+scanint for \count->0
+scanint for \ifnum->0
+scanint for \count->1
+scanint for \ifnum->1
+tiftest26 true
+\ifdim27
+scanint for \dimen->0
+scandimen for \ifdim->12.0pt
+scanint for \dimen->1
+scandimen for \ifdim->12.0pt
+tiftest27 true
+\ifdim28
+scanint for \skip->0
+scandimen for \ifdim->28.45274pt
+scanint for \skip->1
+scandimen for \ifdim->2.84526pt
+tiftest28 true
+\fi28
+\fi27
+\fi26

```

This is one solution to our problem of producing N sharp signs in a row:

```

\ifnum \N>0 \sharp1\fi\ifnum \N>1 \sharp2\fi\ifnum \N>2 \sharp3\fi
\ifnum \N>3 \sharp4\fi\ifnum \N>4 \sharp5\fi\ifnum \N>5 \sharp6\fi
\ifnum \N>6 \sharp7\fi\ifnum \N>7 \sharp8\fi\ifnum \N>8 \sharp9\fi

```

The following construction is a priori more efficient (on the average there are less tests) but it takes more memory.

```

\ifnum \N>0 \sharp1\ifnum \N>1 \sharp2\ifnum \N>2 \sharp3%
\ifnum \N>3 \sharp4\ifnum \N>4 \sharp5\ifnum \N>5 \sharp6%
\ifnum \N>6 \sharp7\ifnum \N>7 \sharp8\ifnum \N>8 \sharp9%
\fi\fi\fi\fi\fi\fi\fi\fi\fi

```

You can test whether a character can be read from an input channel, via the `\ifeof` command. Here is an example from the Tralics torture file. The file `tortureaux.tex` has six lines, the first one contains `abc`, the second one is empty, the third one contains `\a \b {\c`, the fourth one contains `{} \d} \e`, the next one contains `123`, the last one is empty. The `\testeq` commands compares two commands: things should be equal here. (See T<sub>E</sub>Xbook, exercise 20-18, if you do not understand the setting of `\endlinechar`). Commands starting with ‘bad’ are not evaluated in this example. Details can be found in section 5.12.

```

{
\openin 5=tortureaux
\endlinechar=-1
\ifeof5 \badifeofatentry\fi
\read 5 to \foo\testeq\foo{abc}
\read 5 to \foo\testeq\foo{}

```

```

\read 5 to \foo\testeq\foo{\a\b{c{ } \d} \e}
\global\read 5 to \foo
\closein5\relax
\ifeof5\else\badifeofatexit\fi
}\testeq\foo{123}
\ifeof3\else \badifeofnonexists\fi

```

You can say `\ifvoid25`, `\ifhbox25` or `\ifvbox25`. In  $\TeX$  these command would test the content of box register 25: if empty, the `\ifvoid` is true, the other tests are false; if not empty, the box contains a horizontal list or a vertical list, and `\ifhbox` and `\ifvbox` are respectively true, the two other tests being false. In `Tralics`, a box contains a character string or an XML element, but there is no associated orientation; hence `\ifhbox` and `\ifvbox` always evaluate to false. Instead of 25, any number can be given (provided it is a valid register number) In the example that follows, only the first equals sign is part of an assignment, and box number one is tested.

```

\count0=1
\ifvoid\count0=2\fi
\ifvbox\count0=3\fi
\ifhbox\count0=4\fi

```

You can say `\ifmmode`, `\ifvmode`, `\ifhmode` and `\ifinner`. These commands check the current mode. The first three evaluate to true if the mode is math mode, vertical mode, or horizontal mode. The last is true if the mode is inner (internal vertical mode, restricted horizontal mode, or (nondisplay) math mode). The following example shows these modes.

```

\def\wm{\edef\res{\ifinner i\else I\fi
\ifhmode h\else H\fi
\ifvmode v\else V\fi
\ifmmode m\else M\fi}\res}
\par \wm$$\wm \hbox{\wm $\wm$} \eqno \wm$$

```

The result is: ‘IHvM IHVm ihVM iHVm iHVm’. If you remove the ‘`\edef`’, the trouble will be that typesetting the ‘I’ enters horizontal mode. This example fails if ‘`$$...$$`’ is replaced by ‘`\[...]`’, because `\eqno` switches to inner math mode, and ‘`\]`’ checks for outer math. The same test provokes an error in `Tralics`, because of the implementation of `\eqno`, that expands all tokens, including the token that follows `\edef`. `Tralics` knows whether it is in or out of math mode; in math mode it knows whether it is in display math or not. In these cases, it produces the same result as  $\TeX$ . Otherwise `\ifinner` is false, and `\ifvmode` or `\ifhmode` produce results in accordance to the current mode, that has little to do with  $\TeX$  modes.

An extension of  $\epsilon$ - $\TeX$  is `\isdefined`. This reads a token, and yields true unless it is a macro (or active character) that is undefined. The command `\ifcsname` reads all characters up to `\endcsname` and constructs a character string in the same way as `\csname`. The value is true if a command with that name exists (possibly undefined); it is false otherwise (the important point is that the command is not created). In the example that follows, assuming `\foo` and `\FOO` undefined, you will see aBc (or abc, in case someone dedfined `\undefined`). You will also see DEF, because the  $\LaTeX$  command `\@ifundefined` creates the token if it deos not exists, and sets it to `\relax`.

```

\makeatletter
\ifcsname foo\endcsname A\else a\fi
\ifx\foo\undefined B\else b\fi
\ifdefined\foo C\else c\fi
\@ifundefined{FOO}{D}{d}
\ifcsname FOO\endcsname E\else e\fi
\ifdefined\FOO F\else f\fi

```



The command `\iffontchar` is another extension; it reads a font identifier (for instance `\font` denotes the current font) and an integer (a character position); it yields true if the font specifies a character at that position.

The last conditional to explain is `\ifx`. This reads two tokens and compares them. Two tokens are equal if they are character tokens (implicit or explicit) with same character value and category code, or two T<sub>E</sub>X primitives with the same meaning, or two user-defined commands with the same value (same arguments, same body, same `\long` and `\outer` flags)<sup>25</sup>,<sup>26</sup>.

### 2.11.2 Examples of conditional commands

Using `\ifx` we can code our `\Color` command properly, like that

```
\def\Color#1#2{%
  \def\crouge{rouge}\def\cvert{vert}\def\cc{#1}%
  \ifx\cc\crouge\enrouge{#2}\else\ifx\cc\cvert\envert{#2}\else#2\fi\fi}
```

It is possible to avoid these assignments in the `\Color` macro, provided that they are hidden elsewhere. For instance

```
\def\ifstringeq#1#2#3#4{%
  \def\tempa{#1}\def\tempb{#2}%
  \ifx\tempa\tempb#3\else#4\fi}

\def\Color#1#2{%
  \ifstringeq{#1}{rouge}{\enrouge{#2}}
  {\ifstringeq{#1}{vert}{\envert{#2}}{#2}}}
```

Note that the `ifthen` package provides the `\equal` command as helper for such a situation: you could say `\ifthenelse{\equal{A}{B}}{X}{Y}` instead of `\ifstringeq {A}{B}{X}{Y}`. Caveat: the `\equal` command fully expands its two arguments, our version expands nothing.

In any computer language, you would define a command that compares two strings and returns true or false; this is not possible in T<sub>E</sub>X because commands return no value. All you can do is modify some variable (a command, a register, a token list, etc). This assignment can be done by the caller or the callee. Here is a solution where the token `\next` is set by the caller:

```
\def\Color#1{%
  \ifstringeq{#1}{rouge}{\let\next\enrouge}
  {\ifstringeq{#1}{vert}{\let\next\envert}{\let\next\relax}}%
  \next}
```

Note that, if `\envert` accepts an optional argument, for instance if `\envert[clair]{text}` typesets the text using light green, you can say `\Color{vert}[clair]{text}`. We consider now a case where the assignment is done by the callee (via `\equaltrue` or `\equalfalse`; there is a variant that uses `\setboolean`).

```
\newif\ifequal
\def\streq#1#2{%
  \def\tempa{#1}\def\tempb{#2}%
  %%variant: \setboolean{equal}{\ifx\tempa\tempb true\else false\fi}
  \ifx\tempa\tempb\equaltrue\else\equalfalse\fi}

\def\Color#1{%
  \streq{#1}{rouge}%
```

<sup>25</sup>If you remember the code of `\fooviv`, you can see that two L<sup>A</sup>T<sub>E</sub>X commands with optional arguments always compare unequal.

<sup>26</sup>Note that  $\epsilon$ -T<sub>E</sub>X adds a third flag, `\protected` that is also compared.

```

\ifequal\let\next\enrouge\else
  \streq{#1}{vert}%
  \ifequal\let\next\envert\else \let\next\relax\fi\fi
\next}

```

A subtlety of T<sub>E</sub>X is that tokens are read only when needed. Said otherwise, if you say ‘\if AB C\else D\fi’, T<sub>E</sub>X will evaluate the test; it will remember that a new conditional has started. If the test is false, it will skip at high speed until the \else, and resume normal evaluation; but if the test is true, it will resume normal evaluation right now. It is only when T<sub>E</sub>X sees an \else token (and this can be another one) that it will read all tokens at high speed until the \end. And, when T<sub>E</sub>X sees the \fi, it will pop the conditional stack. Consider the following example:

```

\def\ifstringeq#1#2#3#4{%
  \def\tempa{#1}\def\tempb{#2}%
  \ifx\tempa\tempb\aux{#3}\else\aux{#4}\fi}
\def\aux#1#2\fi{\fi#1}
\def\color#1{%
  \ifstringeq{#1}{rouge}{\enrouge}{\ifstringeq{#1}{vert}{\envert}{\relax}}}

```

Assume that the test is true. Then \aux reads all tokens, up to ‘\fi’, provides a \fi to finish the conditional now, then expands to its first argument (which is argument 3 of \ifstringeq). In the case where the test is false, the same thing happens. This is nicer than the solution that consists in defining conditionally \next and evaluating it after the \fi, it avoids an assignment.

### 2.11.3 Testing the next token

Let’s consider now a variant of the color problem. We want to write a command with three arguments A, B and C, it is assumed to read a token, compare it with A, and expand to B or C. We need an auxiliary command that reads the token. Thus the solution

```

\def\ifnextchar#1#2#3{%
  \let\tempa=#1\def\tempb{#2}\def\tempc{#3}%
  \ifaux
}
\def\ifaux#1{%
  \let\lettoken=#1%
  \ifx\lettoken\tempa\let\tempd\tempb\else\let\tempd\tempc\fi
  \tempd
}

```

Note that we have put an equals sign after ‘\let\tempa’ and ‘\let\lettoken’ for the case where the token to match is an equals sign. If you want to catch spaces, a bit more complicated machinery must be used. There is a problem with this command, because, if the argument of \ifaux is not a single token, say ‘foobar’, then only ‘f’ will be put in \lettoken and ‘oobar’ will be typeset. On the other hand, if the argument is empty, then ‘\ifx’ will be put in \lettoken; after that \lettoken will be expanded. Since this is \ifx, the following tokens will be compared (said otherwise ‘\tempa’ and ‘\let’), this is not exactly what is required. In order to solve this problem, we first modify slightly our code:

```

\def\ifnextchar#1#2#3{%
  \let\tempa=#1\def\tempb{#2}\def\tempc{#3}%
  \ifaux
}
\def\ifaux#1{\let\lettoken=#1\ifnch}
\def\ifnch{%
  \ifx\lettoken\tempa\let\tempd\tempb\else\let\tempd\tempc\fi
}

```

```

    \tempd
  }

```

The `\ifnch` command given above looks like the L<sup>A</sup>T<sub>E</sub>X version of the beast. In fact, spaces are ignored in L<sup>A</sup>T<sub>E</sub>X, so that there is an additional test. Moreover, some variables have a different name, nevertheless, here is the code:

```

\def\@ifnch{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifnch
  \else
    \ifx\@let@token\reserved@d
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}

```

The problem is the `\ifaux` command. The question is: can we rewrite it in such a way as to read a single token, before calling `\ifnch`. Recall that we want to distinguish between `{x}` and `x`. A very interesting question is the following: if we read the opening brace, how can we put it back in the input stream? we cannot do so by just expanding a macro (because the body is always well balanced). You could try something like `{\ifnum0=} \fi` (that leaves an unmatched brace after expansion), or something like `{\iffalse} \fi`. Our solution is much simpler. There is a T<sub>E</sub>X primitive that gets the token without reading it. To be precise, `\futurelet` reads a token A, that has to be a command name or an active character, then a second token B, then a third token C. The value of the token is put in A, using the equivalent of `\let`, then C and B are pushed back in the input stream (in this order, the token to be read first is B). The code of `\ifnextchar` is hence the following:

```

\def\ifnextchar#1#2#3{%
  \let\tempa=#1\def\tempb{#2}\def\tempc{#3}%
  \futurelet\lettoken\ifnch}

```

What `{\futurelet\lettoken\ifnch}` does is read a token. This could be a space character, an equal sign, an open brace, a closing brace, whatever. It puts it back in the input stream. It puts it also in `\lettoken`. After that, it evaluates `\ifnch` (which is a command that should take no argument, of course; it should consult `\lettoken` and depending on the value, call a command that, maybe, reads the token). There are some variants. For instance `amsmath` has a version that omits the comparison with `<@sptoken>`. The `xkeyval` package provides a version where the category codes of the character to test and the actual token may be different.

#### 2.11.4 Reading a space

We consider in this paragraph the following problem: is it possible to define a command `\sptoken` that behaves like a space character inside `\ifx`? One problem with the current version of Tralics is that, as has been mentioned earlier, a newline character in the source file produces a new line character in the XML file; thus has a different representation as a normal space. Thus, there are two different space tokens N and S (they have the same category code, but a different value, 13 or 32). If a macro requires an argument delimited by a space, both these characters can be used. When comparing token lists, these tokens are considered equal. However, when using `\ifx`, these two tokens compare unequal. Our purpose is to create `\sptoken` that compares equal to S; it is trivial to create the N token, and compare them.

We give here three solutions. The first one uses `\futurelet`. If the arguments are A, B and C, where A is the command to define, and C the space, then B has to be a command (if it is a

character, it will be typeset); this cannot be `\foo`, since spaces after `\foo` disappear, it has to be something like `\;`. This command must read the space, otherwise it appears in the output. We provide two solutions: a command that is delimited by a space, and a command that takes an argument (remember that spaces disappear before undelimited arguments):

```
\def\; {}\futurelet\SPtoken\; % comment required
\def\;#1{}\futurelet\SPtoken\; 0
```

In both cases, the command `\;` cannot be used for typesetting (in the  $\text{\LaTeX}$  kernel, it is used for computing the `\SPtoken`, and correctly redefined after that). We give here an example, where the redefinition is temporary, inside the box. We can discard the content of the box.

```
\setbox0\hbox{\def\;{}\global\futurelet\SPtoken\; }
```

We give now a solution using `\let`. Remember the syntax, after `\let` and `\sptoken` (the token to be assigned), comes `\langle equals \rangle` and `\langle one optional space \rangle` and `\langle token \rangle`, where the last token is our space token. Since `\langle equals \rangle` reads an arbitrary number of spaces and an optional equals sign, an equals sign is required. Our optional space cannot be optional. So we must produce a double space. This is not completely trivial. We give here two solutions (the comment is necessary)

```
\def\makesptoken#1{\let\sptoken= #1}\makesptoken{ }
\def\;{\let\SPtoken= } \; % this makes \SPtoken a space token
```

And now, how can we define `\@xifnch`? this command is assumed to read a space, discard it, and check again for the next character. Thus the question is to design a macro that reads a space. This cannot be done via `\def\@xifnch#1...`, since spaces are ignored before undelimited arguments; we cannot use the technique of the command `\;` above, because we cannot read what follows the space; the solution consists in a command that takes no argument, and that starts with a compulsory token, like `\def\foo\bar{etc}`. The non trivial point is that we want `\bar` to be replaced by a space token, but spaces disappear after `\foo`. We give here two solutions.

```
\expandafter\def\expandafter\foo\space{etc}
\def\;{\Foo}\expandafter\def\; {etc}
```

### 2.11.5 Variants of the Map problem

Let's consider the following variant of the `\Map` command. If we have `\do{A}\do{B}\do{C}`, we want to separate arguments with a comma, and put a period after the last argument; we might as well do something with the argument, say, typeset it in italics. This is not always possible. In one of the style sheets used by the Raweb, a Perl postprocessor is used for replacing some commas by a period. We assume here that we know where the list ends. For instance, we assume that we can put a `\endl` token at the end of the list. Then we can write something like

```
\def\foo#1#2\endl{\textit{#1}\ifx#2\endl\endl.\else, \foo#2\endl\fi}
```

Then `\foo{A}{B and C}{D}\endl` produces `'A, B and C, D.'` as expected. Let's analyze the code and try to see why it is wrong. We assume that you never say `\foo\endl`, because the list is assumed non-empty. We also assume that the list does not contain the `\endl` token (in  $\text{\LaTeX}$ , you should use the special marker `\@nil` only as list delimiter). In our case, the first argument is `'A'`, the second is `'{B and C}{D}'`. In the case where the second argument is empty, the test is true, because `\endl` is compared against itself. In our case, the test is false because the brace is compared with the letter B. If we put the second argument in a pair of braces, we get an error: *Too many }'s*, because the test is true, and a part of `'#2\endl\endl'` has been evaluated. This means that our test is wrong. The only safe way to check whether `#2` is empty is to put it in a command, and check whether this is the same as `\empty`. We shall give a second version of the code where the test is replaced by `\ifx\endl#2\endl`. In the case where `#2` is empty, the test evaluates to true, and if `#2` evaluates to some token list that does not start with `\endl`, the test will be false; this is better.

Note that, when `\foo` is called again, it compares ‘D’ with ‘`\endl`’. Does this surprise you? In fact, if you say ‘`\foo{A}{XY}{UV}\endl`’, you get ‘A, XY, U, V.’. The trouble is the following: when T<sub>E</sub>X reads the arguments of a command, a pair of braces disappears, when possible. Thus arguments are ‘A’ (without braces) and ‘`{XY}{UV}`’ (it is not possible to remove the braces). When `\foo` is called again, arguments are ‘XY’ and ‘UV’, without braces. This explains why the test compares U and V (by the way, if ‘UV’ is replaced by ‘UUVV’, the test will be true, yielding an *Undefined control sequence* error). When `\foo` is called again, arguments are now ‘U’ and ‘V’, an unwanted result. There is a simple way to avoid disappearance of braces: it suffices to put a token before each item, for instance like this

```
\def\foo\do#1#2\endl{\textit{#1}\ifx\endl#2\endl.\else, \foo#2\endl\fi}
\foo\do{A}\do{B}\do{C}\endl
```

The good way of testing that the argument is empty is to use `\@ifempty`, which has different syntax:

```
\def\foo\do#1#2\endl{\textit{#1}\@ifempty{#2}{.}{, \foo#2\endl}}
\foo\do{A}\do{B}\do{C}\endl
```

A more elegant solution: notice that #2 starts with `\do`, unless it is empty. There is no need to read the argument for seeing this, we can use the `\ifnextchar` command. With the solution proposed here, the token that marks the end of the list is evaluated: we use `\relax`, because this is harmless.

```
\def\foo{\def\do##1{\textit{##1}\@ifnextchar{\do}{, }{.}}
\foo\do{A}\do{B}\do{C}\relax
```

Note that we can replace `\relax` by something more useful, for instance a period:

```
\def\foo{\def\do##1{\textit{##1}\@ifnextchar{\do}{, }{.}}
\foo\do{A}\do{B}\do{C}.
```

An alternate solution could use ‘`\ifprevchar`’ instead of ‘`\ifnextchar`’. There is no such command in L<sup>A</sup>T<sub>E</sub>X, but the idea is the following: instead of putting a comma after each argument but the last, we can put a comma *before* each argument but the first. All we need to do is to know if this argument is the first. In one application, we have coded this as: apply `\do-first` on the first argument, and map `\do-other` on the rest of the list. If side effects are allowed, we can use a piece of code like this (note how the final period is typeset):

```
\newif\iffirst
\def\do#1{\iffirst\firstfalse\else, \fi\textit{#1}}
\firsttrue
\do{A}\do{B}\do{C}.
```

In fact, there is no need to use an auxiliary command, it suffices to modify `\do` itself:

```
\def\foo{\def\do##1{\textit{##1}\def\do###1{, \textit{###1}}}}
\foo\do{A}\do{B}\do{C}.
```

If you think that there are too many sharp signs, you can try

```
\newcommand\normaldo[1]{, \textit{#1}}
\newcommand\firstdo[1]{\textit{#1}\let\do\normaldo}
\newcommand\foo{\let\do\firstdo}
\foo\do{A}\do{B}\do{C}.
```

There are other possibilities implying conditional commands. We shall see later how to define a `comment` environment that ignores the content of it. It is as if you said

```
\newenvironment{comment}{\iffalse}\fi
```

One can make the following strange construct `{\ifnum0=‘}\fi`. In this case, we compare two numbers, zero and the internal code of the brace (which is in general non-zero). The result of the test is false, but who cares? the body of the conditional as well as the else part is empty. Hence,

the result is like `\bgroup`, there are some differences because  $\TeX$  has two brace counters: the balance counter and the master counter; there is only one counter in `Tralics`. For details, see the  $\TeX$ book and its appendix D, where it is said “If you understand [...] you’ll agree that the present appendix deserves its name.” (the name of the appendix is ‘Dirty Tricks’).

A piece of code like this causes trouble to `Tralics`

```
\def\foo#1{%
  \sbox\tempboxa{#1}%
  \ifdim \wd\tempboxa >\hsize
    #1\par
  \else \hbox to \hsize{\hfil\box\tempboxa\hfil}%
\fi}
```

It is a simplification of the `\@makecaption` command of the `article` class. The idea is to center the caption of an image if it fits on a line (centering is achieved via `\hfil`). The argument is typeset in a temporary box, and the width of the box is compared against `\hsize`. Captions in the `Raweb` are always centered, but this is not aesthetic.

### 2.11.6 More examples

Consider again the following example

```
\def\ifnch{%
  \ifx\lettoken\tempa\let\tempd\tempb\else\let\tempd\tempc\fi
  \tempd
}
```

It would be much simpler to write:

```
\def\ifnch{%
  \ifx\lettoken\tempa\tempb\else\tempc\fi
}
```

The problem here is that the commands `\tempb` and `\tempc` may take an argument, that would be `\else` or `\fi`. The remedy is

```
\def\ifnch{%
  \ifx\lettoken\tempa\expandafter\tempb\else\expandafter\tempc\fi
}
```

In general, you need an `\expandafter` before each token between `\else` and `\fi`. The command `\@afterfi` can be used to simplify such definitions. Its effect is easy: it reads all token, up to the `\fi` tokens, evaluates `\fi`, then the other tokens. Such a command is provided by the following packages: `typehtml`, `grabhdr`, `gmutils`, `gmverb`, `morehelp`, `splitbib`, `babel`, and maybe others. Example:

```
\def\test#1{
  \ifnum\count0=#1
    somecode
  \else\@afterfi\fct v\fi}
```

If the test is true, then `somecode` is evaluated, then everything between `\else` and `\fi` is discarded. But if the test is false, the else part is interpreted as if it were `\fi\fct v`. The command `\@afterelsefi` is to be used in the true part (all tokens between `\else` and `\fi` are discarded). In the example that follows, `\fct` is called with two arguments, the first one is `u` or `v`, the second is `2`.

```
\def\test#1{%
  \ifnum\count0=#1 %
```

```

    \@afterelsefi \fct u
    \else\@afterfi\fct v\fi}
\def\fct#1#2{} \test32

```

The piece of code that follows computes the factorial of a number, using only expandable commands (it requires `\numexpr`, an extension provided by  $\epsilon$ -T<sub>E</sub>X).

```

\def\JGfactorial#1{%
  \ifnum\numexpr#1>1
    \number \numexpr#1*\JGfactorial{(#1-1)}\relax
  \else 1\fi}

\def\factorial#1{%
  \ifnum\numexpr#1>1
    \number \numexpr#1*\factorial{(#1-1)}\expandafter\relax
  \else
    \expandafter1\fi}

\def\Factorial#1{%
  \number\ifnum\numexpr#1>1
    \numexpr#1*\Factorial{(#1-1)}\expandafter\relax
  \else
    1\expandafter\space
  \fi
}

\def\UDfactorial#1{%
  \number\ifnum\numexpr#1>1
    \numexpr#1*\UDfactorial{(#1-1)}\expandafter\relax
  \else
    \numexpr\ifnum\numexpr#1<0 0\else1\fi\expandafter\relax
  \fi
}%

```

Ulrich Diez, wrote versions 3 and 4; Version 3 uses a space character instead of `\space` using one of the techniques shown above; he then produced version 4, which gives a different value for the factorial of a negative number, and the space after the digit 1 is not needed anymore. In fact, if the argument is zero or one (case where the first `\ifnum` is false, version 1 and 2 return the character 1, while versions 3 and 4 return the digits of the number 1, computed by `\number`; in case 3, an optional space is read after the integer constant, in case 4, the `\relax` token is an end marker for `\numexpr`, an no optional space is needed after it (I guess that the purpose of *this* `\numexpr` if to avoid any problems if `\space` is redefined); the first `\numexpr` is needed for the product, and the two other calls are needed if the command calls itself). The difference between versions 2 and 3 is the placement of `\number`. I put it just before `\numexpr`, because `\numexpr` can be used only in a context where a number is seen. Ulrich puts it before the `\ifnum`. Does this make any difference? If you want to compute the factorial of a number, no. What about the following code:

```

\expandafter\expandafter\expandafter\def
\expandafter\expandafter\expandafter\factorialresult
\expandafter\expandafter\expandafter{\JGfactorial{12}}

```

The effect is the following. The command `\JGfactorial` is expanded twice, and the result is put in a command; evaluating this command yields the desired result. The same can be applied to `\UDfactorial`. In any case, the first expansion gives the body of the macro. The second

expansion expands the `\ifnum` and `\number` respectively. In one case you get lines two and three of `\JGfactorial`. This is something like

```
\def\factorialresult{... \else... \fi}
```

If you do not use this command, `TEX` will signal an unterminated `\if`. If you call it twice, you will get an extra `\else` error. On the other hand, if you consider `\UDfactorial`, the one-level expansion of `\number` implies expansion of the `\ifnum`, then the `\numexpr` of the body; expansion of the command means considering all tokens up to the final `\relax`, and since this `\relax` is preceded by `\expandafter`, everything up to the final `\fi` is taken into account. Thus, the one-level expansion of the body is a number, the desired result.

### 2.11.7 Producing N asterisks in a row

In appendix D of the `TEXbook`, there are some examples of how to produce N asterisks in a row. The question is: can we produce this using pure expansion? this is a solution given by D. Kastrup:

```
\def\nlines#1{\expandafter\nlineii\romannumeral\number\number #1 000\relax}
\def\nlineii#1{\if#1m\expandafter\theline\expandafter\nlineii\fi}
\def\theline{A}
\nlines{5}
```

This produces ‘AAAAA’. The idea is the following: ‘`\romannumeral3000`’ expands to ‘mmm’. It is then rather easy to convert this sequence of m into a sequence of A. The argument of the command can be ‘`\count0`’; the ‘`\number`’ has as effect to convert the value of this counter into a number, it gobbles a space. The argument of the command can be ‘`\count1□`’; the second ‘`\number`’ will gobble the second space (I don’t know if there is some other reason for these two `\number` commands). Here is the same idea, *without tests*:

```
\def\recur#1{\csname rn#1\recur}
\def\rn#1{}
\def\rnm#1{\endcsname{#1}#1}
\def\replicate#1{\csname rn\expandafter\recur
\romannumeral\number\number#1 000\endcsname\endcsname}

\dimen0=4sp \replicate{\dimen0}{P}
```

You may wonder how this works. Here is the transcript file of `Tralics`.

```
1 [216] \replicate{\dimen0}{P}
2 \replicate #1->\csname rn\expandafter \recur \romannumeral
3 \number \number #1 000\endcsname \endcsname
4 #1<-\dimen 0
5 {\csname}
6 {\expandafter \recur \romannumeral}
7 +scanint for \dimen->0
8 +scanint for \number->4
9 +scanint for \number->4000
10 +scanint for \romannumeral->4000
11 \recur #1->\csname rn#1\recur
12 #1<-m
13 {\csname}
14 \recur #1->\csname rn#1\recur
15 #1<-m
16 {\csname}
17 \recur #1->\csname rn#1\recur
18 #1<-m
19 {\csname}
```



```

20 \recur #1->\csname rn#1\recur
21 #1<-m
22 {\csname}
23 \recur #1->\csname rn#1\recur
24 #1<-\endcsname
25 {\csname}
26 {\csname->\rn}
27 \rn #1->
28 #1<-\recur
29 {\csname->\rnm}
30 \rnm #1->\endcsname {#1}#1
31 #1<-P
32 {\csname->\rnm}
33 \rnm #1->\endcsname {#1}#1
34 #1<-P
35 {\csname->\rnm}
36 \rnm #1->\endcsname {#1}#1
37 #1<-P
38 {\csname->\rnm}
39 \rnm #1->\endcsname {#1}#1
40 #1<-P
41 {\csname->\rn}
42 \rn #1->
43 #1<-P
44 Character sequence: PPPP .

```

This is now something else, it is part of a command defined in the RR style file:

```

\bggroup
\edef\foo{\ifnum 0<0#1x\else y\fi}\def\xbar{x}%
\ifx\foo\xbar
\global\compteurtheme=#1
\else \global\compteurtheme=0 \@latex@error{Pas un thème #1}\@eha\fi
\egroup

```

Assume that #1 contains a positive number, for instance 25. In this case, the test will be true, \foo will be defined as ‘x’, and will be equal to \xbar. In this case, our command puts 25 in \compteurtheme. Some other tests (not shown here) are done for instance, the value should be a number between 1 and 4, or a number with two digits, each one being between 1 and 4. Assume that the argument is not a number, say it is ‘gee’; then \ifnum will compare 0 and 0, the test will be false, \foo will be defined as ‘y’ hence is not equal to \xbar. Assume that the argument is ‘3a’; this is not a theme, but a theme and a subtheme. In this case, the test is true, but \foo expands to ‘3x’, and this is not equal to \xbar. Nowadays, themes are ‘com’, ‘cog’, etc, and this piece of code has become useless. It is replaced by something different, see end of section 6.9.

## 2.12 A nontrivial command \verb

The code that follows is a simplified version of a L<sup>A</sup>T<sub>E</sub>X command

```

1 \def\verb{%
2   \bgroup
3     \let\do\@makeother \dospecials
4     \verbatim@font\@noligs
5     \@vobeyspaces \frenchspacing\@sverb}
6
7 \def\verb@egroup{\global\let\VBG\@empty\egroup}

```

```

8 \let\VBG\@empty
9
10 \def\@sverb#1{%
11   \catcode'#1\active
12   \lccode'\~'#1%
13   \gdef\VBG{\verb@egroup\error{...}}%
14   \aftergroup\VBG
15   \lowercase{\let~\verb@egroup}}

```

Note first that this code contains two empty lines, that are read by T<sub>E</sub>X as a `\par` token (it is ignored, provided that the definition is read in vertical mode). Lines 5, 7, and 15 are terminated by a brace and the end of line character produces a space token, that is ignored for the same reasons. Lines 1, 10, 12, and 15 are terminated by a `%` character, since otherwise, it would produce a space character (ignored in case the command is executed in vertical mode, and that is not always the case). In the case of lines 2, 3, 4, etc., the end of line is converted into a space character that disappears because it follows a command name.

This code defines a command `\verb` that starts a group via `\bgroup`. At line 3, `\dospecials` is executed, after redefining `\do`. This changes the category code of all special characters (included all characters made active by packages like `babel`<sup>27</sup>). Line 4 changes the current font to a typewriter one, and it executes a piece of code that inhibits ligatures (for instance the one that converts a double dash in an en-dash). Note that this document contains a great number of verbatim examples, either inline or as environments. In some cases, we use a smaller font; it is hence important to allow the user to parameterize commands like these. Line 5 contains three commands: The first makes an end-of-line character active (usually, it will behave like `\par`), the second enters so-called french spacing mode (a mode where the width of a space is constant), and the last command `\@sverb` will be explained later. The ‘s’ in the name of this command comes from the ‘starred’ version of `\verb`: If you say ‘`\verb** +`’, you will get ‘`␣`’. We have omitted the test with the star character.

On lines 7 and 8, we define a command `\VBG` that does nothing (i.e. expands to the empty list) and a command that evaluates to `\egroup` preceded by a global assignment of `\VBG` to nothing. On line 13, `\VBG` is defined as calling `\verb@egroup` plus some error, whose text is not shown here. Thus `\VBG` is a command that 1) resets `\VBG` to a harmless command, 2) closes the current group, 3) signals an error.

Let’s consider lines 11 and 12. We assume that the argument of `\@sverb` is some character *c* (If you say `\def\foo{\verb\foo=\foo}` then `\foo`, you will get an error *Improper alphabetic constant*, and after that, you’re really in trouble. In the usual case, the character that follows `\verb` is read with category code 11 or 12, because of the code line 3.) Line 11 makes the character *c* active (of category 13); the category code will recover its old value at the end of the group, and line 13 changes the lc-code of the tilde character (the lc-code will recover its value at the end of the group). The lc-code of a character will be used for hyphenation, as well as conversion from upper case to lower case. We assume here, for the sake of simplicity, that hyphenation is inhibited by the use of a verbatim font. Note that `Tralics` does not care about subtleties like hyphenation. For this reason, when you say `\verb+foo+`, it will execute `\verbprefix {\verbatimfont foo}`. You can redefine both commands (the prefix is empty, the font defaults to `\tt`). Notice that `Tralics` grabs the argument, contrarily to L<sup>A</sup>T<sub>E</sub>X.

Line 14 contains the special command `\aftergroup`. This reads a token, saves it on a stack, and re-inserts it at the end of the current group.

Let’s come back to the L<sup>A</sup>T<sub>E</sub>X implementation of `\verb`. So far, we have read a character, changed its category code, changed the lc-code of the tilde character, changed the font and other

<sup>27</sup>A safe solution would be to change all category codes; but this is time consuming. On the other hand, if a character like `é` appears in a verbatim, it will be converted to a call (put acute accent on letter e), so that you will see `é` even in the case of a 7bit font.

tables, redefined `\VBG`, aftergrouped it (code on line 13: the token is popped at the end of the current group, that was opened on line 2, and normally closed on line 7). Line 15 is a kludge: what `\lowercase` does is replace in its argument every character by its lower case equivalent (using the `lc-code` table). The result is evaluated again. Here the argument is formed of three tokens: `\let`, the tilde and `\verb@egroup`. Since `~` is a character that has a lower-case equivalent, it will be replaced by that, namely the character `c`. Note: category codes are left unchanged by this procedure. It is hence important that `~` be an active character (because `\let` modifies that value of `~`) and that `c` be active (otherwise, there is no meaning in changing the value of `c`).

Consider the case of `\verb+\toto+`. Here the character `c` is the plus sign. After line 15 has been executed, the situation is the following: all characters are of category `other`, ligatures are disabled, french spacing is active, current font is `typewriter`, a group is opened, and a token is waiting for the group to terminate. In such a situation, you cannot go outside L<sup>A</sup>T<sub>E</sub>X properly. In fact, the carriage return has been made active in order to help error recovery (this is not shown here), and the `+` sign has been made active: this will help us. T<sub>E</sub>X sees now the following tokens `\_12 \_t11 \_o11 \_t11 \_o11 \_+13`. The first five tokens are added to the current horizontal list as characters in the current font, while the last one is expanded. The expansion is that of `\verb@egroup`, see line 7. This defines globally `\VBG`, then closes the group, restoring everything. It does not restore `\VBG` (because the last assignment was global). After the group, the after-grouped token `\VBG` is evaluated but it does nothing.

So far, so good: the translation of `'\verb+\foo+'` is the same as `'\texttt{\char'\foo}'`. Note that the author could have entered the previous expression as `'\verb-\verb+\foo+-'`, or using the `fancyvrb` package as `'|\verb+\toto+|'`, but he used `\quoted{\BS verb+\BS foo+}`, because, in the HTML file produced by Tralics, different colors are used for verbatim material; this is explained in the second part of this document.

Consider now the following example:

```
\def\duplicate#1{#1#1} '\duplicate{\verb+x+}++'
```

You would expect `'xx++'` but you get `'x+x+'`. Explanations: the expansion of `\duplicate` is `\_verb\_+12 \_x11 \_+12 \_verb\_+12 \_x11 \_+12 \_+?`. The last two plus signs have not been read, and their category code is still unassigned. The `\verb` command reads the `+12` via `\@sverb`. It changes the category code of the plus sign. The second `\verb` does the same. It reads the `+?` as a `+13`, this finishes evaluation of `\verb`. The second `\verb` command does the same. In the case where you replace `++` by `--`, the `\verb` command will see an end of line character before a plus character and complain with *LaTeX Error: \verb ended by end of line*.

Consider now the following example:

```
\def\braceme#1{#1} '\braceme{\verb+x+}++'
```

You get the following error *LaTeX Error: \verb illegal in command argument*. Let's try to see how this is done. The expansion of `\braceme` produces the following tokens: `\_1 \_verb\_+12 \_x11 \_+12 \_}2`. After `\@sverb` has finished, the first non-inactive character is `}2`, this closes the current group. Hence, as above, this restores category code, fonts, lc-codes, etc. It does not restore `\VBG` because assignment is global (`\gdef` at line 13 is like `\global\def`). The trick is now that a `\VBG` token is popped from the aftergroup stack. This one calls `\verb@egroup` and signals an error. What `\verb@egroup` does is to close a group (the one opened by `\braceme`), and reset `\VBG` to something harmless. Note that T<sub>E</sub>X is in a clean mode when the error is signaled. Tralics has no such error handling mechanism (however, no category codes are changed when scanning for the end of the command, so nothing harmful can be done). What this example shows is that error recovery is not completely trivial; nevertheless nice things can be done.

Note the following special cases;

```

\verb test
\verb+test+
\verb^^abtest^^ab

```

In the first case, the delimiter is a space character; the first line is terminated by a space and you would expect it to be interpreted in the same way as the second line. The trouble is that  $\TeX$  removes all spaces characters at the end of the line (regardless of category codes). The last line has also a problem: the delimiter is character 171 (double hat mechanism), and one  $\verb$  has changed category codes, the double hat sequence is not seen any more as such, and an error is signaled.

There is a variant to  $\verb$ , it is the ‘verbatim’ environment. The classical exercise is: write a command that reads everything up to  $\end{verbatim}$  (backslash and braces are of category 12 in this token list). There are different packages that solve this problem; For instance `fancyvrb` is one of them. A solution is also given in the first chapter. It does not allow an optional space after ‘ $\end$ ’.

We give here the  $\LaTeX$  implementation of the  $\end$  command.

```

\def\end#1{%
  \csname end#1\endcsname\@checkend{#1}%
  \expandafter\endgroup\if@endpe\@doendpe\fi
  \if@ignore\@ignorefalse\ignorespaces\fi}

```

As you can see, if you say  $\end{foo}$ , then  $\endfoo$  is executed first. After that the current environment in  $\@currenvir$  is compared with the argument, in case of error the variable  $\@oneline$  contains the start line of the environment. After that, the group is terminated, and we have two tests. The first uses  $\expandafter$ , this means that the command  $\@doendpe$  is executed outside the environment in the case where the variable  $\if@endpe$  is true inside the environment. This command is very complicated (it redefines  $\par$  and modifies  $\everypar$ ), and not implemented in `Tralics`; the effect is to suppress the indentation of the following paragraph. On the other hand, the two commands  $\@ignoretrue$  and  $\@ignorefalse$  redefine  $\if@ignore$  globally, so that no  $\expandafter$  is needed for this one.

This is an example of  $\aftergroup$ .

```

\def\lrbox#1{%
  \edef\reserved@a{%
    \endgroup
    \setbox#1\hbox{\begingroup\aftergroup}%
    \def\noexpand\@currenvir{\@currenvir}%
    \def\noexpand\@currenvline{\@oneline}%
  }%
  \reserved@a
  \@endpefalse \color@setgroup \ignorespaces}
\def\endlrbox{\unskip\color@endgroup}

```

The effect of the  $\edef$  command is to replace the previous definition by the following (where ‘17’ is to be replaced by the current line number). One important point here is that implementing colors in  $\LaTeX$  is non trivial, and for this reason, there are two hooks (the commands with the name ‘color’, that do nothing if the package is not loaded). Colors are not implemented in `Tralics`.

```

\def\lrbox#1{%
  \endgroup
  \setbox#1\hbox{\begingroup\aftergroup}%
  \def\@currenvir{lrbox}%
  \def\@currenvline{ on input line 17}%
  \@endpefalse \color@setgroup \ignorespaces}

```

The order of evaluation is the following. Assume that the current environment is X. The `\begin` command opens a group via `\begingroup` and changes the environment name to 'lrbox'. The command starts with `\endgroup`, closing this group. After that, we put something in the box whose number is the argument of the environment; the content is a hbox, whose start is defined by the brace (and this brace is a group); we start a group with `\begingroup`, and call `\aftergroup`. This pushes a brace on the stack; this brace indicates the end of the hbox, but it will be evaluated later. After that, we change again the name of the current environment (it was restored to X by the `\endgroup`, but we made a copy of it in the `\edef`). When the end of the environment is reached, the following happens. First, the end-code is executed (this removes space at the end of the box), and `\endgroup` is executed. As a side-effect this restores the current environment name to X. It also pops the after group stack, namely the closing brace that terminates the `\hbox`. One important point here is that the `\setbox` assignment is done outside the environment (it could be done inside, with a `\global` prefix). Such a piece of code is illegal. The `lrbox` environment is not implemented in Tralics version 2.10.

## 2.13 Expandable tokens

Assume that `\err` is an undefined command. The following code

```
\ifnum1=0\err1 \err1 \fi
```

will signal two errors: when T<sub>E</sub>X reads the second number, it expands undefined command (hence a first error), and continues scanning, until finding the space; the test is true, hence the second error.

We give here the list of all tokens that can be expanded.

- All user defined command.
- All undefined commands.
- `\noexpand`: inhibits expansion of the next token.
- `\expandafter`: changes order of expansion.
- `\csname`. This manufactures a token. Note that `\endcsname` marks the end of the list, is not expandable, is an error elsewhere.
- `\number`, `\@arabic`, `\romannumeral`, `\Romannumeral`: these convert numbers.
- `\string`, `\meaning`, `\fontname`, `\jobname`, `\tralicsversion`: These commands convert some internal quantities into tokens; for instance `\jobname` is the name of the file that is translated (without an extension 'tex'), and `\tralicsversion` is the version number of Tralics; it could be '2.5 (pl7)', or more likely '2.9'.
- `\the`. You say `\the\foo`, if you want to typeset the value of a variable `\foo`.
- `\input`, `\include`, `\endinput`: Special macros for files.
- `\if`, `\ifcat`, `\ifnum`, `\ifdim`, `\ifodd`, `\ifvmode`, `\ifhmode`, `\ifmmode`, `\ifinner`, `\ifvoid`, `\ifhbox`, `\ifvbox`, `\ifx`, `\ifeof`, `\iftrue`, `\iffalse`, `\ifcase`. These start a conditional.
- `\fi`, `\or`, `\else`. These modify the conditional stack.
- `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark`, `\splitbotmark`. Marks are not implemented. There are  $\epsilon$ -T<sub>E</sub>X extensions, with the same same but for a final s, that manage a stack of marks.

- A number of commands defined by  $\text{\LaTeX}$ , are implemented as expandable commands in  $\text{\Tralics}$ .
- $\text{\a}$ ,  $\text{\'}$ ,  $\text{\'}$ ,  $\text{\"}$ ,  $\text{\HAT}$ ,  $\text{\~}$ ,  $\text{\k}$ ,  $\text{\H}$ ,  $\text{\v}$ ,  $\text{\b}$ ,  $\text{\d}$ ,  $\text{\u}$ ,  $\text{\C}$ ,  $\text{\f}$ ,  $\text{\c}$ ,  $\text{\.}$ ,  $\text{\=}$ ,  $\text{\r}$ ,  $\text{\T}$ ,  $\text{\V}$ ,  $\text{\D}$ ,  $\text{\h}$ : these commands produce an accented character.
- Commands of the form  $\text{\textunderscore}$ ,  $\text{\AA}$ ,  $\text{\texteuro}$  expand to a Unicode character.
- $\text{\@firstofone}$ ,  $\text{\@firstoftwo}$ ,  $\text{\@secondoftwo}$  expand to the first or second argument.
- $\text{\@car}$ ,  $\text{\@cdr}$ : these give access to first or rest of a list terminated by  $\text{\@nil}$ .
- $\text{\@gobble}$ ,  $\text{\@gobbletwo}$ ,  $\text{\@gobblefour}$ : these commands read and ignore 1, 2 or 4 arguments.
- $\text{\zap@space}$  removes unwanted spaces.
- $\text{\strip@prefix}$  removes the prefix inserted by  $\text{\meaning}$  before the body of a macro.
- $\text{\ensuremath}$  inserts dollars signs outside math mode.
- $\text{\hexnumber@}$ ,  $\text{\multispan}$ , are defined as in  $\text{\LaTeX}$ .
- $\text{\@afterfi}$ ,  $\text{\@afterelsefi}$ : These commands can be used in a if-then-else structure, they terminate the condition and re-insert the interesting tokens.
- $\text{\(}$ ,  $\text{\)}$ ,  $\text{\[}$ ,  $\text{\]}$ : commands for math mode.
- $\text{\UseVerb}$ : restores a quantity saved by  $\text{\SaveVerb}$ .
- $\text{\@stpelt}$ ,  $\text{\stepcounter}$ ,  $\text{\refstepcounter}$ ,  $\text{\addtocounter}$ ,  $\text{\setcounter}$ ,  $\text{\value}$ . These are user-defined commands in  $\text{\LaTeX}$ . The expansion in  $\text{\Tralics}$  can depend on the loading of the  $\text{\calc}$  package.
- $\text{\setlength}$ ,  $\text{\addtolength}$ . These are user-defined commands in  $\text{\LaTeX}$ . The expansion in  $\text{\Tralics}$  can depend on the loading of the  $\text{\calc}$  package.
- $\text{\arabic}$ ,  $\text{\roman}$ ,  $\text{\Roman}$ ,  $\text{\alph}$ ,  $\text{\Alph}$ ,  $\text{\fnsymbol}$ ,  $\text{\@alph}$ ,  $\text{\@Alph}$ ,  $\text{\@fnsymbol}$ : commands to typeset a  $\text{\LaTeX}$  counter.
- $\text{\loop}$ ,  $\text{\@whilenum}$ ,  $\text{\@whiledim}$ ,  $\text{\@whilesw}$ : For loops.
- $\text{\xspace}$ . This adds a space if needed.

## Chapter 3

# Mathematics

### 3.1 Introduction

Mathematics play a great role in T<sub>E</sub>X and Tralics. For instance, T<sub>E</sub>X has three modes: vertical mode, in which no typesetting is done, horizontal mode (where everything happens) and math mode, a mode in which special objects are handled; a two phase process converts these special objects in normal ones. Fonts to be used in math mode have special properties (see appendices F and G of the T<sub>E</sub>Xbook). Not all subtleties of T<sub>E</sub>X math can be implemented in Tralics; on the other hand, the XML translation is conforming to MathML. This defines some entities, for instance in isoamsc.ent, there is a definition of `&rceil`; to `&#x02309`; . As a consequence, Tralics will translate `\rceil` to `<mo>&rceil;</mo>` or `<mo>&#x02309;</mo>`, depending on an option. Translation of a footnote is in general a `<footnote>` element, and the user can change the name of this element; this is not done for maths: the name `<mo>` is a constant.

The syntax of mathematics is often strange. Instead of

```
\math{E=\frac{1}{2} m\superscript{v}{2}}
```

you say

```
$E={1\over 2} mv^2$
```

Three categories codes are defined for use in math mode, they correspond to the dollar sign (math shift), underscore character (subscript) and hat character (superscript). If you want a dollar or underscore character, you can say `\$`, or `\_`, but `\^` produces an accent over what follows, not a hat character (In L<sup>A</sup>T<sub>E</sub>X, you can say `\textasciicircum`, provided that you can guess the name).

In the example above, we have two pseudo commands `\fraction` and `\superscript` (followed by two arguments) whereas the plain T<sub>E</sub>X version uses infix operators (placed between the arguments). The first opertr is greedy. This means that, without the braces in the example above, everything before `\over` would be the numerator, and everthing after it would be the denominator. On the other hand, you see sometimes  $2^16$  instead of  $2^{16}$ , when people forget braces around the superscript. The essential difference however is that arguments are typeset in different style: the nucleus (what precedes the hat operator) is typeset in text style, while numerator, denominator, superscripts and subscripts are in script style; moreover, it two objects are placed one above the other, cramped style is used used for the object that is below the other one (i.e., the denominator or a subscript). The style influences spacing; because of commands like `\over`, the current style is known only after the whole expression is parsed. This explains why you may see: *Package amsmath Warning: Foreign command \over; \frac or \genfrac should be used instead.*

T<sub>E</sub>X has also a notion of “inner” mode. Inside an inner object, you cannot put an outer one. Such a distinction exists also in HTML, where `<div>` is outer and `<span>` is inner. We explained in

the previous chapter that `\ifinner` can be used to check whether current mode is inner or outer, and we mentioned that, outside math mode, this is not well defined in `Tralics`. This may produce surprising results. Consider for instance `\hbox{$$$}`. Inner mode is the rule inside a box, and a double dollar sign signals the start of an outer (display math) formula. You would expect this expression to provoke an error. In fact, `TeX` assumes that you know what you do, enters inner math mode when it sees the first dollar sign, and quits when it sees the second one; this gives an empty math formula (in fact, it will contain all tokens from the `\everymath` hook), surrounded by some space: the value of `\mathsurround` (this can be set to zero using `\m@th`). Note that a math formula defines group: assignments made inside the formula are forgotten after full evaluation (in particular after this space is added).

The essential difference between inner (normal, inline) math and outer (display) math is that a display formula uses a line of its own (very often the formula is centered on the line). One could say that a display formula terminates the current paragraph. In fact, it is just interrupted, the paragraph continues after the formula (this is only interesting in constructions like `\parshape`, whose scope is the current paragraph; here a formula counts for three lines; not implemented in `Tralics`). The construction `\hbox{$$$ x$$$}` produces a display math formula in `Tralics`, instead of two empty math formulas. Before version 2.11.7, an error was signaled (because `Tralics` started a new paragraph at the end of the equation, and this is illegal in a box).

A display math formula can have an equation number (via commands `\eqno`, `\leqno`, `\tag`, `\notag`; these commands were not implemented in early versions, and are described in the last chapter of the second part of this report). The MathML documentation says “One of the important uses of `<mlabeledtr>` is for numbered equations. In a `<mlabeledtr>`, the label represents the equation number and the elements in the row are the equation being numbered. The `side` and `minlabelspacing` attributes of `<table>` determine the placement of the equation number.” Thus, the recommended way, for MathML, is to use a table, like this (replace ellipsis by an expression)

```
<table>
  <mlabeledtr id='e-is-m-c-square'>
    <td>
      <mtext> (2.1) </mtext>
    </td>
    <td>
      ...
    </td>
  </mlabeledtr>
</table>
```

This mechanism is not yet implemented. We do not know how to insert numbers automatically, so that the proposed solution is: you can use `\label`, `\ref` for any display math formula. This will add an `id` attribute to the `<formula>` object, which is a wrapper for the `<math>`.

When you say `{\alpha^2}`, `TeX` will enter math mode with an error of the form *Missing \$ inserted*. On the other hand, `Tralics` will signal two errors, the first is *Math only command \alpha. Missing dollar not inserted*, the second is *Missing dollar not inserted, token ignored: {Character ^ of catcode 7}*. If you want a command that works in math mode and outside math mode, you can say:

```
\def\foo{\ifmmode \alpha^2 \else $\alpha^2$\fi}
```

This can be generalised, using the following command

```
\DeclareRobustCommand{\ensuremath}{%
  \ifmmode
    \expandafter\@firstofone
  \else
    \expandafter\@ensuredmath
```



```

\fi}
\long\def\@ensuremath#1{\relax#1$}

```

The purpose of the `\relax` on the last line is for the case of an empty argument: we do not want `\ensuremath{}` to expand to `$$`. Note that the argument is handled only once (i.e., `\ensuremath` does not read it, but calls a helper), because of subtle bugs, see latex bugs data base amslatex/2104. We shall say later ‘Mode independent commands are interpreted as usual’, this implies that the `\relax` token will do nothing. We shall see later that, in non-mathml mode, `\relax` appear in the result unless it is the first in the list. Other commands, not listed in this chapter, may signal an error. For instance, `\par` is forbidden. Note that `\mathchar` provokes an *Unimplemented command* error. If you want a random Unicode character, you should use commands like `\mathmi`, `\mathmo`, `\mathmn`. You can also define a command via `\chardef` or `\mathchardef` (the result is the same), and use it, the result is always a `<mi>` element. The following example shows that `\amp` produces an ampersand sign in some case, it must be used with care.

```

\chardef\AAA"1000
\chardef\CCC'x
\mathchardef\BBB"2000
$\mathbf{x\AAA\BBB\CCC} \mathmi{foo}\mathmo{\&\#666;}\mathmo{\amp\#777;}$

```

Translation

```

<formula type='inline'>
  <math xmlns='http://www.w3.org/1998/Math/MathML'>
    <mrow>
      <mi mathvariant='bold'>x</mi>
      <mi>&\#x1000;</mi>
      <mi>&\#x2000;</mi>
      <mi>x</mi>
      <mi>foo</mi>
      <mo>&#666;</mo>
      <mo>&\#777;</mo>
    </mrow>
  </math>
</formula>

```

Because a math expression translates as `<math>` inside a `<formula>`, and that the math has a long namespace attribute, examples will never fit on a single line. In order to make the result easier to read, we have inserted some newline characters, and reindented all these examples. Two consecutive newline characters are scanned by T<sub>E</sub>X as space plus `\par`. This space is ignored by T<sub>E</sub>X (see T<sub>E</sub>Xbook, the text between exercises 14.12 and 14.13). Hence the general rule in Tralics: when a `<p>` element is ended, a trailing space or newline is removed from the content of the element, a newline character is added to the parent of the `<p>`. As a result, you will very often see `<p>` at the start of a line and `</p>` at the end of a line in a XML file generated by Tralics.

Consider the following simple example:

```

$\alpha$ and $$\beta \label{foo}$$

```

The translation is the following

```

<p>
  <formula type='inline'>
    <math xmlns='http://www.w3.org/1998/Math/MathML'>
      <mi>&\alpha;</mi>
    </math>
  </formula> and</p>
<formula id='uid1' type='display'>

```

```

<math xmlns='http://www.w3.org/1998/Math/MathML'>
  <mi>&beta;</mi>
</math>
</formula>

```

You can also say

```
\(\alpha\) and \[\beta \label{foo}\]
```

The result is exactly the same. In L<sup>A</sup>T<sub>E</sub>X, the commands `\(`, `\)`, `\[` and `\]` test the current mode. No such test is done by Tralics. The L<sup>A</sup>T<sub>E</sub>X implementation of `\[` is a bit strange. If the formula is in vertical mode, it will be preceded by a box of width `.6\linewidth` containing nothing (except two `\hss` commands to fill it) preceded by the current paragraph indentation. The command `\]` executes `\ignorespaces`. As you can see, there is some difference between a single dollar and a double dollar. In the first case, we are in normal math mode, otherwise in display math mode. One difference is the initial style: it is `\textstyle` (for normal mode) and `\displaystyle` otherwise (this will be explained later). A second difference is that the `\everymath` or `\everydisplay` token list is inserted when scanning the formula depends on the mode. The third difference is specific to Tralics. A display math formula is never ‘trivial’ (see section 3.5), it can have a label (not more than one): in this case, the `<formula>` element has an `id` attribute. In any case, the `<formula>` element has a `type` attribute that explains that the formula is inline or display. A non-display formula starts a paragraph; a display math formula cannot appear in a paragraph (the equivalent of `\par` is executed), if the first non-space token (after expansion) that follows the math formula is not `\par`, a `\noindent` token will be inserted (see line 34 of the transcript at page 92). Note that, in T<sub>E</sub>X, a math formula does not end a paragraph, in the sense that a `\parshape` is valid across math formulas; however what precedes the formula is split into lines, according to parameters in force at the start of the formula. Tralics does not split paragraphs into lines, and does not implement use `\parshape`.

## 3.2 The basic objects

The following environments are recognized outside math mode, and produce a math formula: `eqnarray*`, `align*`, `aligned`, `split`, `multline`, `equation*`, `math` and `displaymath`. When Tralics sees a dollar character, it looks at the next character (without expansion). If this is a dollar sign, it will be read, and display math mode is entered, otherwise, normal math mode is entered. All environments shown above start display math mode (except `math`, which enters normal math mode). The environments `math` and `displaymath` are equivalent to `\(...\)` and `\[...\]` respectively. The environments `eqnarray`, and `split` are implemented as arrays. There is no difference between

```

\begin{eqnarray} a&b \\ c&d \end{eqnarray}
\begin{split} a&b \\ c&d \end{split}

```

and

```

\[\begin{array}{rcl} a&b \\ c&d \end{array}\]
\[\begin{array}{rl} a&b \\ c&d \end{array}\]

```

Environments `equation` and `align` are translated as normal math. A star after the environment name is ignored. In the case of normal math mode, the content of the token list `\everymath` is inserted before the formula, for `displaymath` it is `\everydisplay`. For instance, if you say

```

\everymath={ (N) \ }
\everydisplay={ (D) \ }
$\alpha$ and $$\beta$$

```

the translation will be

```

<p>
<formula type='inline'>
  <math xmlns='http://www.w3.org/1998/Math/MathML'>
    <mrow>
      <mo></mo><mi>N</mi><mo></mo><mspace width='6pt' />
      <mi>&alpha;</mi></mrow></math></formula> and</p>
<formula type='display'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
  <mrow>
    <mo></mo><mi>D</mi><mo></mo><mspace width='6pt' />
    <mi>&beta;</mi>
  </mrow>
</math>
</formula>

```

In T<sub>E</sub>X, you can put anything inside a math formula, provided it is hidden in a box; this is not possible in Tralics, because we want the XML result to be conforming to MathML. We shall list here all commands valid in math mode, and explain later on how they are translated.

Commands `\limits`, `\nolimits` and `\displaylimits` can be used just after an operator and before subscripts or superscripts, as in `\int \limits_x`. They are currently ignored by Tralics.

The following environments are recognized: `array`, `matrix`, `pmatrix`, `bmatrix`, `Bmatrix`, `vmatrix`, `Vmatrix`. All these environments produce arrays. For the first, an argument is required, explaining how cells are aligned. For all other environments, cells are centered. Environments of the form `Xmatrix` have fences, an implicit `\left` and `\right`. In order: parentheses, braces, brackets, simple bars, double bars. There is also an environment `cases`, with two columns, left aligned, that has an open brace as left delimiter, an empty right delimiter. Example

```

$ \begin{array}{lcr} a & b & c \end{array}
\begin{bmatrix} d & e \\ f & g \end{bmatrix} $

```

The translation is the following.

```

<formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
  <mrow>
    <table>
      <tr>
        <td columnalign='left'><mi>a</mi></td>
        <td><mi>b</mi></td>
        <td columnalign='right'><mi>c</mi></td>
      </tr>
    </table>
    <mfenced open='{ ' close='}'>
      <table>
        <tr>
          <td><mi>d</mi></td>
          <td><mi>e</mi></td>
        </tr>
        <tr>
          <td><mi>f</mi></td>
          <td><mi>g</mi></td>
        </tr>
      </table>
    </mfenced>
  </mrow>
</math>

```



$$\alpha \Gamma \sqrt{\phantom{x}}$$

This translates as

```
<formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
  <mrow><mi>&alpha;</mi><mi>&Gamma;</mi><mi>&radic;</mi>
</mrow></math></formula>
```

Next comes the list of all symbols whose translation is like log. There are of type Ord (ordinary symbol), though they should be Op (large operator). The list is divided in two parts: these have movable limits: `\det`, `\gcd`, `\inf`, `\injlim`, `\liminf`, `\limsup`, `\max`, `\min`, `\sup`, `\projlim`, and these have not: `\dim`, `\exp`, `\hom`, `\ker`, `\lg`, `\lim`, `\ln`, `\log`, `\Pr`, `\arccos`, `\arcsin`, `\arctan`, `\arg`, `\cos`, `\cosh`, `\cot`, `\coth`, `\csc`, `\deg`, `\sec`, `\sin`, `\@mod`, `\sinh`, `\tan`, `\tanh`. Example

$$\lim_a \liminf_a \sin_a \hom_a$$

The L<sup>A</sup>T<sub>E</sub>X translation is  $\lim_a \liminf_a \sin_a \hom_a$ , and the Tralics version is

```
<formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mstyle scriptlevel='0' displaystyle='true'>
<mrow>
  <msub><mo movablelimits='true' form='prefix'>lim</mo> <mi>a</mi> </msub>
  <msub><mo movablelimits='true' form='prefix'>lim inf</mo><mi>a</mi></msub>
  <msub><mo form='prefix'>sin</mo> <mi>a</mi> </msub>
  <msub><mo form='prefix'>hom</mo> <mi>a</mi> </msub>
</mrow></mstyle></math></formula>
```

From now on, all symbols translate into the form `<mo>...</mo>`. We start with symbols of type Ord. In reality, most of them they should be of type Op (large operator). `\mho`, `\clubsuit`, `\diamondsuit`, `\heartsuit`, `\spadesuit`, `\aleph`, `\backslash`, `\Box`, `\imath`, `\jmath`, `\square`, `\cong`, `\lnot`, `\neg`, `\forall`, `\exists`, `\coprod`, `\bigvee`, `\bigwedge`, `\biguplus`, `\bigcap`, `\bigcup`, `\int`, `\sum`, `\prod`, `\bigotimes`, `\bigoplus`, `\bigodot`, `\oint`, `\bigsqcup`, `\smallint`. Examples

$$\bigcap \int \oint$$

The translation is

```
<mrow><mo>&bigcap;</mo><mo>&int;</mo><mo>&oint;</mo></mrow>
```

These are of type Bin (binary operator). `\triangleleft`, `\triangleright`, `\bigtriangleup`, `\bigtriangledown`, `\wedge`, `\land`, `\vee`, `\lor`, `\cap`, `\cup`, `\multimap`, `\dagger`, `\ddagger`, `\sqcap`, `\sqcup`, `\amalg`, `\diamond`, `\Diamond`, `\bullet`, `\wr`, `\div`, `\odot`, `\oslash`, `\otimes`, `\ominus`, `\oplus`, `\uplus`, `\mp`, `\pm`, `\circ`, `\bigcirc`, `\setminus`, `\cdot`, `\ast`, `\times`, `\star`, `\in`. Example

$$\cap \cup \wr$$

The translation is

```
<formula type='inline'><math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow><mo>&cap;</mo><mo>&cup;</mo><mo>&wr;</mo></mrow></math></formula>
```

These are of type Rel (relation). `\propto`, `\sqsubseteq`, `\sqsupseteq`, `\sqsubset`, `\sqsupset`, `\parallel`, `\mid`, `\dashv`, `\vdash`, `\Vdash`, `\models`, `\nearrow`, `\searrow`, `\nrightarrow`, `\swarrow`, `\Leftrightarrow`, `\Leftarrow`, `\Rightarrow`, `\neq`, `\neq`, `\leq`, `\leq`, `\geq`, `\geq`, `\succ`, `\approx`, `\succeeds`, `\preceq`, `\prec`, `\doteq`, `\supseteq`, `\subset`, `\supseteq`, `\subseteq`, `\bindasrepma`, `\ni`, `\gg`, `\ll`, `\gtrless`, `\geqslant`, `\leqslant`, `\not`, `\notin`, `\Leftrightarrow`, `\leftarrow`, `\owns`, `\gets`, `\rightarrow`, `\to`, `\mapsto`, `\sim`, `\simeq`, `\perp`, `\equiv`, `\asymp`, `\smile`, `\iff`, `\leftharpoonup`, `\leftharpoondown`, `\rightharpoonup`, `\rightharpoondown`, `\hookrightarrow`,

`\hookleftarrow`, `\Longrightarrow`, `\longrightarrow`, `\longleftarrow`, `\Join`, `\longmapsto`, `\frown`, `\bowtie`, `\Longleftarrow`,

`\longlefttrightarrow`, `\Longlefttrightarrow`. Example.

```
\$ \approx \leftrightharpoonup \Longlefttrightarrow \$
```

Translation:

```
<formula type='inline'><math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow><mo>&approx;</mo><mo>&leftrightharpoonup;</mo>
<mo>&Longlefttrightarrow;</mo></mrow></math></formula>
```

These are of type Inner: `\cdots`, `\hdots`, `\vdots`, `\ddots`. These are of type Between (they are of type Ord in  $\TeX$ , but are used as opening or closing delimiters): `\Vert`, `\lvert`, `\rvert`, `\uparrow`, `\downarrow`, `\Uparrow`, `\Downarrow`, `\Updownarrow`, `\updownarrow`. These are of type Open and Close: `\rangle`, `\langle`, `\rmoustache`, `\lmoustache`, `\rgroup`, `\lgroup`, `\rbrace`, `\lbrace`, `\lceil`, `\rceil`, `\lfloor`, `\rfloor`.

The following characters are classified as ‘small’: `<`, `>`, `.`, `:`, `*`, `?`, `!`, `x`, these are classified as ‘small-l’ and ‘small-r’: `()` `[]`, the vertical bar is small-l, these are bin: `+` `/` and the equals sign is of type Rel. Note: what you see here as `x` is in reality the character 215. It cannot be printed in verbatim mode by  $\LaTeX$ .

```
$ < , . : * ? ! x ( ) [ ] | + / = $
```

Translation:

```
<formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow><mo>&lt;</mo><mo>&gt;</mo><mo>,</mo><mo>.</mo><mo>:</mo>
<mo>;</mo><mo>*</mo><mo>?</mo><mo>!</mo><mi>&times;</mi><mo>( </mo>
<mo>)</mo><mo>[ </mo><mo>]</mo><mo>| </mo><mo>+ </mo><mo>- </mo>
<mo>/ </mo><mo>=</mo>
</mrow></math></formula>
```

The following commands are used for accents: `\acute`, `\grave`, `\mathring`, `\ddddot`, `\dddot`, `\ddot`, `\tilde`, `\widetilde`, `\bar`, `\breve`, `\check`, `\hat`, `\widehat`, `\vec`, `\overrightarrow`, `\overleftarrow`, `\underrightarrow`, `\underleftarrow`, `\dot`.

The following commands are special. They will be explained later: `\overline`, `\underline`, `\stackrel`, `\underset`, `\overset`, `\mathchoice`, `\frac`, `\overbrace`, `\underbrace`, `\genfrac`, `\dfrac`, `\tfrac`, `\sqrt`, `\root`.

### 3.3 Parsing a math formula

This is a non-trivial operation, for this reason in verbose mode, the math expression will be printed on the transcript file. For instance, given

```
\tracingall
$\begin{cases} x & y \\ a & b \end{cases} \mkern18mu x^{\{2\}}!$
```

whose translation in no-mathml mode is

```
<texmath type='inline'>
{\left\rbrace \begin{array}{ll} x & \&y \\ a & \&b \end{array}\right.}
\hspace{10.0pt}x^{\{2\}}!
</texmath>
```

the transcript file will contain

```
1 {math shift character $}
2 +stack: level + 2 for math entered on line 2
```

```

3 +stack: level + 3 for math entered on line 2
4 \cases ->\left \{\begin {array}{ll}
5 +stack: level + 4 for math entered on line 2
6 +stack: level + 5 for cell entered on line 2
7 +stack: level + 6 for math entered on line 2
8 +stack: level - 6 for math from line 2
9 +stack: level - 5 for cell from line 2
10 +stack: level + 5 for cell entered on line 2
11 +stack: level - 5 for cell from line 2
12 +stack: level + 5 for cell entered on line 2
13 +stack: level - 5 for cell from line 2
14 +stack: level + 5 for cell entered on line 2
15 \endcases ->\end {array}\right .
16 +stack: level - 5 for cell from line 2
17 +stack: level - 4 for math from line 2
18 +stack: level - 3 for math from line 2
19 +scanint for \mkern->18
20 +scandimen for \mkern->18.0mu
21 +stack: level + 3 for math entered on line 2
22 +stack: level - 3 for math from line 2
23 +stack: level + 3 for math entered on line 2
24 +stack: level + 4 for math entered on line 2
25 +stack: level - 4 for math from line 2
26 +stack: level - 3 for math from line 2
27 +stack: level - 2 for math from line 2
28 Math: $\begin {cases}{\left\{\begin {array}{ll} x & y\end{array}\end{cases}
29 \end {array}\right.} \mkern\hspace{10.0pt}x^{2 }!$
30 +scanint for \hspace->10
31 +scandimen for \hspace->10.0pt
32 {scanglue 10.0pt\relax }
33 Realloc xml math table to 20
34 {Push p 1}

```

We shall explain for each line in the transcript file where it comes from. Math mode scanning is entered when the translator sees a math shift character (line 1). The scanner reads some tokens and puts them in a list. The list is printed at the end (lines 28-29). The start of the formula is a bit special, in that the token that follows the first dollar sign is considered unexpanded when we check for a double dollar sign. A new group is entered, before scanning the whole formula (line 2).

The loop is as follows:

- A token is read and expanded. Lines 4 and 15 show expansion of user commands. An error is signaled in the case of end of data.
- In the case of `\nobreakspace`, we insert a `~`.
- If we get a font command, we proceed as follows. First `\cal` is transformed into `\mathcal`. The font can be `\mathtt`, `\mathcal`, `\mathbf`, `\mathrm`, `\mathit`, `\mathbb`, `\mathsf`. These are basic math fonts; they have an inner variant, of the form `\@mathtt`. There is also `\mathnormal`. The command `\mathfrak` selects a Fraktur variant. We allow old fonts (like `\rm`, `\sf`, `\tt`, `\bf`, `\it`, `\sl`), fonts switches of the form `\rmfamily`, or font commands that take an argument like `\textrm`. These fonts have an inner variant, say *T*. If the font takes no argument, then the token *T* is inserted (as explained for `\cal` above). Otherwise, let *S* be the current math font. In this case, an argument is read, then *S*, *T* and this argument is

pushed back, to be read again. For instance, if the current font is ‘sf’, then `\mathrm{foo}` produces `\@mathrm foo\@mathsf`: these are five tokens to be read again. Note: `\mathbbm` is an alias for `\mathbb`. The name of the internal font has changed since the first edition, it has the form `mml@font@xxx`, where the suffix is one of normal, upright, bold, italic, bolditalic, script, boldscript, fraktur, doublestruck, boldfraktur, sansserif, boldsansserif, sansserifitalic, sansserifbolditalic, and monospace. Details can be found in the second part of this report.

- In all other cases the current token is added to the list. In particular, this explains while the trace starts with a dollar and `\begin`.
- If the token is an open brace, in fact any character of category code 1, a new math group is read. You can see on lines 23 and 24 that the stack level increases (a new semantic level is entered, all assignments are local).
- If the token is a close brace, in fact any character of category code 2, this terminates the current math group (see lines 25 and 26). An error is signaled in case the current group should be closed differently (for instance with `\end`, or `\right`, etc.)
- If the token is a dollar sign, in fact any character of category code 3, then four alternatives can be chosen. This dollar sign can be the end of the math formula. If we are in display math, a token is read with expansion. An error is signaled *Display math should end with \$\$* if this is not a dollar sign (in fact, a character of category code 3). If the current group is defined by `\hbox`, this can be the start of a math formula (never display math). The token in `\everymath` are inserted, then a math formula is read. Otherwise, an error is signaled *Extra \$ ignored...*, parsing continues.
- In the case of `\label`, an argument is read. If we are not in display math, or if the formula already has a label, you get an error: *Some labels may be lost*. Wherever the location of the label, an attribute will be added to the `<formula>` element that contains the `<math>` element.
- In the case of `\ensuremath`, a token list is read, and pushed back, so that this command acts as `\@firstofone`.
- The case `\begin` or `\end` is considered next. We make the assumption that this is a user defined environment, or a math environment. In the case of the example, we have a user environment that expands to a math environment. For a user defined environment, the following is executed:

```
{\cases .... \endcases}
```

In the trace, lines 28-29, you will see both `\begin{cases}` and the result of the expansion of `\cases`, but not `\cases` or the brace. However, you can see on lines 4 and 15 the expansion of the user defined commands, and on lines 3 and 18 the braces; these braces can also be seen in the translation in no-mathml mode. You can see on lines 6 and 16 that a group (named ‘cell’) is opened and closed, because the builtin math environment starts a cell. This allows `&` or `\` tokens. The group defined on lines 7-8 does not exist in T<sub>E</sub>X. Let’s hope for the best: the argument of the array should contain only letters. Whether these characters should be expanded is unclear.

- In the case `\left` and `\right`, a delimiter is read. The rules are: `\relax` and space tokens are ignored. After full expansion, the result should be one of the tokens listed above as valid delimiters, otherwise an error of the form *Invalid character in \left or \right* can be signaled. These commands come in pairs; you might get errors like *Missing \right. inserted*, or *Unexpected \right*. These commands define a group, see lines 5 and 17.



- Case `&` and `\`. These are valid only inside a cell group. They terminate a cell group and start a new one. See lines 9 to 14.
- The `\of` token is ignored.
- The `\mathchoice` command reads 4 arguments, which are remembered.
- Case of `\frac`, etc. These commands read their arguments. The main token list will contain a special slot, with the name of the command and the arguments. In the case of `\sqrt`, the first argument is optional. You say `\root A \of B`.
- The syntax of `\genfrac` is special. It takes six arguments. The equivalent of the following commands is executed when Tralics bootstraps:

```
\def\binom{\genfrac(){}{}}
\def\dbinom{\genfrac(){0pt}0}
\def\tbinom{\genfrac(){0pt}1}
```

This defines three commands that take two arguments with regular syntax. The first two arguments of `\genfrac` are delimiters or empty. The next one is a dimension or empty. If empty, a default dimension will be used. In the example, the first argument is an opening parenthesis, the second is a closing parenthesis, the third is zero. The next argument is empty or a number between 0 and 3. These numbers correspond to a style: `\displaystyle`, `\textstyle`, `\scriptstyle`, and `\scriptscriptstyle` respectively. Currently an explicit number is required; everything else is treated as an empty list. On the other hand, the dimension is scanned via the `scandimen` routine (the procedure that prints lines of the form 19-20).

- `\hbox` and friends. Currently, only `\hbox` is implemented. The current value of `\everyhbox` token list is inserted, and the argument is read. There are restrictions, see later.
- Case of `\mbox`, `\text`, `\makebox`. Like `\hbox`, but no `\everyXXX` token list is inserted. Example

```
\everyhbox{A}
\everymath{B}
\everydisplay{C}
[a=\hbox{bc d $ef g$}h i\text{OK}\]
```

Translation, in no-mathml mode.

```
<texmath type='display'>Ca=\text{Abc} \text{d} Bef gh i\text{OK}</texmath>
```

- Case of a math font, for instance `\@mathcal`. The font command is inserted in the token list, but the variable holding the current font is (locally) updated.
- Case of a math command (like `\alpha` listed as above). The command is read. There is a special hack: `\not\in` is converted to `\notin`, and `\not=` to `\ne`.
- Case of `\hspace`, `\vspace`. An argument, preceded by an optional space, is read. In the case of `\hspace`, a space is added, otherwise the command is ignored.
- `\kern`, `\mskip`, `\mkern`, `\hskip`, `\vskip`. See the example: on line 19 and 20, there is the trace of the routines that read the argument. The result is converted into a `\hspace`, with argument delimited by braces. Look at the trace, line 29. This will be read again later, see lines 30 to 32. In the case of `\vskip`, we should convert to `\vspace`, and re-insert, but the argument is ignored. In the case of `\mkern`, the result is converted into pt units, using the rule  $18\mu=10\text{pt}$ ; in the case of `\mskip`, the stretch and shrink parts of the glue are discarded.

- An apostrophe is handled in a special way, as explained in the  $\TeX$  book. Essentially  $x'$  is  $x^{\prime}$  and  $x'^2$  is  $x^{\prime 2}$ .
- Mode independent commands are interpreted as usual (this includes undefined commands). This should not typeset anything.
- A character is remembered, together with the current font.
- Mode-independent tokens are evaluated. These are commands like `\def` that change the environment, and have empty transation. For instance, `\relax` commands are handled here.
- Everything else is inserted without evaluation.

We give here an example with some fonts.

```

 $\mathhtt{Ab}\mathcal{Cd}\mathbf{Ef}\mathrm{Gh}\mathit{Ij}$ 
 $\mathbb{Kl}\mathsf{Mn}$ 

```

The translation is as follows. You can notice that some variants affect only uppercase letters.

```

<formula type='inline'>
  <math xmlns='http://www.w3.org/1998/Math/MathML'>
    <mrow>
      <mi mathvariant='monospace'>A</mi>
      <mi mathvariant='monospace'>b</mi>
      <mi>&Cscr;</mi>
      <mi>d</mi>
      <mi mathvariant='bold'>E</mi>
      <mi mathvariant='bold'>f</mi>
      <mi> G </mi>
      <mi> h </mi>
      <mi>I</mi>
      <mi>j</mi>
      <mi>&Kopf;</mi>
      <mi>l</mi>
      <mi mathvariant='sans-serif'>M</mi>
      <mi mathvariant='sans-serif'>n</mi>
    </mrow>
  </math>
</formula>

```

### 3.4 Translation of arrays

Whenever we see an array (this can be a global environment like `eqnarray` or a local one, like `array`), we translate all cells one after the other. The character `&` is the cell separator. The command `\\` is the row separator. In the case where an array ends with a `\\`, this gives an empty row: it will be removed. Each cell has an alignment, left, right, or center. An attribute is added only if this is not center. The `array` environment has an argument that explains the type of the columns (columns not indicated are centered). The default alignment is `'rl'` for `split` and `align`, `'rc'` for `eqnarray`, centered for `matrix`. You can use `\multicolumn`. This command takes three arguments: the span which should be some integer, then the alignment (one of `r`, `l` or `c`) and the content of the cell. The program may signal errors in case of wrong syntax. Here is an example:

```

 $\begin{array}{rc}
a&b&c&d\\$ 

```

```
A&\multicolumn{1}{r}{B}&C&D\\
\end{array}$
```

This is the translation of the array.

```
<table>
<tr>
<td columnalign='right'><mi>a</mi></td>
<td><mi>b</mi></td>
<td columnalign='left'><mi>c</mi></td>
<td><mi>d</mi></td>
</tr>
<tr>
<td columnalign='right'><mi>A</mi></td>
<td columnalign='right' colspan='1'><mi>B</mi></td>
<td columnalign='left'><mi>C</mi></td>
<td><mi>D</mi></td>
</tr>
</table>
```

### 3.5 Trivial math

If you say ‘ $x$ ’ and ‘ $123$ ’, the translation will be

```
<p><formula type='inline'><simplemath>x</simplemath></formula> and 123</p>
```

Initially, we found this a good idea; because this can easily be converted in HTML into `<i>x</i>`. Moreover ‘ $2^{\grave{e}}$ ’ gives

```
<temporary>2<hi rend='sup'>e</hi></temporary>
```

Here the `<temporary>` element will not show in the XML tree, but is printed on the terminal if Tralics is called with the ‘interactivemath’ switch. If you invoke Tralics with the ‘-notrivialmath’ switch, these hacks are not tried, and the formula translates into:

```
<formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<msup>
<mn>2</mn>
<mrow>
<mi>i</mi>
<mover accent='true'><mi>e</mi> <mo>&grave;</mo></mover>
<mi>m</mi>
<mi>e</mi>
</mrow>
</msup>
</math>
</formula>
```

There are three hacks: the first is when the formula contains only a letter, the second is when the formula contains only digits, and the last one is when people use a math formula instead of `\textsuperscript`. This hack is applied only if the math formula starts with digits (no digit at all is OK; braces are ignored) followed by an exponent marker, followed by a special exponent; this has to be a single token or a token list. In the case of a single token, the hack is applied only if this is `e` or `o`. Typically, it applies in cases like  $2^e$  and  $N^o$ . In the case of more than one token, it applies when the exponent is ‘`th`’, ‘`st`’, ‘`rd`’ and ‘`nd`’, for cases like  $1^{\text{st}}$ ,  $2^{\text{nd}}$ ,  $3^{\text{rd}}$ , and  $4^{\text{th}}$ . There are four rules for French: ‘`e`’, ‘`eme`’, ‘`ieme`’, ‘`eme`’ and ‘`ieme`’ convert to ‘`e`’, ‘`ier`’ and ‘`er`’ convert to

‘er’, ‘iemes’, ‘ièmes’ and ‘es’ convert to ‘es’, ‘ère’ and ‘re’ convert to ‘re’. The accented letter can be given as è, or \‘e or \{e} or \grave{e} or \grave e. The hack is applied in a case like:

```
$2 ~{\text{\small\rm \grave ere}} $
```

Instead of \text, \hbox can be used. Instead of \small or \rm any font change or font size command can be used. Up to two commands can be given. The original Perl version had 30 exceptions, including  $\Sigma^{\text{it}}$  and  $\text{\ddot{o}}$ . Compare  $\Sigma^{\text{it}}$  with  $\Sigma^{\text{it}}$  and  $\text{\ddot{o}}$  with  $\text{\ddot{o}}$ .

Since version 2.8, there is an integer register named \notrivialmath, that controls these hacks; it contains initially 1, it is set to zero if Tralics is called with the -notrivialmath switch, to seven if Tralics is called with the -trivialmath math switch (and to 349 if Tralics is called with -trivialmath=349). If the value is  $A + 2B + 4C$  modulo 8, where A, B, and C are zero (false) or one (true), then the behavior is the following (by default A is true, other flags are false).

- If A is true, in the case where the math formula contains optional digits followed by a special exponent, the rule explained above is applied; the special exponent can be one of th, rd, nd, st (for English), or those shown above for French.
- If B is true, some math formulas containing a single token produce a non-math result. We have shown above the translation of a digit or a letter. The translation of a minus sign is a en-dash (so that  $\$-\$$  is the same as --). Other characters are not considered trivial math. Most commands, whose translation is a single MathML element, including Greek letters, are converted into their content. Thus, the translation of  $\alpha$  is &alpha; rather than some long formula.
- If C is true, in the case where the formula starts with a hat or underscore, followed by a character, or a simple token list, and nothing more, then the result is a superscript, or subscript, out of math mode. It is as if you had used \textsuperscript or \textsubscript. A simple list is a list of characters, with an optional font change at the beginning. This rule has precedence over the first. Said otherwise,  $X^{\text{eme}}$  is translated as  $X^{\text{eme}}$ . Example

```
$1^e$, $3^{\text{eme}}$ X$^{\text{eme}}$ $4^{\text{i\grave{e}me}}$
$1^{\text{st}}$ $2^{\text{nd}}$ $3^{\text{rd}}$ $4^{\text{th}}$
$x$ $1$ $\alpha$ $\pm$ $\longleftarrow$ $-$
$_{\text{foo}}$ $^{\text{2+3}}$ $_{\text{bf Foo}}$
$+$ $x^{\text{eme}}$ $\log$ $_{\text{F\bf oo}}$
```

Translation (with MathML namespace removed), all hacks enabled:

```
<p>1<hi rend='sup'>e</hi>, 3<hi rend='sup'>e</hi>
  X<hi rend='sup'>eme</hi> 4<hi rend='sup'>e</hi>
1<hi rend='sup'>st</hi> 2<hi rend='sup'>nd</hi> 3<hi rend='sup'>rd</hi>
  4<hi rend='sup'>th</hi>
<formula type='inline'><simplemath>x</simplemath></formula>
  1 &alpha; &pm; &longleftarrow; &#x2013;
<hi rend='sub'>foo</hi> <hi rend='sup'>2+3</hi>
  <hi rend='sub'><hi rend='bold'>Foo</hi></hi>
<formula type='inline'><math><mo>+</mo></math></formula>
<formula type='inline'><math><msup><mi>x</mi>
  <mrow><mi>e</mi><mi>m</mi><mi>e</mi></mrow> </msup></math></formula>
<formula type='inline'><math><mo form='prefix'>log</mo></math></formula>
<formula type='inline'><math><msub><mrow>
  <mrow><mi>F</mi><mi mathvariant='bold'>o</mi>
  <mi mathvariant='bold'>o</mi></mrow> </msub></math>
</formula></p>
```

### 3.6 Conversion to XML

In the case where the value of the counter `\@nomathml` is negative, then the translation is a `<texmath>` element containing all tokens of the math list. For instance,

```
\csname@nomathml\endcsname=-1
$\begin{pmatrix}
\binom{12}{0}\int_0^{\infty} f(x)dx\ [2cm]
\mathfrak{W}_2\text{xyz}=\sqrt{xyyz}
\end{pmatrix}$
```

translates as

```
<p><texmath type='inline'>\begin{pmatrix}
\genfrac{}{0.0pt}{}{1}{2}&\int_0^{\infty} f(x)dx\ [2cm]
\@mathfrak W\@mathit _2&\text{xyz}=\sqrt{xyyz}
\end{pmatrix}</texmath></p>
```

In all other cases we use a highly recursive procedure that converts a math list into a formula. The procedure takes as argument the current style. This is one of D, T, S, or SS (display, text, script, or script script style). It is D for a display math formula, T for a normal formula.

Consider first the case where the formula has an `\over`, or a variant, not hidden inside braces. This example has 6 subexpressions, each of them have such an operator.

```
 ${a\over b}{a\above2mm b}{a\atop b}
 {a\overwithdelims[] b}{a\abovewithdelims[]2mm b}{a\atopwithdelims[] b}$
```

The translation is

```
<formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow>
<mfrac><mi>a</mi> <mi>b</mi></mfrac>
<mfrac linethickness='2mm'><mi>a</mi> <mi>b</mi></mfrac>
<mfrac linethickness='0.0pt'><mi>a</mi> <mi>b</mi></mfrac>
<mfenced open='[' close=']'>
<mfrac><mi>a</mi> <mi>b</mi></mfrac></mfenced>
<mfenced open='[' close=']'>
<mfrac linethickness='2mm'><mi>a</mi><mi>b</mi></mfrac></mfenced>
<mfenced open='[' close=']'>
<mfrac linethickness='0.0pt'><mi>a</mi> <mi>b</mi></mfrac></mfenced>
</mrow>
</math>
</formula>
```

It is an error if the formula has more than one such operators. Otherwise, we have two parts: what precedes the operator and what follows the operator. As the example shows, some operators need delimiters. Other operators read a dimension. This dimension must be given explicitly as a sequence of digits and a unit of measure (we could do better; if you want `\parindent` instead of `2mm`, you should use `\genfrac` instead). After splitting the formula into two parts, the same idea than `\genfrac` is used. If the current style is C, the next style in the list is used for both parts of the formula (if the style is D or T, the next style is S, otherwise it is SS). Note that `\choose` is like `\over`, you should use `\binom` instead.

We assume from now on that the formula contains no more operators like `\over`. This means that the current style can be used for the current object. Items are handled as follows:

- A space is ignored.

- If the current token is `\text`, `\hbox`, `\mbox`, this is a command with an argument, that is interpreted using special rules. A sequence of characters produces `\mtext`, a space produces a `\mspace`, and math formulas are allowed. Errors may be signaled if the content of the argument is too complicated. The translation of

```
$ x=0 \text{provided that $y=0$ or } a=1$
```

is

```
<formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow>
<mi>x</mi> <mo>=</mo> <mn>0</mn>
<mrow>
<mtext>provided</mtext>
<mspace width='0.5em' />
<mtext>that</mtext>
<mspace width='0.5em' />
<mrow> <mi>y</mi><mo>=</mo><mn>0</mn> </mrow>
<mspace width='0.5em' />
<mtext>or</mtext><mspace width='0.5em' /></mrow>
<mi>a</mi><mo>=</mo><mn>1</mn>
</mrow>
</math>
</formula>
```

- In the case of `$(\mathop{\rm sin})$`, the translation is `<mo form='prefix'>sin</mo>`. Any sequence of characters is allowed instead of 'sin'. Instead of `\rm`, any font change command that switches to 'rm' can be used.
- In the case of `\hspace`, an argument is read, converted to a dimension (in fact, a glue is read via the `scanglue` routine, the shrink and stretch parts of the glue are discarded), and the result is a `<mspace>` element. For instance `\hspace{2cm plus 3pt}` produces `<mspace width='56.9055pt' />`.
- In the case of `\displaystyle`, `\textstyle`, `\scriptstyle`, `\scriptscriptstyle`, the current style is changed, to D, T, S and SS respectively.
- In the case of `\nonscript`, the token is discarded if the style is D or T, kept otherwise. We shall see later that space disappears after such a token, if it is not discarded. Example.

```
$\def\foo{\nonscript~} \foo x^{y\foo}_{\textstyle z\foo}$
```

The translation is

```
<formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow>
<mspace width='3.3333pt' />
<msubsup>
<mi>x</mi>
<mstyle scriptlevel='0' displaystyle='false'>
<mrow><mi>z</mi><mspace width='3.3333pt' /></mrow>
</mstyle>
<mi>y</mi>
</msubsup>
</mrow>
```

```

    </math>
  </formula>

```

- If the token is a character of category code 7 or 8, it is left unchanged (typically, case of  $\hat{\phantom{x}}$  and  $\_$ ).
- If the token is `\limits`, `\nolimits`, `\mathord`, `\mathop`, `\mathbin`, `\mathrel`, `\mathopen`, `\mathclose`, `\mathpunct`, `\mathinner`, `\ensuremath`, `\nonumber`, `\nolinebreak`, it is ignored. However, we remember that the next object should be Rel, Bin (if `\mathrel` or `\mathbin` has been seen.)
- If the token is `\big`, `\bigl`, `\bigm`, `\bigr`, `\bigg`, `\biggl`, `\biggm`, `\biggr`, `\Big`, `\Bigl`, `\Bigm`, `\Bigr`, `\Bigg`, `\Biggl`, `\Biggm`, `\Biggr`, we remember that a big object is wanted, with subtype left, right, middle, or other.
- The current token or group is translated, according to the rules given below. After that, flags may be added (if the object is declared Bin or Rel or big). The current style is changed to the next style in the case we are in a group, and the group is preceded by  $\hat{\phantom{x}}$  or  $\_$ .
- If the current token is a character, its translation will be a `<mi>`, `<mn>` or `<mo>` element. In the case of a letter, this may depend on the font attribute associated to the character. For instance `{\bf x=1}` gives `<mi mathvariant='bold'>x</mi>` and `<mo>=</mo>` and `<mn>1</mn>`.
- If the current token is `\left`, `\right`, or already translated, it is left unchanged.
- If the token is a constant, like `\alpha`, see the big list at the start of the chapter, its XML value is inserted.
- If the token is a list, like `{...}`, it will be translated, using a copy of the current style.
- If the token is a list of the form `\left ... \right`, it will be translated. After that, fences will be added (using what follows the `\left` and `\right`).
- If the token is a list of the form `\begin{xxx}...\end{xxx}`, we assume that this is an array, or a matrix, we already explained how it can be translated.
- If the token is `\mathchoice`, with its four arguments, one of them is selected according to the mode. For instance

```

\def\foo{\mathchoice{1}{2}{3}{4}}
\foo{\displaystyle \foo}^{\foo^{\foo}}$

```

translates to

```

<mrow>
  <mn>2</mn>
  <msup>
    <mstyle scriptlevel='0' displaystyle='true'><mn>1</mn></mstyle>
  <msup>
    <mn>3</mn>
    <mn>4</mn>
  </msup>
</msup>
</mrow>

```

- It is an error if the current token is not a command of the form `\acute`, etc, or `\overline`, those listed at the end of the section 'basic objects' on page 92. As a general rule, for instance for `\frac`, arguments are translated using the next style (i.e., smaller), unless the style is indicated (for `\dffrac` and `\tfrac`, the style is T and S, the style may be indicated for `\genfrac`). If `\foo` is as above, the translation of

```


$$\frac{\frac{\text{foo}}{\text{foo}}}{\frac{\text{foo}}{\text{foo}}} = \frac{\frac{\text{foo}}{\text{foo}}}{\frac{\text{foo}}{\text{foo}}} = \frac{\text{foo}}{\text{foo}}$$


```

is

```

<mrow>
  <mstyle scriptlevel='0' displaystyle='false'>
    <mfrac><mn>3</mn> <mn>3</mn></mfrac>
  </mstyle>
  <mo>=</mo>
  <mstyle scriptlevel='0' displaystyle='true'>
    <mfrac><mn>2</mn> <mn>2</mn></mfrac>
  </mstyle>
  <mo>=</mo>
  <mfrac><mn>3</mn> <mn>3</mn></mfrac>
  <mstyle scriptlevel='0' displaystyle='true'>
    <mfrac><mn>2</mn> <mn>2</mn></mfrac>
  </mstyle>
</mrow>

```

The translation of

```

\def\xbar#1{\genfrac{}{}{#1}{\text{foo}}{\text{foo}}}

$$\frac{\frac{\text{foo}}{\text{foo}}}{\frac{\text{foo}}{\text{foo}}}$$


```

is the following. You can notice that, if the argument of `\xbar` is 2 or 3, this does not change the translation of the fraction. In  $\text{\TeX}$  we get two formulas that have the same size but are not vertically aligned (why?).

```

<mstyle scriptlevel='0' displaystyle='true'>
  <mfrac><mn>2</mn> <mn>2</mn></mfrac>
</mstyle>
<mstyle scriptlevel='0' displaystyle='false'>
  <mfrac><mn>3</mn> <mn>3</mn></mfrac>
</mstyle>
<mstyle scriptlevel='1' displaystyle='false'>
  <mfrac><mn>4</mn> <mn>4</mn></mfrac>
</mstyle>
<mstyle scriptlevel='2' displaystyle='false'>
  <mfrac><mn>4</mn> <mn>4</mn></mfrac>
</mstyle>
<mfrac><mn>3</mn> <mn>3</mn></mfrac>

```

As a final example, the translation of

```


$$\overline{x} \grave{y} \underbrace{z} \stackrel{a}{\underset{b}{\text{}}} \overset{a}{\underset{b}{\text{}}}$$


```

is

```

<mover accent='true'><mi>x</mi> <mo>&OverBar;</mo></mover>
<mover accent='true'><mi>y</mi> <mo>&grave;</mo></mover>
<munder accentunder='true'><mi>z</mi> <mo>&UnderBrace;</mo>
</munder><mover><mi>b</mi> <mi>a</mi></mover>
<mover><mi>b</mi> <mi>a</mi></mover>

```



### 3.7 Final math mode hacks

Before we forget it: when the formula is completely translated, we have a list of XML elements. If the list is empty, the result is `<mrow/>`. For instance, in the case of  $x^{\{}}$ , then exponent is empty. If the list has a single XML token, this will be the result. Otherwise, everything is put in a `<mrow>`. If the current formula, or subformula contains a style change, it is put in a `<mstyle>` element. This is not always the good solution, because the same style is used for everything, what precedes and what follows the style command. If you look at the `\genfrac` example above, you can see that styles are added by the `\genfrac` interpreter (the single T<sub>E</sub>X switch is associated with two MathML attributes).

If we have a formula, of the form  ${}_x^{\{2\}}_{\{abc\}}$ , the translation rules explained so far tell us that we have: an underscore character, an XML element for  $x$ , a hat character, an XML element for  $\{2\}$ , an underscore, and an XML element for  $\{abc\}$ . We may have `\nonscript` tokens; they will be removed, as well as a space that follows. We have to evaluate the commands that control subscripts and superscripts. A hat character gives `<msup>`, an underscore character gives `<msub>`, and both give `<msubsup>`. It is possible for a formula to start with an underscore or a hat: in this case, the kernel is empty. It is not possible for a formula to end with hat or underscore. A kernel can have at most one subscript and at most one superscript; hence the formula above is wrong: the letter  $x$  is the first subscript to the empty kernel. A valid formula is for instance  ${}_y x^2$ . It translates as

```
<mrow>
  <msub><mrow></mrow> <mi>y</mi> </msub>
  <msup><mi>x</mi> <mn>2</mn> </msup>
</mrow>
```

We have mentioned above that some operators can be flagged as left, right, and that adding `\bigl` may convert a left operator into a right operator. There is a magic that converts, in some cases, the `\big` operator into fences. For instance

```
 $\bigl [ A \big ( x^2 \big ) B \bigr [ \ $$ 
```

translates as

```
<mfenced open='[' close='['>
  <mi>A</mi>
  <mfenced open='(' close=')'><msup><mi>x</mi> <mn>2</mn> </msup></mfenced>
  <mi>B</mi>
</mfenced>
```

There is another trick, that works in some cases. Consider:

```
 $\int_0^{\infty} f(x) dx = \bigl [ U \big ]$ 
```

the translation is

```
<mrow>
  <msubsup><mo>&int;</mo> <mn>0</mn> <mi>&infin;</mi> </msubsup>
  <mrow>
    <mi>f</mi><mo>(</mo><mi>x</mi><mo>)</mo><mi>d</mi><mi>x</mi>
  </mrow>
  <mo>=</mo>
  <mfenced open='[' close=']'><mi>U</mi></mfenced>
</mrow>
```

The interesting point here is the placement of the inner `\mrow`. The idea is that the parentheses should remain small (not larger than the `\mrow`). In particular, it should not be influenced by the integral that precedes and the fence that follows. In some cases, it works.

### 3.8 Extensions

In Tralics, you can use the following three commands `\mathmo`, `\mathmi`, and `\mathmn`. They take an argument and produce a `<mo>`, `<mi>`, or `<mn>`. There is a file `tralics-iso.sty` that contains

```
\def\makecmd#1{\expandafter\newcommand\csname math#1\endcsname}
\def\makemo#1#2{\makecmd{#2}{\mathmo{\amp\##1;}}}
\def\makemi#1#2{\makecmd{#2}{\mathmi{\amp\##1;}}}
\def\makemn#1#2{\makecmd{#2}{\mathmn{\amp\##1;}}}
```

Then you can say `\makemo{x02190}{slarr}`, and this will define a command `\mathslarr`, whose translation (in math mode only) is `<mo>&#x02190;</mo>`. The file provides nearly 2000 such definitions, taken from the MathML entity files, with the MathML names. These commands can be used instead of  $\TeX$  commands like `\mathchar`: remember that a math-char is a 15bit integer, where 8 bits are used for the position in a font table, 3 bits for the type, and 4 bits for the family. Only three types are defined for Tralics, but the content of the element is arbitrary (most math symbols are between U+2100 and U+27FF, there are also letters between U+1D400 and U+1D7FF). There is a command `\mathattribute` that adds an attribute pair to the last created math element. You can say for instance

```
\providecommand\operatorname[1]{%
  \mathmo{#1}%
  \mathattribute{form}{prefix}%
  \mathattribute{movablelimits}{true}%
}
```

After that,

```
\min _xf(x) >\operatorname{min} _xf(x)
```

translates as

```
<formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow>
<msub><mo movablelimits='true' form='prefix'>min</mo> <mi>x</mi> </msub>
<mrow>
<mi>f</mi> <mo></mo> <mi>x</mi> <mo></mo> <mo>&gt;</mo>
</mrow>
<msub><mo movablelimits='true' form='prefix'>min</mo> <mi>x</mi> </msub>
<mrow>
<mi>f</mi> <mo></mo> <mi>x</mi> <mo></mo>
</mrow>
</mrow>
</math>
</formula>
```

The command `\DeclareMathOperator` takes two arguments (say ‘foo’ and ‘bar’), with an optional star before the first argument. It defines `\foo` to be the command `\operatorname` applied to ‘bar’ (with a star when required). The command `\operatorname` is as shown above (the `movablelimits` attribute is only added if the command is followed by a star).

You can use the command `\mathchardef`. This is like `\chardef`, it reads a command and a number. The number should fit on 15 bits. Otherwise, you will see an error of the form: *Bad mathchar replaced by 0: 1234567*. The `\mathchardef` command reads a command, say `\foo`, and an integer `N`; there is no difference between `\foo` and `\mathcharN`, except that `\the\foo` returns the integer `N`, and is faster to parse. Some constants, like `\@cc1vi=256`, are defined in this way by the  $\TeX$  kernel and should not be used as math characters. Some commands, like `\eta=11116`, are meant to be used as a math character. In Tralics, until version 2.8 an error will be signaled.

In version 2.9, the translation, in math mode, is a `<mi>` element containing this character; you might say `\mathchardef\eta"3B7`. Outside math mode, this gives an error: that takes the form *Undefined command \eta; command code = 264, instead of Math only command \theta. Missing dollar not inserted*; inside math mode, the behavior is the same as the standard one.

T<sub>E</sub>X has a special register called `\fam`. If you say something like

```
\fam3 ${\fam9 \the\fam}\ \the\fam$
```

then the second `\the` expands to minus one. The first gives 9, but L<sup>A</sup>T<sub>E</sub>X complains with: *\textfont 9 is undefined (character 9)*. In Tralics, you would see

```
<mrow><mn>9</mn><mspace width='6pt' /><mn>3</mn></mrow>
```

As the example shows, the family is unused, and not correctly restored. Each character has a `\mathcode`. The following

```
\mathcode'\a="0941 $a\the \mathcode'\a$
```

is interpreted by Tralics as `$a2369$`. However T<sub>E</sub>X complains, with *\textfont 9 is undefined (character A)*, because you ask the lower case letter a to be printed like the upper case letter A with textfont 9. A mathcode is a 15bit integer, with an exception: a character whose mathcode is 32768 behaves like an active character, the action associated to it must be defined somehow, for instance like this:

```
{\catcode'\=' \active \global\let'\active@math@prime}
```

There is a command `\delimiter`, it reads a number, but you cannot use it. There is a command `\radical`, it reads a number, then signals an error. The `\mathaccent` command is similar.

There are commands `\raise` and `\lower`, as well as `\vcenter`. The last one is not implemented in Tralics. The translation of

```
a\raise2cm\box{foo}{bar}\lower 2pt\box{xfoo}{xbar}
```

is

```
<p>a<foo>bar</foo><xfoo>xbar</xfoo></p>
```

As you can see, the specification disappear. Maybe in a future version, we will add an attribute to the box. You cannot use these commands in math mode in Tralics. In T<sub>E</sub>X, you can get an error of the form: *You can't use '\raise' in vertical mode*, while `\vcenter` is a math only command. Currently `\indent` and `\noindent` are ignored in math mode (in T<sub>E</sub>X `$\indent_b$` produces a kernel and an index; the kernel is an empty box of width `\parindent`, of type Ord).



## Chapter 4

# Translating a bibliography

### 4.1 Introduction

As said in [6], “citations are cross-references to bibliographical information outside the current document, such as to publications containing further information on a subject and source information about used quotations. [...] There are numerous ways to compile bibliographies and reference lists. They can be prepared manually, if necessary, but usually they are automatically generated from a database containing bibliographic information.”

There are different ways to cite an author, or a text or a specific part of a text. The easiest way (for an automated system) is to use numbers, as above; if you are reading an interactive version of this document, you can click on the number, and you will see the entry in the bibliography, at the end of the document (between the index and the table of contents). This is standard practice; recommendations for a book series say: *References are cited in the text simply as numbers in square brackets, e.g. [165], do not use the abbreviations “Ref./Refs/” in the middle of a sentence. Only at the beginning of a sentence should you write “Reference [165]”*. In some cases, you can see ‘[17, p23]’, as the result of ‘`\cite[p23]{foo}`’; this means page 23 of the reference numbered 17. A bit more sophisticated are references like ‘[GMS93]’ instead of ‘[2]’ for a book by Goosens, Mittelbach and Samarin published in 1993. Computing the key is not obvious, because, if you cite a book by, say, Goethe, Molière and Shakespeare in 1793, the key will be the same, and a post-processor has to add a suffix (typically, this is done by a couple of routines named `forward.pass` and `reverse.pass` in a `bst` file). Sometimes, a more explicit scheme is used, for instance ‘Knuth, The Art Of ..., Algorithm P’, in the text, and the full reference can be found in the bibliography. A text of R. Ridolfi can be cited as ‘*Vita di Girolamo Savonarola*, 5<sup>e</sup> éd, Florence, 1974, t. II., p. 182-183’. Note that the name of the author appears before the citation, and is not repeated inside it. In some books, citations are given as footnotes, and you can often see ‘*ibid.*’, meaning the previous cited text. These kinds of things are generally hard to fully automate. For this reason, only a simple scheme is provided by Tralics: a link to a bibliography section via a key.

The problem is essentially the following: The L<sup>A</sup>T<sub>E</sub>X source file contains a given number of citations, introduced by the `\cite` command or a variant. Each command defines one or more references. For each reference, a key has to be computed and typeset, an item added to the bibliography, and a link created. In L<sup>A</sup>T<sub>E</sub>X, the document has in general to be processed three times; the first run will print `\citation{companion2}` in the auxiliary file. This file is processed by BibT<sub>E</sub>X, that generates a `bb` file containing `\bibitem{companion2}`. On the second run, the bibliography is typeset, and the key is constructed; if it is 6, then `\bibcite{companion2}{6}` will be printed in the auxiliary file. On the last run, we know, after reading the auxiliary file, that the `\cite` command should typeset as 6.

The mechanism in *Tralics* is a bit different: there is only one run. Each `\cite` command produces a `<cit>` element, plus an entry into a biblist. At the end of the document, the bibliography is constructed, with all the necessary entries; details will be given later. This gives the equivalent of a `bbl` file, it is translated. The result of the translation is some XML element, that will be inserted somewhere in the main XML tree. Finally, a check is made to see if all references are defined. The mechanism is much simpler than in *L<sup>A</sup>T<sub>E</sub>X*; this is really because, in *Tralics*, you can add an element or an attribute anywhere in the tree (at the start if you like) at any moment. In *T<sub>E</sub>X*, on the contrary, once a paragraph is typeset, you cannot modify it, and once a page is shipped out, you cannot modify the whatsits associated to it (pages numbers, in the case of `\label`, `\ref` are computed only when the page is shipped out; they are left in a `\write`, which is a special kind of whatsit).

There are some tentatives to design an XML format for bibliography data bases; none of them is really satisfactory. We give an example of an entry using the DocBook syntax:

```
<biblioentry id="abc123" type="book">
  <author>
    <surname>Flynn</surname>
    <firstname>Peter</firstname>
  </author>
  <title>Understanding SGML and XML Tools</title>
  <titleabbrev>SGML & XML Tools</titleabbrev>
  <publisher>
    <publishername>Kluwer</publishername>
    <address>Boston</address>
  </publisher>
  <isbn>0-7923-8169-6</isbn>
  <date YYYY-MM-DD="1998">1998</date>
</biblioentry>
```

This should be referenced in the text as:

```
<citation><biblioref linkend="abc123"/></citation>
```

This is the same using the TEI syntax:

```
<biblFull id="abc123" rend="book">
  <titleStmt>
    <title>Understanding SGML and XML Tools</title>
  <author>
    <persName>
      <foreName>Peter</foreName>
      <surname>Flynn</surname>
    </persName>
  </author>
  <respStmt>
    <name>http://imbolc.ucc.ie/~pflynn/books</name>
  </respStmt>
</titleStmt>
<extent>432</extent>
<publicationStmt>
  <publisher>Kluwer Academic Publishers</publisher>
  <pubPlace>Boston</pubPlace>
  <idno type="isbn">0-7923-8169-6</idno>
  <date value="1998">1998</date>
```

```

    </publicationStmt>
  </biblFull>

```

This should be referenced in the text as:

```
<cit><ref target="abc123"></ref></cit>
```

These two citations were found on the Web<sup>1</sup>. The careful reader may notice that two elements are used for the citation (in the DocBook case, they are `<citation>` and `<biblioref>`, in the TEI case, they are `<cit>` and `<ref>`). Tralics uses the TEI syntax for the citation but a completely different one for the entries (the syntax is very near to BibT<sub>E</sub>X). We shall explain, in the second part of this document, Chapter 6, how to convert the Tralics DTD into the TEI DTD (at least for the bibliography). The transformation is incomplete: in BibT<sub>E</sub>X, a name has four components, and the example shows only two, surname and forName (or firstname). A non-trivial question concerns mathematics: how can we insert math formulas like  $H^\infty$  and what about special words like: “the T<sub>E</sub>Xbook?”. The main reason why Tralics does not read databases written in XML is the need of an XML parser (we have written a BibT<sub>E</sub>X parser, this is more challenging).

The interaction between the main document and the bibliography is via the ‘cite key’ on the L<sup>A</sup>T<sub>E</sub>X level, in the XML document, this is via the Bid attribute, and for the typeset document, this is the ‘print key’. As an example, we shall consider a bbl file, created by Tralics, that contains

```
\citation{60}{footcite:thesefabien}{bid9}{foot}{phdthesis}[Sey98] ...
```

This is a temporary piece of stuff, the cite key is ‘thesefabien’, the Bid is ‘bid9’, there are two choices for the print key, ‘60’ or ‘Sey98’. The XML translation is

```
<citation from='foot' key='60' id='bid9' userid='footcite:thesefabien'
  type='phdthesis'> ...
```

As you can see, the effective print key is ‘60’. We shall explain in due time all details. Let’s start with the cite key, the only quantity that the author can choose freely. For the references from the web, this key is ‘abc123’, this is clearly a randomly chosen value, not mnemonic at all. At the start of the chapter, we have shown a reference with key ‘companion2’, this is the cite key for the second version of the L<sup>A</sup>T<sub>E</sub>X companion. The cite key ‘thesefabien’ is for the Ph.D. thesis of F. Seyfert. There is no constraint on the cite key for L<sup>A</sup>T<sub>E</sub>X: the only important thing is that the key can be printed in the auxiliary file and read by BibT<sub>E</sub>X (some years ago, a colleague corrected `\cite{Christele}` to `\cite{Christèle}`, this gave an awful error; in current L<sup>A</sup>T<sub>E</sub>X, there seems to be no problem). On the other hand, BibT<sub>E</sub>X needs an identifier. This is a character string that does not start with a digit, and contains anything but space, tabulation, double quote, percent sign, sharp sign, backslash, comma, equals sign, braces, parentheses. For XML, there are additional constraints for an ID: it has to be unique for the whole document, and some characters like a plus sign are forbidden. In a first version, we imagined to use the ‘userid’: this is formed of a prefix (of the form ‘cite:’ or ‘footcite:’, thus making it unique), followed by the cite key, where forbidden characters like the plus sign were replaced by a minus sign. However, we found an example where a rather long key differed from another one only by a forbidden character. Replacement introduced a conflict. For this reason, we added the Bid: this is automatically generated, hence is clearly unique and valid. A special feature of BibT<sub>E</sub>X is that it does not create lines longer than 78 characters. It adds percent characters in a sensible position; in some cases, the choice is wrong. Here is an example:

```
\bibitem[13]{Bergamini-Champelovier-Descoubes-Garavel-Mateescu-Serwe-04-a}\RAS%
c{D.~Bergamini, D.~Champelovier, N.~Descoubes, H.~Garavel, R.~Mateescu,
  W.~Serwe},
```

As a result, you will get an error: *Undefined control sequence \RAS*. Note that there are few people who use such very long cite keys. A simple idea that works most of the time: use 4 letters for the first author, three letters for the others, two letters for the year, for instance ‘Bara-Chy-Pom02’.

<sup>1</sup>From the web page of BiblioX: <http://www.silmaril.ie/bibliox/biblioxdoc.html>

As explained, Tralics cannot use an XML database; instead it will use a `bbl` file (this is some  $\LaTeX$  file, that will be translated by Tralics). The `bbl` can be part of the source document; in general it will be automatically constructed by Tralics (in the current version, Bib $\TeX$  or any other external program can be used instead). This `bbl` file should contain, for each unsolved citation, a command that solves it (either `\citation` which is a Tralics command, or `\bibitem` which is a standard  $\LaTeX$  command, see section 4.2).

One question is: can the `bbl` contains other items, together with these `\bibitem` commands? If the bibliography is very long, it can be interesting to divide it into subsections, and add a comment at the start of each section; this is easy to do, if the `bbl` is not produced by Bib $\TeX$ , or if you edit it, and if you know how to convince  $\LaTeX$  not to start the bibliography with a `\bibitem`. In general, we have a unique `\begin{thebibliography}` at the start, a `\end{thebibliography}` at the end. The effect is to produce a chapter (or a section), in general unnumbered, whose name depends on the current language. In the case of the Raweb, Bib $\TeX$  produces more than one such environments. In fact, three databases are used: ‘foot’, ‘refer’ and ‘year’. Each of the two database files ‘foot’ and ‘refer’ produce a set of references (the ‘foot’ bibliography was originally typeset as footnotes, via the `footcite` package). The third database produces a sequence of sections, such as theses, books, articles, conferences, reports, etc.<sup>2</sup> Whenever Bib $\TeX$  sees an entry with a different category than the preceding entry, it prints the `\end{thebibliography}` followed by a `\begin{thebibliography}`. Note: the modified environment takes a required argument (as usual, the longest label) and an optional argument (the name of the section title; the title itself being in the class file). As a consequence, the `bbl` files produced by the Raweb are incompatible with standard  $\LaTeX$  classes. Since year 2001, Bib $\TeX$  is not used anymore for the Raweb and the XML result contains just a sequence of references. However, each entry has a category (this depends on the `from` and `type` attributes), entries are sorted by category. The style sheets that convert the XML to HTML or XSL/Format are assumed to create these sections, one for each category (see part two of this document). A nontrivial question is then to guarantee that these two style sheets use the same splitting algorithm, and the same section titles.

The ‘print key’ is the value that is printed on the paper or displayed on the screen. Each `<citation>` has a `key` attribute that can be used as print key. However, an XML processor may as well ignore it, and use numbers 1, 2, 3, etc. It can even sort the entries, before assigning them a number<sup>3</sup> (see part two of this document). In some cases, Tralics computes a symbolic key of the form ‘Sey98’. If the post-processor sorts the entries, and if the keys are not in alphabetic order, this is a bad idea.

The ‘key’ of an entry is a quantity defined in the database, whose purpose is to help sorting. In most cases, it is empty, (in some cases the values are junk); this value is used only in the case where no author is given (this is standard Bib $\TeX$  practice, it means that this is rather useless). The ‘sort key’ of an entry is the character string used for sorting (this is `lost`<sup>4</sup>; Tralics could insert it in the resulting XML; this would allow one to merge two bibliographies). In some cases, the print key is part of the sort key. Imagine for instance a book by Samarin, Mittelbach and Goossens, written in 1993. The standard key would be GMS93. Assume however that the authors are taken in the given order, so that the key would be ‘SMG93’. Alphabetically, this is after ‘Sey98’, but if we sort by authors, Samarin comes before Seyfert.

## 4.2 Citing a document

In this paragraph, we shall explain the commands that can be put in the source document for inserting a citation, and the companion commands that solve the reference. When the `\end{document}`

<sup>2</sup>Exact ordering on page 118.

<sup>3</sup>Sorting after assigning a number is weird.

<sup>4</sup>Exemples on lines 135 to 148 below were produced with a hacked version of Tralics and bst files



command is about to be translated, Tralics will have created a big list (maybe empty) called the ‘biblist’. Each item in the list has four slots: Reference, Rtype, Bid and Definition. Here Reference is the cite key, Rtype is a subtype (when merged, these two quantities give the ‘userid’; this subtype is not standard L<sup>A</sup>T<sub>E</sub>X, you can ignore it. In some cases, two items with the same Reference and different Rtype are considered unequal, in some cases they are considered equal; thus, it is a bad idea to use the same cite key with different subtypes). The Bid is the unique id of the target, of the form ‘bid17’, and Definition is the internal number of the target of the reference (in Tralics, each XML element has an internal number). You can say: element number 25 is the target of reference ‘foo’ (syntax described later). This will solve the entry: If the entry with key foo has Bid 17, the action is to mark the entry as solved, and to add `id='bid17'` to the element number 25. When the end of the document is sensed, the list of unsolved entries is computed, and a request is made for constructing a bbl. A warning or an error is signaled for missing items by this construction. This bbl is then translated. It is forbidden to add unresolved entries to the list. In BibT<sub>E</sub>X, there is cross reference mechanism: if X has a cross reference to Y, then X must become before Y; when Y is read, its fields are used to fill missing fields in X. Unless cited explicitly, Y will not appear in the bibliography.

The variable `distinguish_refer_in_rabib` was introduced in 2006. Since this is a long name, we shall abbreviate it to DRY. If it is true, we distinguish ‘year’ and ‘refer’, otherwise there is no distinction. By default the flag is true, you can set it on the command line, or a configuration file. For the case of the Raweb, three Rtypes are defined, ‘foot’, ‘year’ and ‘refer’. There is one command, `\footcite`, to cite elements with Rtype ‘foot’ and a command, `\cite`, for anything else. We generalized this mechanism: for all commands described here, there is no difference between ‘year’ and an empty Rtype. If DRY is false, the ‘refer’ is the same as ‘year’. In 2006, commands `\yearcite` and `\refercite` have been introduced. If DRY is false, these two commands behave the same.

The translation of ‘`\footcite {Knuth}`’ or ‘`\footcite [p.25] {Knuth}`’ is the same as ‘`\cite [foot] [] {Knuth}`’ or ‘`\cite [foot] [p.25] {Knuth}`’. The translation of ‘`\yearcite {Knuth}`’ or ‘`\refercite {Knuth}`’ is the same as ‘`\cite [year] [] {Knuth}`’ or ‘`\cite [refer] [] {Knuth}`’. These commands have an optional argument. The `\cite` command has two optional arguments, a type and an optional value. If only one optional argument is given, it is the value (so that ‘`\cite [p.25] {Knuth}`’ has the same meaning as in L<sup>A</sup>T<sub>E</sub>X). The translation of ‘`\cite [x] [y] {z}`’ is the same as ‘`\cite@one {x} {z} {y}`’ (note the order of the arguments). However, if you say ‘`\cite [p.25] {Knuth,Lamport}`’, the result is the same as ‘`\cite@one {} {Knuth} {p.25}`’, followed by ‘`\cite@one {} {Lamport} {}`’, said otherwise, the second optional argument applies only to the first citation. Between two `\cite@one` commands (that come from the same `\cite`) are inserted some `\citepunct` tokens. This is a command that can be redefined by the user. Its expansion is a comma followed by a space.

People generally say ‘`Text\footcite{blah}`’, like ‘`Text\footnote{blah}`’, without any space, because `\footcite` is assumed to produce a footnote; but this is not always the case; for this reason, the command `\footcitepre` is evaluated before insertion of the XML element associated to the citation. The default behavior is the following: if the last object on the XML tree is a normal or non-breaking space, nothing happens; otherwise, if the object is not an opening parenthesis, a space will be added. Moreover, the `\citepunct` is replaced by `\footcitesep`, a command whose translation is comma space (the idea is that you can redefine it, so that ‘`Text\footcite{foo,bar}`’ shows as ‘`Text\textsuperscript{13,15}`’, exercise left to the reader). This is a slight difference between `\footcite` and `\cite` with ‘foot’ as optional argument.

The command `\nocite` can take one optional argument (a Rtype). The effect of `\nocite{foo}` is the same as `\cite`, regarding the biblist, but it does not modify the XML tree. If you say `\nocite{*}`, this inserts a special marker, meaning: the whole database should be inserted. The Rtype is ignored in this case. Note that the correct behavior should be: Rtype is ignored only if one of ‘year’, ‘refer’ or ‘foot’.

In order to implement the `natbib` package, we make the following assumptions. The primitive command is `\cite@one`, it takes a single reference (defined by a Reference and a Rtype), inserts when needed a new item in the biblist, and construct a Bid for the reference. The command calls `\leavevmode`, for the case where it appears at the start of a paragraph (Remember the recommendations given above: a paragraph should start with a word, not a reference). The result of the translation is `<ref target='bid17' />`, where 'bid17' should be replaced by the value of the Bid. This element can be non-empty (it contains a note), and is the child of `<cit>` element, that has some attributes. The L<sup>A</sup>T<sub>E</sub>X companion, example 12-3-5, says that `\citet {LGC97}` should produce 'Goossens et al. (1997)'. The translation by Tralics does not contain the name nor the year, so that there should be an attribute that says how parentheses are to be inserted in the final HTML or Pdf document. Another example is `\citep [see] [chap. 2] {LGC97}`, this produces '(see Gossens et al., 1997, chap. 2)'. This does not really fit in our model: we can put the post-note in the `<ref>` element, and the pre-note as an attribute. This makes these two quantities asymmetric: the pre-note must contain only characters. Consider now example 12-3-15, `\citet [cf.] [p. 55] {vLeunen:92, Knuth-CT-a}`. Here the pre-note is added to each citation, the post-note to the last one (the default is to put the single note on the first element). The result is 'van Leunen (cf. 92); Knuth (cf. 1986, p.55)'. What Tralics should do in such a case is unclear. The file `natbib.plt` defines `\citeyear` and `\citeyearpar` as follows

```
\def\cite@type#1#2{{\def\cite@type{#1}\cite{#2}}
\def\citeyear{\cite@type{year}}
\def\citeyearpar{\cite@type{yearpar}}
```

The idea is to call `\cite`, the dispatcher function, and to put locally in `\cite@type` the type of the citation (year, or parenthesized year). There is also `\cite@prenote` for the prenote. To be precise: the translation of `\cite@one {bar} {foo} {p25}` is `<cit rend='bar' type='mtype' prenote='mynote' ><ref target='bid17' /> p25</cit>`, where 'mtype' is the value of `\cite@type`, 'mynote' is the value of `\cite@prenote`. Arguments 'foo' and 'bar' define the reference (normally, the Rtype 'foo' is empty).

You can say `\XMLsolvecite*[25] [bar] {foo}`. The star is optional, as well as the '25' and the 'bar'. If only one optional argument is given it is the first one. This should be the identifier of an XML element (you can use `\XMLlastid`, the identifier of the last created element, or `\XMLcurrentid`, the identifier of the current element). The current element is used if the argument is missing or empty. In any case, this gives an element, say Target. The second optional argument is the Rtype. The required argument is the cite key. The result of the command is to solve the entry defined by the Reference and the Rtype. The easy case is when the reference has not yet been cited. In this case, we can use as Bid either the id of the Target, if it exists, or a new id. In this case, an attribute pair `id='Bid'` is added to the Target. If the entry exists in the biblist, it might be already solved, and you get an error of the form *Already solved foo*. An attribute pair `id='Bid'` is added to the Target, unless the Target has already an id, case where an error will be signaled, for instance *Cannot solve (element has an Id) foo* in the case

```
\cite{foo}\section{something}\XMLsolvecite{foo}
```

The problem here is the following: the section element has a Uid, this is like a Bid, it can be used as target of a `\label`. The XML norm forbids using two ids for the same element. Maybe in a future version, this will be allowed (it suffices to implement a double indirection mechanism). However, I doubt if this is a good idea: if you say `\label{foo}`, then `\ref{foo}` will produce a `<ref>` element, this is identical to the `<ref>` that comes from the `\cite`. Note that the Raweb DTD says: the target of a `<ref>` in a `<cit>` should be a `<citation>`.

If a star is given in `\XMLsolvecite`, there is a little hack. If Reference/Rtype is not found in the biblist, Tralics tries to see if there is an unsolved entry with the same Reference, Rtype arbitrary. In such a case, this entry will be solved. If there is no such entry, then a new slot is added to the reference list.

Some commands may produce strange results. Consider

```
\setbox0 =\hbox{\XMLsolvecite{foo}} \copy0 \copy0
\setbox1 =\xbox{Box}{\XMLsolvecite{bar}} \copy1 \copy1
```

This constructs two empty boxes, with an id bid0 and bid1. Since the first box is unnamed, the tag will not appear in the XML tree; and no tag implies no attribute list, so that the first line is an error. On the other hand, the second box is copied twice; hence the id bid1 appears twice in the XML tree, this is also an error (the XML is well-formed, but not valid against any DTD that says that the Bid should be an ID).

You can say `\bibitem[XX]{foo}`, the result is the same as `\par \leavevmode \XMLsolvecite*{foo}`. The optional argument is ignored. Note that the `\par` command terminates the current paragraph, and `\leavevmode` starts a new paragraph (in L<sup>A</sup>T<sub>E</sub>X, `\bibitem` calls `\item` that does more or less the same thing). The important point is that this newly created `<p>` element is the target of the reference. If you feed Tralics with the bbl of this document, produced by L<sup>A</sup>T<sub>E</sub>X, you will see something like

```
<Bibliography><p id='bid0'>
David Carlisle, Michel Goossens, and Sebastian Rahtz.
De XML à PDF avec <hi rend='tt'>xmltex</hi> et Passive<TeX/>.
In <hi rend='it'>Cahiers Gutenberg</hi>, number 35-36, pages 79&ndash;114,
2000. </p>
<p id='bid1'>
Michel Goossens, Frank Mittelbach, and Alexander Samarin.
<hi rend='it'>The <LaTeX/> companion</hi>.
Addison Wesley, 1993.</p>
```

On the other hand, translation of the second reference is:

```
<citation from='year' key='GMS93' id='bid4' userid='cite:companion'
type='book'>
<bauteurs><bpers prenom='M.' nom='Goossens' prenomcomplet='Michel' />
<bpers prenom='F.' nom='Mittelbach' prenomcomplet='Frank' />
<bpers prenom='A.' nom='Samarin' prenomcomplet='Alexander' /></bauteurs>
<btitle>The <LaTeX/> companion</btitle>
<bpublisher>Addison Wesley</bpublisher>
<byear>1993</byear>
</citation>
```

### 4.3 Using Tralics instead of BibT<sub>E</sub>X

The content of the BibT<sub>E</sub>X database is a sequence of entries of the form

```
1 @article{example,
2   Author= "Joseph Garrigue and Didier R{\`e}my",
3   Title=  "Extending {ML} with semi-explicit higher-order polymorphism",
4   Number= "1/2",
5   Volume= 155,
6   Year=   1999,
7   Pages=  "134-169",
8   Journal= "Journal of Functional Programming",
9   Remark=  {a random example},
10  OptMonth = jan,
11  Url=     "ftp://ftp.inria.fr/INRIA/Projects/Cristal/iandc.ps.gz"}
This is a second example.
```

```

12 @PhdThesis{thesefabien,
13   author =      {Seyfert, Fabien},
14   title =      {Problèmes extrémaux dans les espaces de Hardy,
15     Application à l'identification de filtres hyperfréquences à
16     cavités couplées},
17   school =     {Ecole de Mines de Paris},
18   year =      1998
19 }

```

These examples are translated by Bib<sub>T</sub>E<sub>X</sub> as follows

```

\bibitem{example}
Joseph Garrigue and Didier R{\`e}my.
\newblock Extending {ML} with semi-explicit higher-order polymorphism.
\newblock {\em Journal of Functional Programming}, 155(1/2):134--169, 1999.

\bibitem{thesefabien}
Fabien Seyfert.
\newblock {\em Problèmes extrémaux dans les espaces de Hardy, Application à
  l'identification de filtres hyperfréquences à cavités couplées}.
\newblock PhD thesis, Ecole de Mines de Paris, 1998.

```

After the @ character, there is a keyword, or an entry type. The recognized entry types are article, book, booklet, conference, coursnotes, inbook, incollection, manual, masterthesis, misc, phdthesis, techreport, unpublished, as well as mastersthesis, a synonym of masterthesis. These types are not part of the Bib<sub>T</sub>E<sub>X</sub> language, but are described in any good book about L<sup>A</sup>T<sub>E</sub>X, they are the only ones recognized by Tralics. The case is irrelevant (in one example, we have ‘article’ in lower case, in the other, we have ‘PhdThesis’, mixed case). Since Tralics2.9.1, you can extend the list of known types, by putting a line like the following in the configuration file (this will define the types ‘hdr’ and ‘movie’):

```
bibtex_extensions = "hdr movie"
```

There are three keywords. The first is ‘comment’. If you say @comment{foo}, this makes ‘foo’ a comment. Since everything outside the scope of a keyword or an entry is discarded, there is no real need for a comment keyword, or a comment character. In particular, the percent sign is not a comment character inside a Bib<sub>T</sub>E<sub>X</sub> file. If you insert a percent sign in a field, you have to remember that Bib<sub>T</sub>E<sub>X</sub> will replace newline characters by spaces, and insert newline characters in the bbl file wherever it judges adequate. Hence, the percent character will behave, in the bbl, as a comment character with a random scope.

The second keyword is ‘string’. It defines a string, for instance @string{Foo="bar"} defines the string ‘foo’ (the case is irrelevant) with value ‘bar’. In the example, there is a string after the equals sign, but any expression could be used, including one that uses macros. A macro must be defined before its use; it is always possible to redefine the macro. There are 12 predefined macros; there are jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec,. You can see a use of ‘jan’ on line 10. These macros are defined by every bst file, to be ‘janvier’, ‘January’ or ‘Januar’, depending on the language (since there is no way to tell Bib<sub>T</sub>E<sub>X</sub> what the current language is, there are two solutions: either write frplain.bst, that is a copy of plain.bst, with all keywords translated into French, or use indirection: the value is \bbljan{ }, a L<sup>A</sup>T<sub>E</sub>X command defined in a style file depending on the current language. In Tralics, these strings are defined at bootstrap, to be English names, and redefined when \begin{document} is seen. This gives you a chance to select the correct language. Only Frech, English and German are known languages.

The last keyword is ‘preamble’. If you say @preamble{"foo"}, the effect is to add the string ‘foo’ to the preamble. More than one preamble keyword can be given, they will be merged, in order. Standard bibliography styles print the preamble at the start of the bbl file, just before

the `\begin{thebibliography}`. In Tralics, the string is inserted at the start of the file, but the environment is implicit. The string should not produce text, otherwise strange errors are signaled, of the form *Error signaled at line 4: Non-empty buffer foo Some text may be lost.*, because only bibliographic entries are allowed in the bbl; you can cheat by changing the current mode via `\@setmode`. Instead of "foo", a general value can be used, for instance `@preamble{jan}` puts January in the preamble. *Note:* Instead of braces, you can use parentheses to delimit the value of an entry, a string or the preamble. Inside the value, you can use braces instead of double quotes. Thus `@preamble({foo})` is a valid preamble.

After an entry type comes the cite key, followed by a sequence of pairs, of the form field=value, separated by commas. The following field names are recognized: address, author, booktitle, chapter, crossref, doi, edition, editor, howpublished, institution, isbn, isrn, issn, journal, key, month, note, number, organization, pages, publisher, school, series, title, type, url, volume, year. The case is irrelevant. If a field name is given, whose value is not in the previous list, it will be ignored. In the example, line 10, we have an unused field, 'OptMonth' (some text editors propose templates where optional fields like 'month' are preceded by 'opt', and sometimes people forget to remove the prefix). In the first example, field names start with an initial capital, and there is no space on the left of the equals sign, in the second, field names are all lower case, there is a space on the left of the equals sign, and opening braces are vertically aligned (this is the template proposed by Emacs); these subtleties are ignored by Tralics.

If the configuration file contains line like

```
bibtex_fields = "firstpage lastpage"
bibtex_fields = "+allpages"
```

then three additional fields are read by Tralics, namely firstpage, lastpage, and allpages. They will be inserted in the XML tree, after other fields, but before the 'note', via a call to `\cititem`.

The value of a field can be a number (lines 5 and 6 in the example), or a macro name (as on line 10), or a constant in braces (line 9), or a constant in double quotes (other lines). It is possible to concatenate basic fields, for instance `apr # "-1"`, via the use of the sharp operator. The way BibT<sub>E</sub>X handles braces, quotes and backslashes is a bit special. When BibT<sub>E</sub>X parses a value, there should be as many opening braces than closing braces; trying to put a backslash before a brace has no effect<sup>5</sup>. If a string is delimited by double quotes, then braces are needed to hide double quotes. Special characters should be entered as `{\e}`, never as `\{e}`, but Tralics accepts `é`; in fact, any Unicode character is accepted, provided that you declare the proper encoding. The case of a non-ascii character is undefined. When looking for a particle in a name, Tralics must decide whether a character is upper case or not, and when sorting, the whole string is converted into lower case letters. In the case of `{\e}`, the whole group is converted by BibT<sub>E</sub>X to the single letter e; Tralics leaves it unchanged; in the same fashion, `é` is left unchanged (it is represented internally in UTF8 as the two bytes `Ã©`).

Assume that Tralics has seen `@article`, then an opening brace or parenthesis, followed by `example`. All fields up to the closing brace (or parenthesis) are read, but, if the entry is useless, no error is signaled in case of undefined macros, or duplicate fields. If an entry is useful, all fields are remembered; if it has a crossref to an entry X, then X becomes useful. Remember: each entry has a Rtype, this is in general empty; it is added as a prefix to the cite key. For instance, 'thesefabien' gives 'footcite:thesefabien'. In the case of a crossreference from Y to X, we use as prefix for X the prefix of Y. An entry can be useful because the user has said `\nocite{*}`. There is a special hack for the Raweb: we have three types of entries, 'foot', 'year' and 'refer'. We already mentioned that the types 'year' and 'refer' could be the same as the empty type. The difference is that `\nocite` applies only to entries from the file 'year', never to 'foot' (there is an implicit `\nocite` for 'refer').

An entry is useful because it is cited (by `\cite` or `\nocite`). Since BibT<sub>E</sub>X is generally case insensitive, the entry shown above is useful if you say `\cite{Example}`. However, for L<sup>A</sup>T<sub>E</sub>X,

<sup>5</sup>This is not true in Tralics; you must escape this characters.

`\cite{Foo}` and `\cite{f00}` are two different items, as a consequence, two references are needed. Thus, an entry named ‘foo’ is ambiguous. For this reason, you should always capitalize entries in a consistent way (say, use always lowercase letters), and use the same method in the L<sup>A</sup>T<sub>E</sub>X document.

After some manipulations, the entry is printed on the bbl like this (BibT<sub>E</sub>X version)

```

20 \citation {GR99a}{example}{article}
21 \bateurs{\bpers\RAo J.\RAB \RAB Garrigue\RAB \RAf \bpers\RAo D.\RAB \RAB
22   R{\'e}my\RAB \RAf }
23 \cititem{btitle}{Extending {ML} with semi-explicit higher-order polymorphism}
24 \cititem{bjournal}{Journal of Functional Programming}
25 \cititem{bnumber}{1/2}
26 \cititem{bvolume}{155}
27 \cititem{byear}{1999}
28 \cititem{bpages}{134--169}
29 \url{ftp://ftp.inria.fr/INRIA/Projects/cristal/iandc.ps.gz}
30 \endcitation

```

or like that (Tralics version)

```

31 \citation{60}{footcite:thesefabien}{bid9}{foot}{phdthesis}[Sey98]
32 \bauthors{\bpers[Fabien]{F.}{Seyfert}{}}
33 \cititem{btitle}{Problèmes extrémaux dans les espaces de Hardy, Application
34 à l'identification de filtres hyperfréquences à cavités couplées}
35 \cititem{btype}{Ph. D. Thesis}
36 \cititem{bschool}{Ecole de Mines de Paris}
37 \cititem{byear}{1998}
38 \endcitation

```

There are some slight differences between these two entries. If you compare lines 20 and 31, you can see that the number of arguments of the `\citation` command has changed from three in the original version to six in the current version. The following were added: the type (here ‘foot’), the unique id (here ‘bid9’), the numerical print key (here ‘60’). The first entry was created by BibT<sub>E</sub>X, that cannot guess the Rtype of the reference nor the Tralics unique id. It could have computed the number 60, but we initially thought that only one of the two keys were useful (in the current version, the `\citation` command takes five arguments, plus an optional one after these.) If you compare lines 21 and 32, you can notice two differences. First, we have decided, in 2005, to add an optional argument to the `\bpers` command (it contains the full first name). This might be used for the Ra2005. The second difference is that it is impossible, in BibT<sub>E</sub>X, to print braces inside a name. Thus we used `\RAo` for an opening brace, `\RAf` for a closing brace and `\RAB` for a pair of closing and opening braces. Omitting the first line, the fields are printed in the following order:

1. Unless the type is proceedings, the author.
2. In the case of a book or inbook, the editor.
3. The title.
4. In the case of proceedings or incollection, the editor.
5. In the case of an article, the journal, number, and volume.
6. In the case of a book or inbook, the edition, series, number, volume, publisher, address.
7. In the case of a booklet, the howpublished and address.
8. In the case of incollection, the booktitle, series, number, volume, publisher, address.
9. In the case of inproceedings or conference, the booktitle, series, number, volume, organization, publisher, editor, pages, address.

10. In the case of a manual, the organization, edition, address.
11. In the case of masterthesis, coursereports, or phdthesis, the type, school, and address.
12. In the case of a techreport, the type, number, institution, address. (For the case of masterthesis, phdthesis and techreport, the type has a default value, that depends on the language, and is initialized together with the ‘jan’ macro).
13. In the case of misc, the howpublished, editor, booktitle, series, number, volume, publisher, address.
14. In the case of proceedings, the organization, series, number, volume, publisher, address.
15. In any case, the month, year.
16. In the case of inbook or incollection, the chapter.
17. In the case of inbook, incollection, article or proceedings, the pages.
18. In any case, the doi, url, additional fields, note.
19. In the case of an extension, all fields mentioned above are considered, in some order.

This may seem confusing (is there a standard way for formatting entries?). Note that missing fields are not printed. In some case, BibT<sub>E</sub>X prints a message like “there’s a number but no series” or “can’t use both volume and number”. No such message is printed by Tralics.

Two keys are computed, the ‘Sey98’ or ‘GR99a’ in the example, and the sort key, which is something longer. In fact, handling the author or editor field produces four characters strings  $L_1$ ,  $L_2$ ,  $L_3$  and  $L_4$ . The  $L_4$  string is the argument of the `\bauthors` or `\beditors` (see lines 21, 32). The  $L_1$  string is ‘Sey’ or ‘GR’, the  $L_2$  string contains the full name (it is like  $L_4$ , without the full first name, and braces) and  $L_3$  contains only the last name (not the first name).

We consider the author (the editor in the case of proceedings). This may give a triple  $L_1$ ,  $L_2$ ,  $L_3$ , unless the field is missing. If it is missing, we consider the ‘key’ field. If it is not empty, then  $L_1$  is formed of the first three characters of the field,  $L_2$  is empty,  $L_3$  is the field. If it is empty, we consider the editor (author, in case of proceedings). If this is empty, we consider the cite key, handle it like the ‘key’ above. Note: in the case of `Lo{\i}c` the first three characters are `Lo{\i}`, the last two characters are `{\i}c`. In the case of `Lo\ic`, asking for the first three or last two characters gives the full string. The last two characters of the year are added to  $L_1$ , so that we may obtain ‘GR99’. This gives the print label. The L<sup>A</sup>T<sub>E</sub>X companion says that you can use `year="{\SortNoop{86}}1991`". With the rules above, the last two characters of the year are ‘91’. However, Tralics uses the full year, not ‘861991’ when it computes the sort key. In the case when Tralics processes the Raweb for, say year 2003, if a reference has type ‘year’, then its year field should not be missing, and should be ‘2003’. Otherwise an error is signaled<sup>6</sup>. The sort key is computed as follows: first a prefix, then the cite label, then  $L_2$ , then the year, then the title. All characters are converted to lower case. Note: when BibT<sub>E</sub>X converts `{\E}` to lower case, the result is ‘e’. Converting ‘É’ can produce strange results. Such subtleties do not exist in Tralics (the style sheet that converts the XML to HTML sorts all entries; how can we tell it that the author used a `\SortNoop` command?).

Note: Tralics defines `\sortnoop` to gobble its argument. On the other hand, the BibT<sub>E</sub>X interpreter, when computing the title part of the sort key, in the case of `{\noopsort foo}` removes the command and the braces; the same is done for `\SortNoop` and `\noopsort`. In a case like `title="study of {\H^p}`, `part {I}`" it removes the braces (character after opening brace must be dollar or upper case letter). The reason for this is that otherwise ‘part II’ comes before ‘part I’, and this looks silly.

Because of this sort-again, we try to be clever. Said otherwise, for the Raweb, and only the Raweb, we use a prefix, formed of a letter and  $L_3$ . The prefix 0 for an entry of Rtype ‘refer’, 1

<sup>6</sup>No error is signaled if you invoke Tralics with the switch ‘nobibyearerror’

for an entry of Rtype ‘foot’, and for entries of Rtype ‘year’, it is: 2 for book, booklet, proceedings, 3 for phdthesis, 4 for article, inbook, incollection, 5 for conference, inproceedings, 6 for manual, techreport, coursenotes, 7 for masterthesis, misc, unpublished. These numbers are indices into a table. Currently the order is 02345671. In a future version, this might be changed (however, the result should be compatible with the style sheets described in the second part of this report).

Let’s repeat: for the Raweb case, we have in the sort key a prefix that depends on the type and Rtype, followed by the author names, the print key, the full author names, the year, the title. In this case, the content of the `bbl` will be as on line 31: the first argument of `\citation` is not the print key, but the index of the reference in the table after sorting. On the other hand, for the non-Raweb case, the sort key starts with the print key, the `bbl` looks like line 20. The important point is: assume that we have two entries with the same print key, say ‘GR99’; we must change them to ‘GR99a’ and ‘GR99b’, this is easy to do when they are consecutive. The following piece of code comes from a standard `bst` file. Parsing a `bst` file is rather easy (maybe one day, Tralics will do it). The important point is that a postfix language is used: instead of: if a then b else c, you say: a b c if. This piece of code computes a suffix for every entry that has the same key as the previous one.

```

39 FUNCTION {forward.pass}
40 { last.sort.label sort.label =
41   { last.extra.num #1 + 'last.extra.num :=
42     last.extra.num int.to.chr$ 'extra.label :=
43   }
44   { "a" chr.to.int$ 'last.extra.num :=
45     "" 'extra.label :=
46     sort.label 'last.sort.label :=
47   }
48   if$
49 }

```

Here is the companion routine, executed in reverse order. Its purpose is to add the ‘a’ suffix when the next entry has a ‘b’ suffix. There is a piece of code, not shown here, that computes the longest label. This is sometimes nonsense (consider the ‘De La Cruz’ case below).

```

50 FUNCTION {reverse.pass}
51 { next.extra "b" =
52   { "a" 'extra.label := }
53   'skip$
54   if$
55   label extra.label * 'label :=
56   extra.label 'next.extra :=
57 }

```

In summary, when Tralics is used instead of BibTeX, the following happens. We have a big entry list, and a list of typed databases. From the entry list, we consider only unsolved ones. For each entry, a prefix is computed, for instance, ‘footcite:fabien’, by considering the Rtype, the word ‘cite:’ and the cite key. If the Rtype is anything else than ‘foot’, an empty value will be used.<sup>7</sup> When an entry with cite key ‘foo’ is read from a database of type ‘bar’, the same mechanism is applied. The type of a database is currently one of ‘year’, ‘refer’ or ‘foot’ (the default being ‘year’). We plan to extend this mechanism: more than these three types can be used; ‘year’ and ‘refer’ are sometimes the same as empty, but ‘refer’ has an implicit `\nocite`.

All entries from the database files are read, and stored if useful. For each entry X that has a crossreference to Y, missing fields in X are copied from Y. After that Y is discarded (unless cited

<sup>7</sup>But the prefix ‘refer’ is kept if the current year is at least 2006, and the magic switch `distinguish-refer-in-rabib` has been given



via `\cite` or `\nocite`). An error is signaled in case some references are undefined. After that, the sort label is computed, entries are sorted, the print label is computed, and everything is printed on the `bbl` file. This is `apics_bbl` if the jobname is `'apics'`. Note the underscore in the name.

This is the XML version of the reference above, as used in the `Raweb2004`.

```

58 <citation from='foot' key='60' id='bid9' userid='footcite:thesefabien'
59   type='phdthesis'>
60 <bauteurs><bpers prenom='F.' part='' nom='Seyfert' junior=''></bauteurs>
61 <bttitle>Problèmes extrémaux dans les espaces de Hardy,
62   Application à l'identification de filtres hyperfréquences à cavités
63   couplées</bttitle>
64 <btype>Ph. D. Thesis</btype>
65 <bschool>Ecole de Mines de Paris</bschool>
66 <byear>1998</byear>
67 </citation>

```

## 4.4 The format of a name

We shall discuss in this section how names can be used in a Bib<sub>T</sub><sub>E</sub><sub>X</sub> file, and how Tralics constructs keys. We have already mentioned a procedure that gives ‘Sey’ from ‘Seyfert’. It is not satisfactory, but is used only in rare cases (when the year is strange, or a strange key has been used). The important point that, when we fetch the first three letters of `Lo\ic`, we do not obtain neither ‘`Lo\`’ nor ‘`Lo\`’. The mechanism explained here is more subtle. The L<sup>A</sup>T<sub>E</sub>X companion explains that, in order to get ‘Göd’ for the key, you should use one of the first names shown here, not the others.

```

author = {A. G{"o}del and B. G{"{o}}del},
editor  = {C. {G{"{o}}del} and D. {G{"{o}del}}

```

The rule is that special Bib<sub>T</sub><sub>E</sub><sub>X</sub> characters are formed by a left brace followed by a backslash. In the case C, the brace is inside another brace. In fact, if the bibliography contains the following

```

68 @Article{GoA,
69   author = {A. G{"o}del      }, title="X" }
70 @Article{GoB,
71   author = {B. G{"{o}}del   }, title="X" }
72 @Article{GoC,
73   author = {C. {G{"{o}}del} }, title="X" }
74 @Article{GoD,
75   author = {D. {G{"{o}del}  }, title="X" }

```

then the translation by Tralics2.9 looks like this. If you compare with lines 60 and 61 above, you can see that the full first name appears, empty attribute pairs `part` and `junior` are not shown.

```

76 <biblio>
77 <citation from='year' key='Ga' id='bid2' userid='cite:GoC' type='article'>
78 <bauteurs><bpers prenom='C.' nom='Gödel' prenomcomplet='C.'/></bauteurs>
79 <bttitle>X</bttitle>
80 </citation>
81 <citation from='year' key='Gb' id='bid3' userid='cite:GoD' type='article'>
82 <bauteurs><bpers prenom='D.' nom='Gödel' prenomcomplet='D.'/></bauteurs>
83 <bttitle>X</bttitle>
84 </citation>
85 <citation from='year' key='Göd' id='bid0' userid='cite:Goa' type='article'>
86 <bauteurs><bpers prenom='A.' nom='Gödel' prenomcomplet='A.'/></bauteurs>

```

```

87 <bttitle>X</bttitle>
88 </citation>
89 <citation from='year' key='Göd' id='bid1' userid='cite:GoB' type='article'>
90 <bauteurs><bpers prenom='B.' nom='Gödel' prenomcomplet='B.'/></bauteurs>
91 <bttitle>X</bttitle>
92 </citation></biblio>

```

The same file processed by Bib<sub>T</sub>E<sub>X</sub> gives the following keys:  $\{G\{\}\}a$ ,  $\{G\{\}\}b$ ,  $G\{\}\{o\}da$  and  $G\{\}\{o\}db$ . The first two keys are invalid. The reason why suffixes a and b are added is that a special Bib<sub>T</sub>E<sub>X</sub> function removes braces and funny characters when comparing keys. Such a function is not implemented in Tralics, thus labels  $G\{\}\{o\}$  and  $G\{\}\{o\}d$  are considered different, although their translation is the same. In Tralics, the best thing to do is use ‘Gödel’ as name.

Since lots of errors may be found in bibliography files, Tralics tries to be clever. First, it replaces ‘ $\{c\}$ ’ by ‘ç’ and ‘ $\{C\}$ ’ by ‘Ç’. It also replaces ‘ $\{v\}$ ’ by ‘{v c}’. Expressions of the form  $\{a\}e$  are replaced by  $\{e\}$ . We also replace backslash-space by a single space. Maybe other replacements of this kind will be made in a future version. For instance, we could expand all accent characters, and interpret double-hat construct, so that ‘é’, ‘ $\{e\}$ ’, and ‘ $\{e\}$ ’ are interpreted in the same way (the translation is the same).

After that, characters or group of characters are classified, this will make parsing easier. A sequence like ‘ $\{foo\}$ ’ will be considered as a single random character; something like  $\{e\}$  as a single lower case letter,  $\{E\}$  as a single uppercase letter. The expression  $\{e\}$  will be replaced by  $\{e\}$  with a warning,  $\{i\}$  will be rejected (unless inside braces) because a single character is needed after backslash-accent. Commands like  $\{foo\}$  are also rejected. Note that an ampersand & is an error (some people try to use this instead of ‘and’). Character categories are: space, comma, dash, and tie (this is a ~). In a case like this,

```

93 @Article{cruz,
94   author = {Maria {\MakeUppercase{d}e La} Cruz},
95   title="X" }

```

the print key computed by Bib<sub>T</sub>E<sub>X</sub> is  $\{MakeUppercase{d}e La\}C$ , this typesets as ‘De LaC’. Such a construct is not understood by Tralics, that thinks that the last name is ‘Cruz’.

If more than one author is given, in the author or editor list, you should use ‘and’ as separator. Case is irrelevant, a space is required. For instance, the following citation contains 3 authors and others. The print key is ‘AAJA+’, because the last author has a double last name.

```

96 @Article{many,
97   author = {Joe~And and And,Joe and Joe-And And others}
98   title="X" }

```

The Bib<sub>T</sub>E<sub>X</sub> transformation of this is

```

99 \bibitem[AAJA{\etalchar{+}}]{many}
100 Joe And, Joe And, Joe-And, et~al.

```

If the list is too long, you can use ‘others’ as the last name (case is important). A name has four components: von, First, Last and Junior. On line 32, you can see the value of the full first name, then the abbreviated first name, then the von part (empty) then the last name, then the junior part (empty). In Tralics, the von part is always merged with the last name. Consider somebody named Jean de la Fontaine. French rules say that the particle ‘de’ should be omitted, unless preceded by the first name or a word like ‘Monsieur’. In particular, in the dictionary, you will find him between La Follette (an American politician) and Lafontaine (a Canadian politician), not between Delacroix and Delage. More interesting is the case of Marie Joseph Gilbert Motier, marquis de La Fayette. The name of this guy is ‘Motier’, but he is known as ‘La Fayette’. Another example is William Thomson (For his work on the transatlantic cable Thomson was created Baron Kelvin of Largs in 1866. The Kelvin is the river which runs through the grounds of Glasgow

University and Largs is the town on the Scottish coast where Thomson built his house.) How this guy should be cited is unclear: William Thomson or Lord Kelvin?

The simple case is when two fields are given, with a comma between. The first field is the last name, the other field is the first name. Then comes the case of three fields: last name, junior, and first name. You cannot use more than three fields, that is, you cannot give more than two commas. In the case no comma is given, we look at a ‘von’ part. This is something that starts at a lower case letter. For instance,

```
101 @Article{poussin,
102   author = {Charles Louis Xavier Joseph de la Vall{\a'e}e Poussin   },
103   title="X" }
```

This is what BibT<sub>E</sub>X puts in the bbl file:

```
104 \bibitem[dLVP]{poussin}
105 Charles Louis Xavier~Joseph de~la Vall{\a'e}e~Poussin.
106 \newblock X.
```

The translation by Tralics is the same, but no ties are inserted (BibT<sub>E</sub>X inserts one for the first name, the von part, the last name, see T<sub>E</sub>Xbook, page 92); in my opinion, it is better to split a line between two names, rather than split a name (what hyphenation patterns should be used in a case like ‘Michel Goosens’, the current patterns, here english, or those found in the bibliography, thus french if we cite the French version of the L<sup>A</sup>T<sub>E</sub>X companion?). The ‘De La Cruz’ example shows how you can fool BibT<sub>E</sub>X. Tokens between names are recognized. For instance, consider:

```
107 @Article{strange,
108   author = {A-b-C and A.b.C and A~b~C and A.Bb.Cc},
109   title="X" }
```

This is how BibT<sub>E</sub>X interprets the names. Authors number two and four have only a last name, no von part, no first name.

```
110 \bibitem[bCAbCA]{strange}
111 A~b~C, A.b.C, A~b~C, and A.Bb.Cc.
112 \newblock X.
```

This is the translation by Tralics. You can see that, for the last author, one dot has been replaced by a space: this is done in case no other way is found to split the name, but there is an upper case letter on each side of the dot. You can also see that BibT<sub>E</sub>X inserts some characters (here ties) instead of dashes. Tralics keeps the dashes, whenever possible.

```
113 \citation{bCABCB}{cite:strange}{bid3}{year}{article}
114 \bauthors{\bpers[A]{A.}{b-C}{ }
115           \bpers[]{}{A.b.C}{ }
116           \bpers[A]{A.}{b~C}{ }
117           \bpers[A]{A.}{Bb.Cc}{ }}
118 \cititem{btitle}{X}
119 \endcitation
```

Here is another example.

```
120 @Article{strange2,
121   author = {Jean-Claude XX and J.-Ch. YY and J.-{Ch.} ZZ},
122   title="X" }
```

This is the translation by BibT<sub>E</sub>X, in ‘abbrv’ mode. The format used in plain mode is `{ff }{vv }{ll}{, jj}`, and in abbrv mode, it is `{f. }{vv }{ll}{, jj}`. This is explained in any good reference about BibT<sub>E</sub>X<sup>8</sup>.

<sup>8</sup>For instance *Tame the BeaST*, by Nicolas Markey, available on CTAN

```

123 \bibitem{strange2}
124 J.-C. XX, J.-C. YY, and J.-C. ZZ.
125 \newblock X.
126 \bibitem{poussin}
127 C.~L. X.~J. de~la Vall{\a'e}e~Poussin.
128 \newblock X.

```

This is the translation by Tralics. The quantity ‘{Ch.}’ is considered as a single character. No dot is added after it, since it is terminated by a dot.<sup>9</sup>

```

129 \citation{XYZ}{cite:strange}{bid3}{year}{article}
130 \bauthors{\bpers[Jean-Claude]{J.-C.}{XX}{
131           \bpers[J.-Ch.]{J.-C.}{YY}{
132           \bpers[J.-{Ch.}]{J.-{Ch.}}{ZZ}{}}
133 \cititem{btitle}{X}
134 \endcitation

```

The print key is computed as follows: Each author gives an initial (if the name is complicated, more than one will be used, for instance Poussin gives four letters ‘dlVP’). If a single author is cited, and if it gives less than three letters, then the first three letters of its name are used (for instance, Seyfert gives ‘Sey’). If more than four authors are given, only the first three ones give an initial, there is a ‘+’ sign at the end. If ‘and others’ is given, there is also a ‘+’ sign.

We show here the sort key, as computed by Tralics, for some the entries shown above. Remember that these entries have no year field and that the title is X.

```

135 cru m. {\makeuppercase{d}e la}. cruz          x
136 g c. {g{"o}del}          x
137 g d. {g{"o}del}          x
138 g{"o}d a. g{"o}del          x
139 g{"o}d b. g{"o}del          x
140 aa+a j. and j. and joe-and etal          x
141 dlvp c. l. x. j. de la vall{\a'e}e poussin          x

```

These are the keys, for the same entries, computed by Bib<sub>TEX</sub>, using the alpha style. You can see that Bib<sub>TEX</sub> uses last name and first name, whereas Tralics uses abbreviated first name then last name. The format is: {vv{ }}{ll{ }}{ ff{ }}{ jj{ }}.

```

142 delac   dela cruz maria          x
143 god     godel a          x
144 god     godel b          x
145 g       godel c          x
146 g       godel d          x
147 aaaja   and joe and joe joe and et al          x
148 dlvp    de la vallee poussin charles louis xavier joseph          x

```

## 4.5 Commands for the bbl

The Raweb DTD explains that the following items can appear inside a bibliography entry.

- <baddress>, the address of the publisher.
- <bateurs>, the names of the authors of the document.
- <bbooktitle>, the title of the document.
- <bchapter>, the chapter number, in case it is relevant.

<sup>9</sup>According to N. Markey, you should say {\relax Ch}ristopher in Bib<sub>TEX</sub>

- `<bedition>`, the edition number.
- `<bdoi>`, the DOI.
- `<bediteur>`, the names of the editors.
- `<bhowpublished>`, how a document is published, in unusual cases.
- `<binstitution>`, the institution, for the case of a report.
- `<bjournal>`, the journal where an article can be found.
- `<bmonth>`, month of publication.
- `<bnote>`, a note.
- `<bnumber>`, the number of a report, etc.
- `<borganization>`, organizer (of a conference ...)
- `<bpages>`, page number (or sequence of pages).
- `<bpublisher>`, the publisher of a book.
- `<bschool>`, the school for a thesis.
- `<bseries>`, the name of a series.
- `<bttitle>`, the title of the reference.
- `<btype>`, the type (of a thesis, technical report).
- `<bvolume>`, the volume.
- `<xref>`, an external link.
- `<byear>`, the year of publication.

In almost every case, if the database file contains a field ‘foo’ with value ‘bar’, the `tbl` file will contain `\cititem{bfoo}{bar}`, and this is translated into `<bfoo>bar</bfoo>`. The `\cititem` command takes two arguments. The second argument is translated as usual. The first argument is the name of the resulting element. There is a hook: in the case where `\cititem-foo` is defined (this is `\cititem` followed by a dash followed by the name of the field), this macro is used instead of the default procedure. If the database contains a ‘url’ field, the result is a call to the `\url` command, that will produce a `<xref>` element. The `\cititem` command should be used only in a bibliography.

If the entry in the database contains a ‘author’ or ‘editor’, the `\bauthors` or `\beditors` commands will be called. These two commands must be used inside a bibliography. They take a single argument, translate it, and put the result in a `<bauteurs>` or `<beditor>` element. Note: the bibliography part of the Raweb DTD was meant to be temporary. For this reason, the names were chosen so as to replace them easily with new names (hence the prefix ‘b’); For some reason, ‘auteurs’, ‘editeur’ and attributes of ‘bpers’ have French names. Later on, we decided to modify the Tralics names, hence the ‘bauthors’ and ‘beditors’. Because ‘bauteurs’ had a final s, we added an s to both command names; not the best choice.

The `\bpers` command takes one optional argument, and 4 required arguments. The translation is an empty `<bpers>` element with following attributes: `prenomcomplet` for the optional argument, and `prenom`, `part`, `nom`, `junior` for the required arguments.

The `\citation` command constructs a `<citation>` element. It takes 5 required arguments, and an optional argument. The optional argument is ignored. Other arguments are converted to attributes. The whole text, up to `\endcitation` is translated in bibliography mode, and added to the `<citation>` element. Example:

```

149 \citation{a}{b}{c}{d}{e}
150 \cititem{foo}{bar}
151 \beditors{\bpers[a]{b}{c}{d}{e} \bpers[]{}{}{} \cititem{etal}{}}
152 \endcitation
    The translation is
153 <citation from='d' key='a' id='c' userid='b' type='e'>
154   <foo>bar</foo>
155   <bediteur>
156     <bpers prenom='b' part='c' nom='d' junior='e' prenomcomplet='a' />
157     <bpers prenom='B' nom='C' />
158     <etal />
159   </bediteur>
160 </citation>

```

## 4.6 Other commands

The `\bibliography` command takes one argument, this is a comma separated list of database files. Spaces are ignored. The command can be given more than once. This command (the last occurrence) defines the position where the bibliography should be inserted.

The command `\insertbibliothere` can be used to force the position of the bibliography. It overwrites the location specified by the previous command.

The environment ‘`thebibliography`’ can be used for typesetting the bibliography. There is an optional argument (ignored), a required argument (ignored), an optional argument (ignored). The result is an XML element whose name is defined by `\refname`, by default ‘`Bibliography`’, and whose content is formed of the translation of the environment. You can redefine this `\refname` command. An error is signaled if strange commands appear in the argument, but not for invalid characters (in particular, space cannot appear in an element name). The command can be empty. In this case, the name will not appear in the XML result.

The command `\bibliographystyle` takes one argument. Its translation is empty. The argument is remembered. This is the style to use. If the argument is ‘`bibtex:`’, this is an indication that BibTeX should be used instead of Tralics for the production of the `bbl`. The style can be given after the colon, or with the invocation of another command. If the argument is ‘`program:foo`’, this means to use `foo` as program. For instance `\bibliographystyle{program:cat -v}`. In this example, this will print the auxiliary file; this is not good, because the command should create the `bbl` file (its argument is `jobname.aux`, data must be written on `jobname.bbl`). A second `\bibliographystyle` command can be used for specifying the style (the default is ‘`plain`’). Example. Consider a file that contains these lines

```

161 \documentclass{article}
162 \begin{document}
163 \AtEndDocument{\bibitem{unused}Hey}
164 \bibliography{torture}
165 \bibliographystyle{bibtex:}
166 \cite{poussin,cruz,many,strange,unused}
167 \end{document}

```

When Tralics sees the `\end{document}` command, it evaluates it (with the hooks, etc.) After that, a `bbl` is created and translated. If there is no unsolved entry, nothing happens. If no style command indicates that BibTeX or an external program should compute the `bbl`, then Tralics does it, as explained above. In the case of the Raweb, three database files are used: `apicsfoot_2004`, `apicsrefer_2004`, and `apics2004`. These files are typed ‘`foot`’, ‘`refer`’ and ‘`year`’. In the non-Raweb

case, files in the list indicated by `\bibliography` are used. If a file is named ‘miaou+refer’ or ‘miaou+foot’ and does not exist, then miaou is tried instead; in this case the type will be ‘refer’ and ‘foot’ (otherwise, it is ‘year’). In the case an external program is used, a minimal auxiliary file is created. In the case of the example, it will contain

```

168 \citation{poussin}
169 \citation{cruz}
170 \citation{many}
171 \citation{strange}
172 \bibstyle{plain}
173 \bibdata{torture}

```

The database torture.bib contains a sequence of entries, plus the following lines. In order to understand the last line, you have to remember that character strings are always balanced against braces. Hence it is not: open brace concatenated with 1 concatenated with 1 and close brace. It is: open brace, double quote, space, sharp, etc, up to double quote, close brace.

```

174 @String{ stra= {\def}}
175 @String{ strb= "#1" }
176 @String( strc= "\mycmd " )
177 @Preamble (stra # strc # strb )
178 @Preamble( "{" #1 #1 "}")

```

After that, the external program is called, and the bbl file is read. In the example this gives the following. The first line is the preamble.

```

179 \def\mycmd #1{" #1 #1 "}
180 \begin{thebibliography}{1}
181
182 \bibitem{many}
183 Joe And, Joe And, Joe-And, et~al.
184 \newblock X.
185
186 \bibitem{strange}
187 A~b~Cde.
188 \newblock X.
189
190 \bibitem{poussin}
191 Charles Louis Xavier~Joseph de~la Vall{\a'e}e~Poussin.
192 \newblock X.
193
194 \bibitem{cruz}
195 Maria {\MakeUppercase{d}e La}~Cruz.
196 \newblock X.
197
198 \end{thebibliography}

```

After that, the bibliography is translated and inserted. The resulting XML file is shown here.

```

1 <?xml version='1.0' encoding='iso-8859-1'?>
2 <!DOCTYPE std SYSTEM 'classes.dtd'>
3 <!-- Translated from latex by tralics 2.9.1, date: 2006/11/02-->
4 <std>
5 <biblio>
6 <Bibliography><p id='bid2'>
7 Joe And, Joe And, Joe-And, et al.
8 X.</p>

```

```

9 <p id='bid3'>
10 A b Cde.
11 X.</p>
12 <p id='bid0'>
13 Charles Louis Xavier Joseph de la Vallée Poussin.
14 X.</p>
15 <p id='bid1'>
16 Maria De La Cruz.
17 X.</p>
18 </Bibliography>
19 </biblio><p><cit><ref target='bid0'/></cit>, <cit><ref target='bid1'/></cit>,
20 <cit><ref target='bid2'/></cit>, <cit><ref target='bid3'/></cit>,
21 <cit><ref target='bid4'/></cit></p>
22 <p id='bid4'>Hey</p>
23 </std>
24

```

Finally, we show here everything printed on the screen, including all warnings by BibTeX.

```

1 This is tralics 2.9.1, a LaTeX to XML translator
2 Copyright INRIA/MIAOU/APICS 2002-2006, Jos'e Grimm
3 Licensed under the CeCILL Free Software Licensing Agreement
4 Starting translation of file testb.tex.
5 Configuration file identification: standard $ Revision: 2.24 $
6 Read configuration file /Users/grimm/work/cvs/tralics/confdir/.tralics_rc.
7 Document class: article 2006/08/19 v1.0 article document class for Tralics
8 Bib stats: seen 5(1) entries
9 This is BibTeX, Version 0.99c (Web2C 7.5.4)
10 The top-level auxiliary file: testb.aux
11 The style file: plain.bst
12 Database file #1: torture.bib
13 Warning--empty journal in many
14 Warning--empty year in many
15 Warning--empty journal in strange
16 Warning--empty year in strange
17 Warning--empty journal in poussin
18 Warning--empty year in poussin
19 Warning--empty journal in cruz
20 Warning--empty year in cruz
21 (There were 8 warnings)
22 Math stats: formulas 0, kernels 0, trivial 0, \mbox 0, large 0, small 0.
23 Buffer realloc 0, string 1240, size 12510, merge 4
24 Macros created 97, deleted 0; hash size 1565; foonotes 0.
25 Save stack +20 -20.
26 Attribute list search 1476(1402) found 906 in 1097 elements (1076 at boot).
27 Number of ref 0, of used labels 0, of defined labels 0, of ext. ref. 0.
28 Modules with 0, without 0, sections with 0, without 0
29 Output written on testb.xml (593 bytes).
30 No error found.
31 (For more information, see transcript file testb.log)

```

Some comments. Line 6 shows the name of the configuration file. If this file contains a line that starts with ‘## tralics ident rc=’ then all characters after the equals sign are printed (see line 5). Since version 2.5 (pl4), in the case where character number 30 is a dollar sign, a space will be



added after it.<sup>10</sup> The reason for this is that the RCS software interprets a string like ‘Revision’ in dollar signs; we do not want it to replace the 2.11 by the revision number of the  $\LaTeX$  document. We shall explain elsewhere how to read the statistics.

Line 8 shows the number of entries in the biblist. If some entries are solved, they are shown in parentheses. Here, we have  $5-1=4$  unsolved entries. If line 5 of the source file is commented out, then Bib $\TeX$  is not used, and lines 9 to 21 will be replaced by the single line ‘Seen 4 bibliographic entries’.

The standard configuration file contains a line that says that ‘article’ is an alias for ‘std’. The ‘std’ configuration defines two quantities: the name of the DTD, hence the root element, it is `<std>`, see line 4 of the XML result. It defines `xml_biblio` to be ‘bibliography’. This is the name of the element that will hold the bibliography. The default value is ‘biblio’, but it can be redefined (see line 5). Do not confuse this with the name of the element produced by the environment ‘thebibliography’, that appears line 6 in the XML result.

---

<sup>10</sup>In fact, in subsequent version, a space is added after the first dollar sign, independently of the position on the line.



## Chapter 5

# Other commands

### 5.1 Character encoding

We have to distinguish between input encoding, internal encoding and output encoding. The internal encoding of T<sub>E</sub>X is ASCII (i.e. 65 is the internal code of the upper case letter A), at least for all characters with code between 32 and 126. The input encoding is the mechanism that converts the code of the letter A supplied by computer into the code 65. Almost all input encodings are nowadays ASCII-based, they produce the same value for the letter A; the results may be different for a character like é. The output encoding indicates for a letter, say A, which position in the font to use. We shall not discuss the output encoding here. Let's just notice that the character ‘{’ exists in the font cmtt10, but not in other text fonts of the computer modern family. If you read a version of this document that uses the original encoding (OT1), braces shown in error messages are taken from a math font, hence are upright. Some years ago, a 8bit encoding (called T1) was designed, which contains braces. You can compare Figure 1 in appendix F of the [4] (describing the font cmr10) with Table 7.32 of [6], describing ecrm1000.

The first version of T<sub>E</sub>X was using 7bit input and output characters (but fonts and dvi files were coded on 8bits). There is an extension  $\Omega$  to T<sub>E</sub>X that accepts 16bit characters as input, using different encoding schemes. Characters that are not part of the ASCII specifications (less than 32 or greater than 126) are not guaranteed to be treated the same in all implementations. For this reason, it is wise to load the `inputenc` package, with the current encoding as argument. The effect will be that some characters, like é will become active, and expand to `\'e`. As a result: only ASCII letters are allowed in control sequence names. On the other hand, if you say `\begin{motclés}`, then L<sup>A</sup>T<sub>E</sub>X complains with *LaTeX Error: Environment motcl\es undefined*. Don't try to define the `motcl\es` environment: the expansion of the accent depends on the context: it is é for `\begin` and `\'e` for the macro that prints the error message. Non-ASCII characters may be printed by T<sub>E</sub>X as `^^ab` (in some older version of T<sub>E</sub>X, I had to pretend, via locale settings, that my computer did not understand English in order for it to output the guillemet as «).

A silly question concerns end-of-line markers. Some systems like Unix use LF (line feed) as line separators, some others like Macintosh use CR (carriage return) and Windows uses CR-LF. This is replaced by T<sub>E</sub>X by a single character: the carriage return with ASCII code 13. Tralics interprets CR-LF, CR and LF alike: as an end-of-line marker. This marker will be replaced by the character whose code is in `\endlinechar`, provided that this value is in the range 0–255<sup>1</sup>. The default value is 13, a character of category 5. The tokeniser converts this into a `\par` token, a space token or ignores it depending on the state. This space token has value 32 (but Tralics uses 10, so as to keep

<sup>1</sup>In Tralics, the range is 0–65535; note that the null character is discarded, and characters U+FFFE, U+FFFF are illegal Unicode characters

the same line breaks in the XML result as in the  $\text{\TeX}$  source). Note that, whenever a line is read, spaces at the end of the line are removed. If you want a space after a control sequence, you say something like  $\text{\TeX}\backslash\_\text{ }^J$ , and if this construct appears at the end of a line, the space is ignored; if the endline character has category code 5, it will be converted to a space, and everything works fine; if this character is for instance 65, you may get a strange error, like this

```
! Undefined control sequence.^J
1.170 ...reaks in the \XML\ result as in the \TeX\^^J
                                     ^^J
? ^^J
```

We have shown here the end of line as  $\text{\TeX}\backslash\_\text{ }^J$ . There are four lines: the error messages, two context lines, and the line with the prompt. The two context lines show that the space at the end of the line is removed.  $\text{\TeX}$  does not print the undefined control sequence: it assumes that it is either the last token on the first context line, or a token marked as  $\langle$ recently read $\rangle$  or something like that; in our case, the undefined control sequence is the one obtained by replacing  $\text{\TeX}\backslash\_\text{ }^J$  by the value of the endline character.

There is a way to enter special characters in  $\text{\TeX}$ , for instance  $\text{\TeX}\backslash\_\text{ }^J$  is a line feed. The algorithm is the following: whenever  $\text{\TeX}$  sees two consecutive identical characters of category code 7, followed by a character whose number is  $x$ , it replaces these three characters by the character whose code is  $y$ , where  $y = x - 64$  if  $x \geq 64$ , and  $y = x + 64$  if  $x < 64$ . Hence  $\text{\TeX}\backslash\_\text{ }^?$  yields  $y = 127$  (this is the delete character). All characters with codes between 1 and 26 can be obtained using the form  $\text{\TeX}\backslash\_\text{ }^A$ ,  $\text{\TeX}\backslash\_\text{ }^B$ , etc. The null character is  $\text{\TeX}\backslash\_\text{ }^@$ , characters with code between 27 and 31 are  $\text{\TeX}\backslash\_\text{ }^$ ,  $\text{\TeX}\backslash\_\text{ }^$ ,  $\text{\TeX}\backslash\_\text{ }^$ ,  $\text{\TeX}\backslash\_\text{ }^$  and  $\text{\TeX}\backslash\_\text{ }^$ . Character 32 can be represented as  $\text{\TeX}\backslash\_\text{ }^$ . All other characters are ASCII characters. This is an example of use:

```
27=\char'\^^[, 28=\char'\^^\, 29=\char'\^^], 30=\char'\^^^, 31= \char'\^^_
```

Because some characters in the list are of category code 15 (invalid), we have used the construction  $\text{\TeX}\backslash\_\text{ }^A$  (with A replaced by some other character). There is no difference between  $\text{\TeX}\backslash\_\text{ }^A$  and  $\text{\TeX}\backslash\_\text{ }^A$ , unless the category code of the character is one of 0, 5, 9, 14, or 15. The result is the character at position 65 or whatever in the current font; the example above selects positions 27 to 31. The translation is

```
27=&#x1B;, 28=&#x1C;, 29=&#x1D;, 30=&#x1E;, 31= &#x1F;
```

Note that these characters are invalid in XML1.0, so that this example is not good; if you compile this document with  $\text{\LaTeX}$ , you will see  $\text{\TeX}\backslash\_\text{ }^=ff$ ,  $\text{\TeX}\backslash\_\text{ }^=fi$ ,  $\text{\TeX}\backslash\_\text{ }^=fl$ ,  $\text{\TeX}\backslash\_\text{ }^=ffi$ ,  $\text{\TeX}\backslash\_\text{ }^=ffl$ . In general you will see a ff ligature or a oe one; this depends on the output encoding.

When  $\text{\TeX}$  switched to 8 bits, the rule changed a little bit: the previous rule applies only if  $0 \leq x \leq 127$ , it gives  $0 \leq y \leq 127$ . Another test was added: if you say  $\text{\TeX}\backslash\_\text{ }^ab$ , these four characters are replaced by the single character whose code is  $ab$  (in base 16, i.e. 171 in base ten in this case). In such a case two characters are needed: a letter or a digit; only lower case letters between a and f are allowed. Thus every character in the range 0-255 has such a representation. Note that, by default, the character  $\text{\TeX}\backslash\_\text{ }^ab$  has category code 12, hence is valid. What appears in the dvi file depends on the output encoding, in the case of a 7bit encoding, the character is unknown, a warning is printed in the transcript file, that's all, otherwise, it should be an opening guillemet, but it could as well be  $\acute{n}$ . The purpose of a package like `inputenc` is to change the category code of all special characters, so that it behaves like a command and produces, in the dvi, something that is, as much as possible, independent of the output encoding.

According to this rule, the character 32 has can be entered as  $\text{\TeX}\backslash\_\text{ }^20$ . There is one situation where the space character can be used in this way: at the end of the line, when  $\text{\TeX}\backslash\_\text{ }^endlinechar$  is non trivial. Note that, in the case where the resulting character has category 7, it can participate in a hat-hat construct. Here is an example.

```

{1^^{^^ab2^^5e^ab3^^5e^5e^ab4\def\Abc{ok}\def\Ac{OK}\^^41bc\b^^41c}
{\catcode '\é=7 ééab $xé2$ %next line should produce M
éé
%$1^è=^^^AééT$ %% hat hat control-A
$1^è=^^^A$ %% hat hat control-A
}\def\msg{a message.^^J}

```

Some explanations are needed. ^^{ is a semi colon, ^^ab is an opening French guillemet, ^^5e is a hat (recursion...), ^^41 is the uppercase letter A. The first line of the example explains that such funny characters can appear in a control sequence name. The second line shows that the hat-hat mechanism can be used with other characters than a hat. It also shows that, if the mechanism cannot be applied, a character with category 7 behaves like a superscript character, whatever its numeric value. The line that follows shows that the end-of-line character is ASCII 13, aka control-M (usually written as ^M). After that, there are two lines containing a control-A character, shown here as ^A. It is preceded by hat-hat, so that the effect should be a single A. The line that is commented out contains a control-T written as ééT (for some strange reasons, this character is invalid in XML1.0, but valid as an entity in XML1.1, [8, 9]). The last line is just a real example of ^^J. This character is printed by Tralics as LF, or CR-LF on Windows. This is the translation of Tralics:

```

<p>1;&#xAB;2&#xAB;3&#xAB;4okOK
&#xAB; <formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'
><msup><mi>x</mi> <mn>2</mn> </msup></math></formula
> M<formula type='inline'><math xmlns='http://www.w3.org/1998/Math/MathML'
><mrow><msup><mn>1</mn> <mi>&#xE8;</mi> </msup><mo>=</mo><mo></mo>
><mi>A</mi></mrow></math></formula>
</p>

```

We inserted some newline characters at unusual places (just before greater than signs), other spaces were produced by Tralics; in order to make sure that 8bit characters are printed correctly, we asked Tralics for a seven bit output.

As said above, Ω accepts 16bit characters, using the notation ^^ ^^abcd. This syntax was implemented in Tralics2.7, via the \char command (remember that in Tralics, the \char and \chardef commands accept 27bit integers); as a consequence, these characters could not be used in a command name; this restriction does not apply anymore (the default category code of characters with code greater than 127 is other, namely 12). Example

```

\def\foo#1#2#3{#1=#2=#3=}
\foo^^^0153^^^0152^^^0178
^^^017b^^8?

```

It is translated by Tralics as &#x153;=&#x152;=&#x178;=&#x17B;x?. The argument to \foo could also have been: \oe\OE{"Y}. The transcript file contains lines of the form:

```

[8] \foo^^^0153^^^0152^^^0178
\foo #1#2#3->#1=#2=#3=
#1<-^^^0153
#2<-^^^0152
#3<-^^^0178

```

It is possible to ask for UTF-8 output in the transcript file. This gives characters that are hard to see using latin1, because characters in the range 128–128+32 are in general unprintable. What is shown here as hat-Ó is a single character.

```

[2] \foo^^^0153^^^0152^^^0178
\foo #1#2#3->#1=#2=#3=

```

```

#1<-Ã^Ó
#2<-Ã^Ò
#3<-Ã,
{Push p 1}
Character sequence: Ã^Ó=Ã^Ò=Ã,= .

```

The original version of the *Tralics* documentation said: Si on a un texte qui contient essentiellement des caractères 7bits, et très peu d'autres caractères, l'utilisation de caractères 16bits consomme énormément de place. This means that using a 16bit encoding consumes a lot of space if you write a French document (and even more, for an English one). The sentence has 159 ASCII characters and 6 others; these can be input using iso-8859-1 (aka latin-1) as input encoding<sup>2</sup>. In  $\TeX$ , it uses 165 bytes, in  $\Omega$ , it uses 330 bytes. Using a construction like `\'e` we need 177 bytes (and 7 bits per byte). Using UTF-8 requires only 171 bytes (8 bits per byte). This explains why UTF-8 is popular. We shall explain (in the second part of this document) how UTF-8 is encoded and how  $\TeX$  may read it. In the case of *Tralics*, the situation is: you can (via an argument to the *Tralics* program) specify that the sources are encoded using UTF-8 or latin1 (this being the default). However, if the tex file contains, on the first line "utf8-encoded" UTF-8 encoding will be used, if it contains "iso-8859-1" then latin1 encoding will be used.

## 5.2 New encoding scheme

Since version 2.9, internal encoding of *Tralics* is 16bit utf8. This has two consequences that will be explained here. The first is that some tables are now much larger. The numeric argument to `\catcode`, `\mathcode`, `\lccode`, `\uccode`, `\sfcode`, `\delcode`, which is a character number can now be anything between 0 and 65535. We also changed the numbers of registers: there are 1024 instead of 256.

The result of a `~~~~abcd` construct fits on 16bits, hence is a character, hence can appear in a command name (in the case of a multicharacter control sequence, it must have category code 'letter'; initially all character with code greater than 128 have category 'other'). In order to save space, a short-verb character must fit on 8bits; otherwise, its category code will not be properly restored when you undeclare it (category other will be used).

All characters are valid in math mode. The translation of an ASCII character may depend on the font, otherwise, it is always `<mi>`. For instance, in the case of `\mathbf{\'e}`, expansion of the accent command produces a 8bit character, unaffected by the font change, and the translation is a `<mi>` containing the e-acute letter. Full 21 bit characters are allowed in Math mode. An expression `\$x\$` is considered trivial math and translates into a `<simplemath>` element only if the character fits on seven bits and has category letter.

The default input and output encoding is latin1, which is no more the internal encoding. As a consequence, there are two conversion procedures. We explained above that the input encoding can be given on the first line of the file. Otherwise a default encoding will be used. This can be explained in the configuration file. As a consequence, the main input file is read without conversion, then the configuration file is considered, and then the main input file is converted; all other files are immediately converted.

On the other hand, a character like é is represented as `Ã©` in the internal tree. This character will appear, in the output file, in the form `&#e9`; if you call *Tralics* with option `-oe8a` or `-oe1a`, as é if you call *Tralics* with option `-oe1` or `Ã©` if you call *Tralics* with option `-oe8`. If the option contains a, the XML file contains only 7bit ASCII characters; the only difference between the two options is the encoding declaration. These options specify also the encoding used for the transcript file. You can specify it independently with the options `-te8a`, `-te1a`, `-te8`, or `-te1`. If the character is too

<sup>2</sup>All French letters exist in iso-8859-1, except `\oe`  $\Omega$ E, `\'Y`, and the Euro sign

big to fit in the encoding, then the hat-hat notation is used (see example above). Because each XML file contains its encoding, a XML processor will handle the file produced by Tralics independently of the output encoding. Moreover, whatever the encoding, input or output, you know that `^^^03b7` is Greek letter eta.

### 5.3 Changing the input encoding

We mentioned in the previous section that whenever Tralics reads a file, it converts its content, according to the current encoding (that can be given at the start of the file, using ASCII characters), with an exception for the main input file. The situation is a bit more complex: configuration files, tcf files, bibliography data files, and T<sub>E</sub>X files opened by `\openin` use a fixed encoding; other source files use a variable encoding.

The default encoding is stored in `\input@encoding@default`. The default value is one, but can be changed via an option to the program (utf8 or latin1 select encoding 0 or 1 respectively).

The current encoding is stored in `\input@encoding`. This is an attribute of the current input file, it can be changed at any time. The new encoding is used when Tralics needs to read a new line in order to fetch the next token. Nothing special is done in the case of `\read`.

Whenever a file is opened, its initial encoding is computed. If the file has a fixed encoding, then all lines are immediately converted, otherwise lines are converted when needed. If the first line of the file contains the string `utf8-encoded`, then encoding 0 is assumed, if the line contains `iso-8859-1`, then encoding 1 is assumed, and if the line contains `tralics-encoding:NN` where NN is a sequence of one or two digits forming a number less than 34, then encoding NN is assumed. There are other heuristics. For instance, if `%&TEX encoding = UTF-8` appears near the start of the file, then encoding 0 is assumed. In all other cases, the default encoding is assumed.

In the current version of Tralics, there are 34 possible encodings. Encoding number 0 is UTF8; this is an encoding where an ASCII character is represented by a single byte (with the same value as the character), and other characters use a variable number (between 1 and 4) of bytes. In encodings like UTF16, a character is represented by more than one byte. There is currently no support for such encodings yet. Stated otherwise, we assume that character C is represented by a byte B, and the encoding specifies the value C at position B. Encoding 1 is latin1 (also known as iso-8859-1), it has B=C. For the 32 remaining encodings, it is possible to specify, for each byte B, the associated character C (default is B). Trying to set the current or default encoding to a value outside the range 0-33 is ignored; trying to modify an encoding outside the range 2-33 raises an *Illegal encoding* error, and invalid byte value gives *Illegal encoding position* error. In case of an illegal character value (negative, zero, 65536 or more), the byte value is used instead. The magic command is `\input@encoding@val`; it reads an encoding, a byte and a value. In the example that follows we change the encoding number 2 so that `\F00` is read as `\foo`:

```

1 \input@encoding@val 2 '0 ='o
2 \input@encoding@val 2 'F ='f
3 \let\foo\bar
4 \showthe\input@encoding@val 2 '0
5 \input@encoding=2
6 \show\F00
7 \showthe\input@encoding@val 2 '0
8 \showthe\input@encoding
9 \input@encoding@default=0
10 \showthe\input@encoding@default
11 \input@encoding=1

```

This example shows three commands in read or write mode: when the command is prefixed by `\showthe` it read a value from memory and prints it on the terminal, otherwise a number is scanned and written in memory. The equals signs before the number is optional. No less than 13 integers are scanned, some are given as an explicit integer, some as a character code. We assume that, for encoding 2, all characters map to themselves. Since `\FOO` is read as `\foo`, the `\show` command should print `\bar`, on lines 4 and 7 you see the value stored of encoding 2 for the character O (first upper case, then lower case), this is twice 111. Other values shown are 2 and 0.

We describe from now on the content of the `inputenc` package. You load it by saying `\usepackage [foo,bar] {inputenc}`. The effect of this command is the following. First, a symbol name is defined for each of the 23 known encoding, for instance `utf8` for UTF-8 (encoding 0), `latin1` for latin1 (encoding 1), etc. The command `\inputencodingname` holds the current input coding name, and `\encoding@value` converts this to an integer. The command `\inputencoding` can be used to change the encoding. It is defined as:

```

12 \def\inputencoding#1{%
13   \the\inpenc@prehook %% pre-hook
14   \edef\inputencodingname{#1}%
15   \input@encoding=\encoding@value{\inputencodingname}%
16   \the\inpenc@posthook} %% post-hook

```

There are two hooks (token lists) that do nothing, added here for compatibility with the `LATEX` package. You can use them to output as messages, such as: switching from encoding A to encoding B (the initial value of the encoding name is `\relax`, this can be used by the pre-hook).

The options, `foo` and `bar` in the example, should be valid names. The last name becomes the current and default encoding. As mentioned above, the current encoding applies to an input file, and there is no reason to change the encoding of the package file. Hence, the following is executed:

```

17 \input@encoding@default\encoding@value{bar}%
18 \AtBeginDocument{\inputencoding{bar}}

```

If the options are, for instance `ansinew` and `applemac`, the tables associated to these encodings are defined; some other tables might also be defined, but you should not rely on this (of course, `latin1` and `utf8`, can be used anywhere, because they are builtin). The package contains

```

19 \edef\io@enc{\encoding@value{latin9}}
20 \DeclareInputText{164}{"20AC}
21 \DeclareInputText{166}{"160}
22 \DeclareInputText{168}{"161}
23 \DeclareInputText{180}{"17D}
24 \DeclareInputText{184}{"17E}
25 \DeclareInputText{188}{"152}
26 \DeclareInputText{189}{"153}
27 \DeclareInputText{190}{"178}

```

The code above defines the `latin9` (iso-8859-15) encoding. It is very like `latin1`, but defines the Euro sign at position 164. Defining 256 characters per encoding using this method is inefficient. For this reason you can see

```

28 \input@encoding@val \encoding@value{latin2} -96 160
29 160 "104 "306 "141 164 "13D "15A 167

```

As explained above, the command on the start of the line reads 3 integers: an encoding value (here, the encoding of `latin2`), a byte position and a character value. The byte position must a number between 0 and 255. Here we use an extension: If a negative number minus N has been read, followed by A such that the sum of A and N is at most 256, then N values will be read, and stored at position A and following (here N is 96, and we have shown only the first eight values).



## 5.4 Characters and Accents

There are some commands that put an accent over a letter. You can say `a\accent 98 cde`, this works in T<sub>E</sub>X, but not in Tralics: you will get an error, *Unimplemented command \accent*. The number 98 is read, and converted to an integer. The Unicode character will be used; thus the translated result is ‘abcde’.

You can say `\a’e`. This is a command introduced by L<sup>A</sup>T<sub>E</sub>X so as to allow accents inside a tabbing. Some care must be taken. If you say `\a{par}{b}` in L<sup>A</sup>T<sub>E</sub>X, you get an error of the form: *Paragraph ended before \@changed@cmd was complete*. The Tralics error message is: *wanted a single token as argument to \a*. If you say `\a\foo12`, there is a single token, and the error is: *Bad syntax of \a, argument is \foo*. In fact, the token after `\a` must be a valid accent character. After that `\a’` is handled exactly like `\’`. You can say `\={ U}`, the space after the command is ignored. You cannot say `\={ U}`, the space is not removed, this is an error. In fact, the argument list of the accent command should contain exactly one token (exception: double accents will be explained later). This token should be a character, with code between 0 and 128. Hence `\’É` is wrong, you must say `\’{\^E}` if you want  $\acute{E}$ . The message is *Error in accent, command = \’; Cannot put this accent on non 7-bit character É*. If the token `\i` is given, it will be replaced by `i`, so that `\"i` and `\"i` produce the same result. You can say `\=\AE`, `\=\ae`, `\’\AE`, `\’\ae`, `\’\AA`, `\’\aa`, `\’\O`, `\’\o`. The result looks like  $\bar{E}\bar{e}\acute{E}\acute{e}\acute{A}\acute{a}\acute{O}\acute{o}$ .

You can put an accent on a letter only in the case where this gives a Unicode character. In the case of `\c{a}` and `\c{\=a}`, the error message is the same: *Error in accent, command = \c; Cannot put this accent on letter a*. Table 5.1 indicates on which letters you can put an accent. See the html page <http://www-sop.inria.fr/apics/tralics/doc-chars.html> for a list of some glyphs.

Some accents are not standard. Examples:

- `\^a` gives  $\hat{a}$ ,
- `\’ a` gives  $\acute{a}$ ,
- `\‘ a` gives  $\grave{a}$ ,
- `\" a` gives  $\ddot{a}$ ,
- `\c c` gives  $\mathfrak{c}$ ,
- `\u a` gives  $\underset{\sim}{a}$ ,
- `\v a` gives  $\underset{\vee}{a}$ ,
- `\~ a` gives  $\tilde{a}$ ,
- `\H o` gives  $\acute{o}$ , it is redefined in the case of double accents,
- `\k a` gives  $\mathfrak{a}$ ,
- `\. a` gives  $\grave{a}$ ,
- `\= a` gives  $\bar{a}$ ,
- `\r a` gives  $\mathring{a}$ ,
- `\b b` gives  $\mathfrak{b}$ ,
- `\d a` gives  $\mathfrak{a}$ ,
- `\f a` gives an inverted breve accent over a,
- `\C a` gives a double grave accent on a,
- `\T e` gives a tilde under e,
- `\V d` gives a circumflex below d,
- `\D a` gives a ring below a,
- `\h a` gives a hook over a.

If in the table you see ‘I’ instead of ‘x’, this means that the accent applies only on capital I. If you see h, j, t, w or y, this applies only to the lower case letter. Otherwise the accent applies to both upper case letter and lower case letter.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
$\tilde{\phantom{x}}$	x	x	x	x	x	x	x	x	x						x			x	x	x	x	x	x	x	x	x
$\acute{\phantom{x}}$	x	x	x	x	x				x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x
$\grave{\phantom{x}}$	x			x				x						x						t	x	x	x	x		
$\mathring{\phantom{x}}$	x		x	x	x	x	x			x	x		x					x	x	x						
$\mathring{u}$	x			x			x	x						x							x					
$\mathring{v}$	x	x	x	x	x	x	x	x	j	x	x	x	x	x	x			x	x	x	x					x
$\mathring{\sim}$	x			x				x						x	x						x	x				x
$\mathring{H}$															x						x					
$\mathring{k}$	x			x				x							x						x					
$\mathring{\cdot}$	x	x	x	x	x	x		x	I		x	x	x	x	x	x		x	x	x			x	x	x	x
$\mathring{=}$	x			x			x	x	x						x						x	x				x
$\mathring{r}$	x																				x	w				y
$\mathring{b}$		x		x				h		x	x		x					x	x							x
$\mathring{d}$	x	x		x	x			x	x	x	x	x	x	x	x			x	x	x	x	x	x	x		x
$\mathring{f}$	x			x				x							x			x			x					
$\mathring{C}$	x			x				x							x			x			x					
$\mathring{T}$				x				x													x					
$\mathring{V}$			x	x							x	x									x	x				
$\mathring{D}$	x																									
$\mathring{h}$	x			x				x							x						x					x

Table 5.1: *All possible accents.* You can put an accent on any letter, except Q. You can put accents on non-letters, for instance  $\mathring{ae}$ , see text. Some characters accept two accents. In general, you can put an accent on a lower case letter, an upper case letter. There is one exception: you cannot put a dot over a lower case I, because there is already a dot. For h, j, t w, and y, there are accents that apply only to lowercase letters.





```
test ligatures&#xA0;:<hi rend='tt'>&lt;&lt;&gt;&gt;‘ ’--&#xA0;et
&#xA0;---&#xA0;!?:;</hi>
```

The soul package provides some commands. Example; `\ul` gives test for ul, `\so` gives test for so, `\st` gives test for st, `\caps` gives TEST FOR CAPS, `\hl` gives **test for hl**.

## 5.5 Verbatim material

We have seen a little example of verbatim code above. It shows that some `&#x200B;` characters are inserted, this is so that, if the XML file is read, a double dash will not be interpreted as an en-dash. What the `\verb` command produces is a sequence of characters, whose category codes are 12, except for some, that are of category 11, namely ‘’-<>~&:;!«». You can compare this with the L<sup>A</sup>T<sub>E</sub>X code, shown in section 2.12: the `\@noligs` command makes some characters of category code 13, the associated action is: output the character, with a zero kern in front. There is an exception: the space character is replaced by the `\nobreakspace` token, but this can be changed.

You can say `\verb*+x y+` or `\verb+ x y+`. All characters between the two plus signs are collected. Any character can be used instead of the plus sign (Try `\verb*abca` and `\verb =a= !`). In the case where a star is given, spaces are replaced by `\textvisiblespace`, otherwise by `\nobreakspace`. You can say `\DefineShortVerb\+`, after that `+foo+` is the same as `\verb+foo+`. Note that the command must be followed by something like `\+` or `\*`, i.e., a macro whose name is formed of a single character. You can say `\UndefineShortVerb\+`, this will undo the previous command. The syntax is the same. If the character fits on 8 bits, the old category code is restored; otherwise, it is set to 12 (other). Note: assume that the input encoding is latin1, but you declare `^^^^abcd` as a short verb. When Tralics sees the four hats, it replaces these 8 bytes by a single character, say *C*, and enters verbatim mode until finding character *C*. Since this character does not exist in the current environment, it cannot be found directly; since we are in verbatim mode, it cannot be found using the four-hat construction. For this reason an error is signalled when the end of line is reached (an implicit *C* character is inserted, so that next line will be translated normally).

In the case where ‘+’ is a short verb character, you can say `\SaveVerb{foo}+\bar+`. This has as effect to remember in a private command all tokens that `+\bar+` gathers. When you say `\UseVerb{foo}`, these tokens are re-inserted in the input stream. Example:

```
\DefineShortVerb\+
\SaveVerb{foo}+\bar +
\UndefineShortVerb\+
\UseVerb{foo}
```

The transcript file will contain, for the `\UseVerb` command the following line.

```
\savedverb@foo ->\verbprefix {\verbatimfont \bar\nobreakspace }
```

Here, the `\` before ‘b’ is not a command delimiter, for otherwise there would have been a space after `\bar`. Note: another explanation is that the ‘b’ is not of category code 11, so that the command is `\b`; exercise: find all interpretations of this line.

There are various packages that provide a `verbatim`-like environment. In Tralics, you can define your own via

```
\DefineVerbatimEnvironment{MyVerbatim}{Verbatim}{xx=yy}
```

This defines `MyVerbatim` to be an environment that behaves like `Verbatim`, that is an extension of the basic `verbatim` environment that takes some optional parameters (here, the default value of `xx` is `yy`). The end of a verbatim environment is defined as a line that contains optional spaces, the `\end` token, optional spaces, the name of the environment enclosed in braces. Additional characters on the current line are assumed to be after the verbatim environment.

In the case of a verbatim environment, all characters on the line are gathered (final spaces disappear, as usual), with category codes as explained above. If this gives an empty list, a no-break space character is added<sup>5</sup>. As is the case of `\verb`, the `\verbatimfont` command is prepended. This is defined to be `\tt`. Moreover, `\verbatimprefix` is also added in front of the token list. In the case of the `\verb` command, there is `\verbprefix` instead. These two commands are defined as `\@empty`. You can redefine them. Each line is followed by `\par` and `\noindent`. If the environment is followed by an empty line, or a `\par` command, this command is removed, as well as the last `\indent`. Example that shows use of the prefix commands:

```
\DefineShortVerb{\|}
\def\verbatimfont#1{{#1}}
\def\verbprefix#1{A#1A}
\def\verbatimprefix#1{B#1B}
Test: \verb+foo+ and |bar|
\UndefineShortVerb{\|}
\begin{verbatim}
line1
line2
\end{verbatim}
```

The translation is:

```
<p>Test: AfooA and AbarA</p>
<p noindent='true'>Bline1B</p>
<p noindent='true'>Bline2B</p>
<p noindent='true'></p>
```

The Verbatim environment is an extension of the verbatim environment. There is an optional argument, an association list. If you say `'numbers=true'`, then lines will be numbered (instead of `'true'`, you can say `'left'` or `'right'`, or anything, the value is ignored). If you say `'counter=17'`, then lines will be numbered, using counter 17, if you say `'counter=foo'`, and `'foo'` is a counter name, then lines will be numbered, using counter foo. If you say `'firstnumber=N'`, where N is a number, then lines will be numbered starting from N; if you say `'firstnumber=last'`, then lines will be numbered incrementing the previous value. The default counter is `FancyVerbLine`. Other features defined by the `fancyvrb` package have not yet been implemented.

If a line number M is given, the following piece of code is inserted before the verbatim line: `\verbatimnumberfont{M}\space`. The funny command is `\let` equal to `\small` at the start of the run. The number is incremented for each line.

Characters after `\begin{Verbatim}`, but on the same line, are ignored. The same is true if an optional argument is given: all characters that follow the closing bracket of the optional argument are ignored. The opening bracket is only looked for on the current line (unless the end of line character is commented out).

```
\begin{Verbatim}                                [numbers=true]
TEST
\end{Verbatim}
and without
\begin{Verbatim}
[ok]TEST
\end{Verbatim}
\begin{Verbatim} %
```

<sup>5</sup>This is because empty paragraphs are generally removed. With this hack, an empty verbatim line does not disappear

```

[ok] this is handled as comment
TEST
\end{Verbatim}

\def\verbatimfont#1{\it #1}
\def\verbatimnumberfont{\large}
\tracingall
\count3=4
\begin{Verbatim}[counter=3]
5,one line
\end{Verbatim}
\begin{Verbatim}[counter=03]
6,one line
\end{Verbatim}
\newcounter{vbcounter}
\setcounter{vbcounter}8
\begin{Verbatim}[counter=vbcounter]
9,one line
\end{Verbatim}
\begin{Verbatim}[counter=vbcounter]
10,one line
\end{Verbatim}

```

This is the translation.

```

<p noindent='true'><hi rend='small1'>1</hi> <hi rend='tt'>TEST</hi></p>
<p noindent='true'>and without</p>
<p noindent='true'><hi rend='tt'>[ok]TEST</hi></p>
<p noindent='true'></p>
<p noindent='true'><hi rend='tt'>TEST</hi></p>
<p noindent='true'><hi rend='large1'>5</hi> <hi rend='it'>5</hi>,one line</p>
<p noindent='true'><hi rend='large1'>6</hi> <hi rend='it'>6</hi>,one line</p>
<p noindent='true'></p>
<p noindent='true'><hi rend='large1'>9</hi> <hi rend='it'>9</hi>,one line</p>
<p noindent='true'><hi rend='large1'>10</hi> <hi rend='it'>1</hi>0,one line</p>
<p noindent='true'></p>

```

Two additional keywords have been added. In order to be compatible, you should add the following code to the T<sub>E</sub>X document.

```

\csname define@key\endcsname{FV}{style}{}
\csname define@key\endcsname{FV}{pre}{}

```

If you say `style=foo`, then the token `\FV@style@foo` is added in front of the token list generated by the `verbatim` environment. If you say `pre=bar`, then `\FV@pre@bar` is added before the token list (and before the style token mentioned above), and `\FV@post@bar` is inserted near the end (to be precise: before the last `\par` or `\par\noindent`). For a case like this

```

\begin{Verbatim}[pre=pre,style=latex,numbers=true]
first line
second line
\end{Verbatim}
third line

```

the tokens gathered by the `verbatim` environment, shown in the transcript file in verbose mode, and re-indented in order to make the structure easy to recognise, are

```

{Verbatim tokens:
\FV@pre@pre \FV@style@latex
\par \noindent {\verbatimnumberfont {1}}
\verbatimprefix {\verbatimfont first\nobreakspace line}
\par \noindent {\verbatimnumberfont {2}}
\verbatimprefix {\verbatimfont second\nobreakspace line}
\FV@post@pre
\par \noindent }

```

Assume that the following definitions are given

```

\def\FV@pre@pre{\begin{xmlelement*}{pre}}
\def\FV@post@pre{\end{xmlelement*}}
\def\FV@style@xml{\XMLaddatt{class}{xml-code}}
%\def\verbatimnumberfont#1{\xbox{vbnumber}{#1}}

```

Then the translation is

```

<pre class='latex-code'>
<p noindent='true'>
  <hi rend='small'>1</hi>
  <hi rend='tt'>first&nbsp;line</hi></p>^^J
<p noindent='true'>
  <hi rend='small'>2</hi>
  <hi rend='tt'>second&nbsp;line</hi></p>^^J
</pre><p noindent='true'>third line^^J
</p>

```

Note: We have re-indented a little bit the code, and marked newline characters by ^^J. As you can see, each verbatim line gives exactly one line in the XML output, and this line is formed of a <p> element. If you apply a style sheet with the following definition

```

<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:apply-templates/>
      </p>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

```

then <p> elements are discarded in a <pre>, and some action is done in case of noindented paragraphs. If moreover the translation of <pre> is defined by the following code

```

<xsl:template match="pre">
  <pre>
    <xsl:attribute name="class">
      <xsl:value-of select="@class"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </pre>

```



```

    <xsl:text>&#x0A;</xsl:text>
  </xsl:template>
we get finally
  <pre class="latex-code"><small>1</small> <tt>first line</tt>
  <small>2</small> <tt>second line</tt>
</pre>
  <p class="nofirst noindent">third line</p>

```

This is not valid HTML, since `<small>` is forbidden in a `<pre>`. We can modify the style sheet so that if `<hi>` is in a `<pre>`, then a special action is taken in the case `rend='small'`; we can also remove the useless `<tt>`. A better solution: we uncomment the definition of `\verbatimnumberfont`. This will have as effect that verbatim line numbers will be in a `<vbnumber>` element, and we can apply the following transformation.

```

  <xsl:template match="vbnumber">
    <span class='prenumber'>
      <xsl:apply-templates/>
    </span>
  </xsl:template>

```

Thus, the HTML code will be

```

  <pre class="latex-code"><span class="prenumber">1</span> first line
  <span class="prenumber">2</span> second line
</pre>
  <p class="nofirst noindent">third line</p>

```

This document was converted into HTML using the techniques shown here. The style sheet changes the background color of the `<pre>` element, according to its class, and the background of the `<span>` to the background of the page.

Note how the `'style'` option of the verbatim environment gives a `'class'` attribute in HTML document. If you say

```

\DefineVerbatimEnvironment{verbatim}{Verbatim}
{listparameters={\topsep0pt },pre=pre}

```

then `verbatim` behaves like `Verbatim`, said otherwise, an optional argument is scanned. Moreover, the list on the second line will be put in `\verbatim@hook`; whenever a verbatim environment of type `'Something'` is read, the value of the command `\Something@hook` is considered (this should be undefined or a command that takes no argument), and the tokens are added to the optional argument, before other arguments.

You can say `\numberedverbatim` or `\unnumberedverbatim`. After that, verbatim environments will be automatically numbered or not. This does not apply to `Verbatim` environments.

There is a command `\fvset` that takes an associated list as argument. If it contains `'showspases=true'` or `'showspases=false'`, this changes how spaces are interpreted in a verbatim environment or command (except for `\verb*`, case where the space is always visible).

## 5.6 Case change

There are different commands for changing the case of letters. For instance, the translation of

```

\uppercase{Einstein: $E=mc^2$}
\lowercase{Einstein: $E=mc^2$}

```

is

```

<p>EINSTÉIN: <formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow><mi>E</mi><mo>=</mo><mi>M</mi><msup><mi>C</mi><mn>2</mn></msup>
</mrow></math></formula>
einstéin: <formula type='inline'>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow><mi>e</mi><mo>=</mo><mi>m</mi><msup><mi>c</mi><mn>2</mn></msup>
</mrow></math></formula>
</p>

```

There are two tables that control these conversions: the lc-table and the uc-table. If the lc value of a character is non-zero, it's the lowercase equivalent of the character; otherwise, the character is left unchanged by `\lowercase`. The same is true for the uc-table. You can use `\lccode` and `\uccode` for changing these tables. They are initialized like this: for all integers  $x$  with value between 'a' and 'z', and between 'à' and 'ÿ', the uc value is  $x - 32$ , the lc value is  $x$ , the same holds for  $x - 32$ . There are four exceptions: the pair 215, 247, this is multiplication and division sign, and the pair 223, 255 this is  $\beta$  and  $\ddot{y}$ . On the other hand, we used the pair 255, 376 (for  $\ddot{y}$  and  $\ddot{Y}$ ).

You can use the two commands `\MakeUppercase` and `\MakeLowercase`. These commands have a regular syntax (in the example that follows, the `\expandafter` would be useless for `\lowercase`). They convert letters, as for `\uppercase` and `\lowercase`, plus some commands that define some characters. This example shows the list of all the recognised commands.

```

\def\foo{foo}
\def\List{{abcABCÉÉ\foo
\oe\OE\o\O\ae\AE\dh\DH\dj\DJ\l\L\ng\NG\ss\SS\th\TH}}
\expandafter\MakeUppercase\List
\expandafter\MakeLowercase\List

```

The translation is

```

ABCABCÉÉfoo&#x152;&#x152;ØØÆÆÐÐ&#x110;&#x110;&#x141;&#x141;&#x14A;&#x14A;SSSSÞÞ
abcabcééfoo&#x153;&#x153;øøææðð&#x111;&#x111;&#x142;&#x142;&#x14B;&#x14B;ßßþþ

```

Since Tralics version 2.9, all commands listed above expand to characters, that have a non-trivial uc/lc pair. Hence, you can say:

```

\def\foo{foo}
\edef\List{{abcABCÉÉ"y"Y\foo
\ij\IJ\oe\OE\o\O\ae\AE\dh\DH\dj\DJ\l\L\ng\NG\ss\SS\th\TH}}
\expandafter\uppercase\List
\expandafter\lowercase\List

```

## 5.7 Simple commands

We consider here some commands that take no arguments. Unless told otherwise, they are not allowed in math mode. A new paragraph is started (via `\leavevmode`) in vertical mode.

- `\#` translates to `#`, character U+23, ok in math mode.
- `\-` is `\discretionary{-}{-}{-}` in  $\LaTeX$ , empty in Tralics.
- `\AA` and `\aa` translate to  $\text{\AA}$  and  $\text{\aa}$ , characters U+C5 and U+E5, ok in math mode, accepts some accents on it.
- `\AE` and `\ae` translate to  $\text{\AE}$  and  $\text{\ae}$ , characters U+C6 and U+E6, ok in math mode, accepts some accents on it.
- `\dag` translates to  $\text{\dag}$ , character U+2020. This is the same character as produced by the math only command `\dagger`, or the alternate name `\textdagger`.

- `\ddag` translates to ‡, character U+2021. This is the same character as produced by the math only command `\ddagger`.
- `\DH` and `\dh` translate to Đ and đ, characters U+D0 and U+F0, ok in math mode.
- `\DJ` and `\dj` translates to Đ and đ, characters U+110 and U+111.
- `\endguillemets` expands to »; character U+BB. Is the same as `\guillemotright`. You should use this as the environment `guillemets`.
- `\fg` translates to »; this is U+A0 (no-break space) followed by U+BB.
- `\guillemets` expands to «. Is the same as `\guillemotleft`. You should use this as the environment `guillemets`.
- `\ieme` is the same as `\textsuperscript{e}\xspace`. Something like `3\ieme` should typeset as 3<sup>e</sup>.<sup>6</sup>
- `\iemes` is the same as `\textsuperscript{es}\xspace`. Something like `3\iemes` should typeset as 3<sup>es</sup>.
- `\ier` is the same as `\textsuperscript{er}\xspace`. Something like `1\ier` should typeset as 1<sup>er</sup>.
- `\iers` is the same as `\textsuperscript{ers}\xspace`. Something like `1\iers` should typeset as 1<sup>ers</sup>.
- `\iere` is the same as `\textsuperscript{re}\xspace`. Something like `1\iere` should typeset as 1<sup>re</sup>.
- `\ieres` is the same as `\textsuperscript{res}\xspace`. Something like `1\ieres` should typeset as 1<sup>res</sup>.
- `\LaTeX` translates to `<LaTeX/>`.
- `\No` and `\Numero` is the same as `N\textsuperscript{o}\xspace`. This should render as N<sup>o</sup>.
- `\no` and `\numero` is the same as `n\textsuperscript{o}\xspace`. This should render as n<sup>o</sup>.
- `\O` and `\o` translates to Ø and ø, characters U+D8 and U+F8, ok in math mode, accepts some accents.
- `\og` translates to «; this is U+AB followed by U+A0 (no-break space).
- `\P` translates to ¶, this is character U+B6. This is like `\textparagraph`, but allowed in math mode.
- `\S` translates to §, this is character U+A7. This is like `\textsection`, but allowed in math mode.
- `\slash` translates to /. No penalty is added.
- `\SS` and `\ss` translate to SS and ß.
- `\TeX` translates to `<TeX/>`.
- `\TH` and `\th` translate to Þ and þ, characters U+DE and U+FE, ok in math mode.

The following commands all start with ‘text’. They are forbidden in math mode.

- `\textasciicutex` translates as the Unicode character U+2032; this is known as ‘prime’. It is not the same as U+27, apostrophe, or U+B4 acute accent, or U+2B9, modifier letter prime.
- `\textasciicircum` translates to ^, character U+2303 .
- `\textasciigrave` translates to `, character U+2035.
- `\textbackslash` translates to \, character U+5C.
- `\textdollar` translates to \$, character U+24.
- `\textnospace` translates to U+200B (zero width space). We use this as a mark to inhibit ligatures in verbatim mode.
- `\textvisiblespace` translates as □, character U+2423.

---

<sup>6</sup>In a previous version the translation was an entity, named `&ieme`, defined by the Raweb. Same remark for the commands that follow, and `\numero`.



- `\textsoftyphen`, translates as soft-hyphen, Unicode character U+AD, (discretionary hyphen).
- `\textregistered` translates as ®, character U+AE.
- `\textasciimacron` translates as ¯, character U+AF.
- `\textdegree` translates as °, character U+B0.
- `\textpm` translates as ±, character U+B1.
- `\texttwosuperior` translates as <sup>2</sup>, character U+B2.
- `\textthreesuperior` translates as <sup>3</sup>, character U+B3.
- `\textasciiacute` translates as ´, character U+B4.
- `\textmu` translates to μ, character U+B5.
- `\textparagraph` translates to ¶, character U+B6.
- `\textpilcrow` translates to ¶, character U+B6. In L<sup>A</sup>T<sub>E</sub>X, the translation is not the same as `\P`;
- `\textperiodcentered` translates to ·, character U+B7.
- `\textasciicedilla` translates to a cedilla, character U+B8.
- `\textonesuperior` translates to <sup>1</sup>, character U+B9.
- `\textordmasculine` translates to <sup>o</sup>, character U+BA.
- `\guillemotright` translates to »), this is character U+BB.
- `\textonequarter` translates to ¼, character U+BC.
- `\textonehalf` translates to ½, character U+BD.
- `\textthreequarters` translates to ¾, character U+BE.
- `\textquestiondown` translates to ¿, character U+BF.
- `\textflorin` translates to ₣, character U+192.
- `\textasciibreve` translates as ˘, Unicode U+306.
- `\textasciicaron` translates as ˇ, Unicode U+30C.
- `\textbaht` translates to ฿, character U+E3F.
- `\textendash` translates as –, Unicode U+2013.
- `\textemdash` translates as —, Unicode U+2014.
- `\textbardbl` translates as ||, Unicode U+2016.
- `\textquoteleft` translates as ‘, Unicode U+2018.
- `\textquoteright` translates as ’, Unicode U+2019.
- `\textdagger` translates to †, character U+2020.
- `\textdaggerdbl` translates to ‡, character U+2021.
- `\textbullet` translates to •, character U+2022.
- `\textellipsis` translates as …, Unicode U+2026.
- `\textquotedblleft` translates as “, Unicode U+201C.
- `\textquotedblright` translates as ”, Unicode U+201D.
- `\textperthousand` translates to ‰, character U+2030.
- `\textpertenthousand` translates to ‰, character U+2031.
- `\textacutedbl` translates as ¨, character U+2033.
- `\textgravedbl` translates to ` , character U+2036.
- `\textreferencemark` translates to ※, character U+203B.
- `\textinterrobang` translates to †, character U+203D.
- `\textfractionsolidus` translates as /, Unicode U+2044.
- `\textlquill` translates to {, character U+2045.
- `\textrquill` translates to }, character U+2046.
- `\textcolonmonetary` translates to ₯, character U+20A1.
- `\textfrenchfranc` translates to the symbol of the French Franc (not used any more), character U+20A3.

- `\textlira` translates to  $\text{₡}$ , character U+20A4.
- `\textnaira` translates to  $\text{₦}$ , character U+20A6.
- `\textwon` translates as  $\text{₩}$ , character U+20A9.
- `\textdong` translates to  $\text{₫}$ , character U+20AB.
- `\texteuro` translates to  $\text{€}$ , character U+20AC.
- `\textpeso` translates to  $\text{₱}$ , character U+20B1.
- `\textcelsius` translates to  $\text{°C}$ , character U+2103.
- `\textnumero` translates to  $\text{№}$ , character U+2116.
- `\textcircledP` translates to  $\text{©}$ , character U+2117.
- `\textrecipe` translates to  $\text{℞}$ , character U+211E.
- `\textservicemark` translates to  $\text{SM}$ , character U+2120.
- `\texttrademark` translates as  $\text{TM}$ , character U+2122.
- `\textohm` translates to  $\text{Ω}$ , character U+2126.
- `\textmho` translates to  $\text{℧}$ , character U+2127.
- `\textestimated` translates to  $\text{€}$ , character U+212E.
- `\textleftarrow` translates to  $\text{←}$ , character U+2190.
- `\textuparrow` translates to  $\text{↑}$ , character U+2191.
- `\textrightarrow` translates to  $\text{→}$ , character U+2192.
- `\textdownarrow` translates to  $\text{↓}$ , character U+2193.
- `\textsurd` translates to  $\text{√}$ , Unicode U+221A.
- `\textopenbullet` translates to  $\text{◦}$ , Unicode U+25E6.
- `\textbigcircle` translates to  $\text{◯}$ , Unicode U+25EF.
- `\textmusicalnote` translates to  $\text{♪}$ , Unicode U+266A.
- `\textlangle` translates to  $\text{⟨}$ , Unicode U+3008.
- `\textrangle` translates to  $\text{⟩}$ , Unicode U+3009.
- `\texttwelveudash` translates as  $\text{—}$ .
- `\textthreequartersemdash` translates as  $\text{—}$ .
- `\textasteriskcentered` translates as  $\text{*}$ .
- `\textquotesingle` translates as  $\text{'}$ .
- `\textquotestraightbase` is not yet implemented.
- `\textquotestraightdblbase` is not yet implemented.
- `\textlbrackdbl` is not yet implemented.
- `\textrbrackdbl` is not yet implemented.
- `\textcompwordmark` is not yet implemented.
- `\texttildelow` is not yet implemented.
- `\textcentoldstyle` is not yet implemented.
- `\textdollaroldstyle` is not yet implemented.
- `\textcopyleft` is not yet implemented.
- `\textzerooldstyle` is not yet implemented.
- `\textoneoldstyle` is not yet implemented.
- `\texttwooldstyle` is not yet implemented.
- `\textthreeoldstyle` is not yet implemented.
- `\textfouroldstyle` is not yet implemented.
- `\textfiveoldstyle` is not yet implemented.
- `\textsixoldstyle` is not yet implemented.
- `\textsevenoldstyle` is not yet implemented.
- `\texteightoldstyle` is not yet implemented.
- `\textnineoldstyle` is not yet implemented.
- `\textcapitalcompwordmark` is not yet implemented.

- `\textguarani` is not yet implemented.
- `\textinterrobangdown` is not yet implemented.
- `\textdiscount` is not yet implemented.
- `\textascendercompwordmark` is not yet implemented.
- `\textdblhyphen` is not yet implemented.
- `\textleaf` is not yet implemented.
- `\textdied` is not yet implemented.
- `\textdivorced` is not yet implemented.
- `\textborn` is not yet implemented.
- `\textmarried` is not yet implemented.
- `\textblank` is not yet implemented.

## 5.8 The fp package

### Introduction

This is an implementation in C++ of the package by Michael Mehlich. It implements fixed point arithmetics in T<sub>E</sub>X. Each number is formed by a sign, then 18 digits before the point and 18 digits after the point. Since  $10^9 \leq 2^{30}$ , four 32bits integers are sufficient. In the code, we shall sometimes write a number as

$$x = \sum_{i=-6}^5 b_i B^i = 10^{-18} \left( \sum_{i=0}^{11} c_i B^i \right)$$

where  $B = 1000$ ,  $b_i$  and  $c_i$  are integers between 0 and 999. This requires 12 integers, instead of 4, but is useful for internal operations. You can say

```
\FPadd\foo{10}{3.5}
\FPmul\xbar\foo\foo
```

This will put 13.5 in `\foo` and 282.5 in `\xbar`. In verbose mode, you will see that the transcript file contains lines of the form:

```
{\FPadd}
{\FP@add}
{FPread for \FP@add=+10.}
{FPread for \FP@add=+3.5}
```

In reality, the first input line is converted into

```
\FP@add\foo10..\relax3.5..\relax
```

Most commands follow this scheme. There are some exceptions. You can use `\FPprint`. This takes one argument and prints (typesets) it. The algorithm is a bit strange: if the argument list is empty, the result is 0. If the argument is 123, or more generally a list of tokens, where the first has category code 12 (other), then nothing happens, the arguments is translated normally. If the argument is ‘foo’, the result is ‘13.5’. More generally, if the first item is a character not of category code 12, the command behaves like `\csname`. Don’t try constructions like `\FPprint{\$x^2$}`. You can say `\FPset{gee}{foo}`. The second argument is handled as for `\FPprint`. The first argument should be a command name, or a sequence of characters that becomes a command name via `\csname`. The effect of the command is the same as `\def\gee{13.5}`.

The general mechanism for a command like `\FPadd` or `\FPsincos` is to call intermediary commands like `\FP@add` or `\FP@sincos`. These read some command names (these must be definable, no check is made, as for `\let`), then parse numerical arguments, compute results and store the results in the commands. The result is always normalized: trailing zeroes are removed as well as

leading zeroes (but at least one digit is returned before the point). If the number is negative a sign is added. A special case is when the result is boolean. In this case the syntax has the form

```
\FPiflt{0.21}{0.20} Wrong\else Correct\fi
```

As a side effect, `\ifFPtest` is made equivalent to `\iftrue` or `\iffalse`. The following line is valid in `Tralics`, it gives an error in `LATEX`.

```
\iffalse \FPiflt{0.21}{0.20} \bad\else \badagain\fi \fi
```

Numbers are read as follows: We assume that `\FP@add` sees a string that contains two dots and a `\relax`, see above. This means that you lose if the argument of the user command contains a `\relax`. Otherwise, we have a list A, a dot, a list B, a dot, a list C, then `\relax`. As you can see from the `\FPMul\xbar\foo\foo` example, these quantities are obtained by expanding the argument (here `\foo`) in a `\edef`. For some reason quantities `\A` and `\B` are expanded again. In a case like

```
\FPadd\foo{\noexpand\noexpand\noexpand\V}{12}
```

this gives `\V` after expansion, this is wrong. In a case like

```
\def\V{10.2}
\FPadd\foo{\noexpand\V.4}{12}
```

expansion of A is 10.2, this is equally wrong: After expansion, there should remain only digits in A and B; there can be an optional sign at the start of A: any combination of + and - characters is OK. Note that C, as well as all digits after a space in A or B are ignored. Thus, the following two lines are valid for `Tralics`, invalid in the `TEX` case.

```
\def\V{10 .2}
\FPadd\foo{\noexpand\V.4}{12}
\FPadd\foo{\V.4}{12}
```

## The list of all commands

In the list that follows, `\C`, `\Ca`, `\Cb` are command names, and `\V`, `\Va`, `\Vb` are values.

- `\FPident\C\V`. This is the identity; it copies `\V` in `\C`.
- `\FPclip\C\V`. This puts in `\C` the number `\V`. In the `TEX` version, trailing zeroes are not removed by default. The `\FPclip` command has as function to removed them. In `Tralics` this is a no-op.
- `\FPprint\V`. See description above. This typesets `\V`.
- `\FPset\C\V`. See description above. This copies `\V` in `\C`.
- `\FPneg\C\V`. This copies the opposite of `\V` in `\C`.
- `\FPsgn\C\V`. This copies the sign of `\V`, that can be +1, -1 or 0, in `\C`.
- `\FPabs\C\V`. This puts in `\C` the absolute value of the number `\V`.
- `\FPadd\C\Va\Vb`. This computes the sum of `\Va` and `\Vb` and stores the result in `\C`.
- `\FPsub\C\Va\Vb`. This computes the difference of `\Va` and `\Vb` and stores the result in `\C`.
- `\FPMul\C\Va\Vb`. This computes the product of `\Va` and `\Vb` and stores the result in `\C`. The `TEX` code is beautiful: `Tralics` uses the same idea: we write each number as  $\sum b_i B^i$  and  $\sum c_i B^i$ , then compute the sum of  $b_i c_j B^{i+j}$ . 144 additions and multiplications are required. One can do better, using Karatsuba's method, but there is an additional cost for finding the  $b_i$  and merging at the end. This cost is 22 multiplications and divisions. The result is



```

\FP@@mul vv \relax\FP@saveshift%
\FP@@mul vu uv \relax\FP@saveshift%
\FP@@mul uu vt tv \relax\FP@saveshift%
\FP@@mul ut tu vz zv \relax\FP@saveshift%
\FP@@mul tt zu uz rv vr \relax\FP@saveshift%
\FP@@mul zt tz ur ru vq qv \relax\FP@saveshift%
\FP@@mul zz rt tr uq qu vp pv \relax\FP@saveshift%
\FP@@mul zr rz tq qt up pu vo ov \relax\FP@saveshift%
\FP@@mul rr qz zq tp pt uo ou vn nv \relax\FP@saveshift%
\FP@@mul rq qr zp pz to ot un nu vm mv \relax\FP@saveshift%
\FP@@mul qq rp pr zo oz tn nt um mu vl lv \relax\FP@saveshift%
\FP@@mul qp pq ro or zn nz tm mt ul lu kv vk \relax\FP@saveshift%
\FP@@mul pp oq qo rn nr zm mz tl lt ku uk \relax\FP@saveshift%
\FP@@mul op po nq qn rm mr zl lz tk kt \relax\FP@saveshift%
\FP@@mul oo pn np mq qm rl lr kz zk \relax\FP@saveshift%
\FP@@mul no on mp pm lq ql kr rk \relax\FP@saveshift%
\FP@@mul nn mo om pl lp qk kq \relax\FP@saveshift%
\FP@@mul mn nm lo ok pk kp \relax\FP@saveshift%
\FP@@mul mm ln nl ko ok \relax\FP@saveshift%
\FP@@mul lm ml kn nk \relax\FP@saveshift%
\FP@@mul ll km mk \relax\FP@saveshift%
\FP@@mul kl lk \relax\FP@saveshift%
\FP@@mul kk \relax\FP@saveshift%

```

Figure 5.2: A part of the T<sub>E</sub>X code of the multiplication in fp.

a number with 72 decimal digits. An overflow is signaled in case there are more than 18 decimals before the point. Last digits are silently ignored.

- `\FPdiv\C\Va\Vb`. This computes the quotient of `\Va` and `\Vb` and stores the result in `\C`. Division is much harder than multiplication. Given  $x$  and  $y$ , if

$$X = 10^n x \quad Y = 10^m y \quad q = Q10^{m-n}$$

then

$$X = QY \iff x = qy$$

We chose  $n$  and  $m$  as large as possible, then compute the quotient  $Q$  of  $X$  and  $Y$ . The quotient  $q$  is then obtained by shifting. By construction  $Q$  is between 10 and 1/10, shifting may underflow (this gives 0), or overflow (this is an error). When we compute the quotient of  $X$  by  $Y$ , we can multiply both numbers by  $10^{18}$ , said otherwise, consider them as integers. Write  $Q = \sum a_k 10^{-k}$ . Only indices  $k$  with  $k \geq 0$  are needed. In the case  $a_0 = 0$ , we know  $a_1 \neq 0$ . Write  $10^k Q = Q'_k + Q''_k$ , where  $Q'_k$  is an integer, and  $Q''_k$  is between 0 and 1. Write

$$X - 10^{-k} Y \cdot Q'_k = Q''_k (10^{-k} Y)$$

Let  $Y'_k = 10^{-k} Y$ . This quantity appears twice above. Let  $X'_k = X - Y'_k Q'_k$ . We have  $X'_k = Q''_k Y'_k$ . The algorithm is then: replace  $X'_k$  and  $Y'_k$  by their integer parts. Multiply  $X'_k$  by 10, find  $n$  the largest integer such that  $nY'_k \leq 10X'_k$ . This is the next digit in the quotient. Replace  $X'_k$  by  $10X'_k - nY'_k$ , divide  $Y'_k$  by 10. The procedure stops when  $Y'_k$  becomes zero. For instance, if we compute the inverse of the inverse of  $\pi$ , we find a number whose last digits are 243 instead of 238. On the other hand, if we divide 10 by  $10/\pi$ , we get all correct digits (this is because we lose a digit when we shift).

- `\FPmin\C\Va\Vb`. This puts in `\C` the smallest of the two values `\Va` and `\Vb`.
- `\FPmax\C\Va\Vb`. This puts in `\C` the largest of the two values `\Va` and `\Vb`.
- `\FPtruncate\C\V\I`. This truncates `\V` to `\I` digits and puts the result in `\C`.
- `\FPround\C\V\I`. This rounds `\V` to `\I` digits and puts the result in `\C`. For the two commands `\FPtruncate` and `\FPround`, the last argument has to be an integer. It is an error if this is a negative integer. Nothing happens if the integer is larger than 18. Truncating  $\pi$  to 4 digits gives 3.1415, rounding gives 3.1416. Rounding 1.25 to 1 digit gives 1.3. Rounding  $-X$  gives the opposite of rounding  $X$ . No overflow is signaled: rounding  $10^{18} - \epsilon$  may give  $10^{18}$ .
- `\FPiflt\Va\Vb`, `\FPifgt\Va\Vb`, `\FPifeq\Va\Vb`. These three commands compare two values and set the boolean `\ifFPtest` to `\iftrue` in the case  $a < b$ ,  $a > b$  and  $a = b$  respectively, to `\iffalse` otherwise. This command is then executed. It is a conditional, so that a `\fi` is required, and an `\else` is allowed.
- `\FPifneg\V`, `\FPifpos\V`, `\FPifzero\V`, `\FPifint\V`. Four commands that test if the argument  $a$  satisfies  $a < 0$ , or  $a > 0$  or  $a = 0$  or  $a$  integer. The command `\ifFPtest` is set and executed as above.
- `\FPe\C`. This put the number  $e$  in `\C`.
- `\FPseed\V`. This is a counter that contains the seed used by the random number generator.
- `\FPrand\C`. This puts in the command a random number. The algorithm is that of Lewis, Goodman & Miller (1969). Let  $q = 127773$ ,  $m = 2147483647$ . Write  $m = Aq + B$ , with  $A = 16807$  and  $B = 2836$ . If  $s$  is the seed,  $a$  and  $b$  the quotient and remainder of  $s$  by  $q$ ,  $w = Ab - Ba$  if this is  $\geq 0$ ,  $w = Ab - Ba + m$  otherwise. Then  $w$  is the remainder of  $As$  by  $m$ . This is the new seed. The random number is  $w/m$ .
- `\FPpascal\C\I`. The last argument must be an integer, between 1 and 63. The result is one row of the Pascal triangle. For instance `\FPpascal\foo{5}` stores in `\foo` the token list `[1.,5.,10.,5.,1.]`.
- `\FPexp\C\V`. This puts in `\C` the exponential of the number `\V`. The command uses the relation

$$\exp(a + b) = \exp(a) \cdot \exp(b)$$

The argument  $x$  is written as  $x = a + b$ , where  $a$  is an integer, and  $b$  is between 0 and 1 in magnitude (same sign as  $x$ ). The quantity  $\exp(a)$  is precomputed (if  $a \geq 42$ , we have an overflow, if  $a \leq -43$ , we have an underflow, and the result is zero). The other factor is computed as

$$\exp(b) = \sum b^n/n! = \sum X_k$$

Here  $X_{k+1}$  is  $X_k$  multiplied by  $b$ , divided by  $k$ . The division by  $k$  uses a special algorithm (if the digits in base 1000 of the numbers are  $c_i$ , dividing  $\sum c_i B^i$  by a number  $k$  is trivial, provided  $k < 1000$ ). We compute terms as long as  $X_k$  is no zero.

- `\FPln\C\V`. This puts in `\C` the logarithm of the number `\V`. The command uses the relation

$$\log(a \cdot b) = \log(a) + \log(b)$$

Here we use for  $a$  a power of ten, so that our number is  $ab$ , with  $b$  between 1 and 10, then a power of two so that it is between 1 and 2. The logarithms of these quantities are precomputed. Let  $y = (x - 1)/(x + 1)$ . If  $1 \leq x \leq 2$  then  $0 \leq y \leq 1/3$ . Let

$$f(y) = 2 \sum y^{2k+1}/(2k+1) = 2 \sum X_k$$

We have  $f'(y) = 2 \sum y^{2k} = 2/(1 - y^2)$ . If we consider  $f(y)$  as a function of  $x$ , say  $g(x)$ , its derivative is  $1/x$ , so that  $g(x)$  is the logarithm of  $x$ . We compute  $X_k$  as  $Y_k/(2k + 1)$ , division as above,  $Y_{k+1} = zY_k$ , where  $z$  is  $y^2$ . This gives, for the number  $\pi$  with 18 digits a logarithm of 1.144729885849400161 (last digits should be 173). For the exponential, we get 23.140692632779268882 (last digits should be 8995). The error is larger, because we multiply two truncated numbers.

- `\FPpow\C\Va\Vb`. This puts in `\C` the first value raised to the power the second value.
- `\FProot\C\Va\Vb`. This puts in `\C` the first value raised to the power the inverse of the second value. This use

$$a^b = \exp(b \cdot \log a) \quad \sqrt[b]{a} = \exp(\log a/b)$$

If we compute `\FPpow\foo{4}{2}`, we get  $16 - 99 \cdot 10^{-18}$  and `\FProot\foo{16}{2}` gives  $4 - 51 \cdot 10^{-18}$ . Other examples: For some numbers, we have computed the square and fourth root. We have then squared the result (or squared twice) and computed the difference with the initial number.

$x$	70000	7000	700	70	7	0.7	0.07	0.007	0.0007
$10^{-18}(x - (\sqrt{x})^2)$	1629349	171653	17569	1871	157	14	2	1	1
$10^{-18}(x - (\sqrt[4]{x})^4)$	2515666	297486	27306	2222	247	7	2	1	1

- `\FPSin\C\V`, `\FPCos\C\V`, `\FPtan\C\V`, `\FPCot\C\V`, These functions compute the trigonometric functions of a value  $v$ . The tangent is the quotient of the sine and the cosine, the cotangent is the quotient with arguments reversed.
- `\FPSincos\Ca\Cb\V`, `\FPtancot\Ca\Cb\V`. The `\FPSincos` takes two commands: the sine will be put in the first, the cosine in the second. The command `\FPtancot` stores the tangent and the cotangent.

The first step is to reduce the argument modulo  $2\pi$ . The same algorithm as for the division is used (however  $2\pi$  is computed with 36 digits, so that the result is more precise than a division via FP commands). The second step is to reduce modulo  $\pi/2$ . This is done according to the following table, where  $S$  and  $C$  and the sine and cosine of  $y$ .

$x$	$y$	$\sin x$	$\cos x$
$0 \leq x \leq \pi/4$	$x$	$S$	$C$
$\pi/4 \leq x \leq \pi/2$	$\pi/2 - x$	$C$	$S$
$\pi/2 \leq x \leq 3\pi/4$	$x - \pi/2$	$C$	$-S$
$3\pi/4 \leq x \leq \pi$	$\pi/2 - x$	$S$	$-C$
$\pi \leq x \leq 5\pi/4$	$x - \pi$	$-S$	$-C$
$5\pi/4 \leq x \leq 3\pi/2$	$3\pi/2 - x$	$-C$	$-S$
$3\pi/2 \leq x \leq 7\pi/4$	$x - 3\pi/2$	$-C$	$S$
$7\pi/4 \leq x \leq 2\pi$	$2\pi - x$	$-S$	$C$

The quantities  $C$  and  $S$  are computed via the obvious formulas

$$\sin y = \sum (-1)^n y^{2n+1} / (2n + 1)! \quad \cos y = \sum (-1)^n y^{2n} / (2n)!$$

The following piece of code can be used to test:

```
\def\T#1{
\FPSin\fooA{#1}\FPCos\fooB{#1}
\FPmul\fooA\fooA\fooA
\FPmul\fooB\fooB\fooB
\FPadd\foo\fooA\fooB\FPadd\foo\foo{-1}
\show\foo}
```

If the argument is  $2x/10$ ,  $x$  integer, between 0 and 30, this procedure gives numbers with magnitude at most  $3 \cdot 10^{-18}$ .

- `\FParsin\C\V`, `\FParccos\C\V`, `\FParcsincos\Ca\Cb\V`. These commands compute the inverse sine, cosine, or both of an argument. The inverse sign is between  $-\pi/2$  and  $\pi/2$ , the inverse cosine is between 0 and  $\pi$ . It is an error if the argument is not between  $-1$  and  $+1$ . Consider

$$f(x) = \sum x^{2k+1} a_k / (2k+1) \quad a_k = a_{k-1} (2k-1) / 2k.$$

We have  $f'(x) = \sum a_k x^{2k}$ . Since  $a_k = (2k-1)!/k!2^k$ , we have  $f'(x) = (1-x^2)^{-1/2}$ , so that  $f$  is inverse sine. If  $n = 2k-1$ , we have to compute  $n/(n+1)$ , this uses a hack. If  $x \leq \sqrt{2}/2$ , we compute  $f(x)$ , otherwise  $f(y)$  with  $y = \sqrt{1-x^2}$ . This is the inverse cosine of  $x$ . The square root is computed exactly, it is not an approximation via logarithms and exponentials. When we compute, for  $x$  integer between 0 and 30, the inverse cosine of the cosine of  $x$ , we get an error at most  $24 \cdot 10^{-18}$ . The error reaches its maximum and changes sign near  $\pi/4$  and  $3\pi/4$ .

- `\FPtan\C\V`, `\FPcot\C\V`, `\FPtancot\Ca\Cb\V`. These commands compute the inverse tangent, or cotangent or both of the argument. For the inverse tangent the result is between  $-\pi/2$  and  $\pi/2$ . For the inverse cotangent the result is between 0 and  $\pi$ . Since  $\tan(\pi/2-t) = 1/\tan t$ , we can assume that the argument is less than one. In the case  $x \geq 8/9$ , we use the following:

$$y = \frac{5-4x^2}{9} \quad \frac{1}{\sqrt{1+x^2}} = \frac{2/3}{\sqrt{1-y}}.$$

If  $x = \tan t$ , then  $1/\sqrt{1+x^2} = \cos t$ . We compute this, then the inverse cosine. For  $1/\sqrt{1-y}$ , we use a variant of the formula shown above: We write  $(2k-1)/(2k) = 1 - 1/(2k)$ , and we compute  $1/(2k)$  via normal division. In the case  $x \leq 8/9$ , we use the formula

$$\arctan x = x - x^3/3 + x^5/5 - x^7/7 + \dots$$

- `\FPlsolve\Ca\Va\Vb`. Assume that the arguments are  $x_1$ , and  $A, B$ . This solves the equation

$$Ax + B = 0$$

and puts the result in  $x_1$ . In `Tralics`, you will find the number of solutions in `\count0`. In the case of an equation of degree one, the result is trivial:  $x = -B/A$ .

- `\FPqsolve\Ca\Cb\Va\Vb\Vc`. Assume that the arguments are  $x_1, x_2$ , and  $A, B, C$ . This solves the equation

$$Ax^2 + Bx + C = 0$$

and puts the result in  $x_1$ , and  $x_2$ . In `Tralics`, you will find the number of solutions in `\count0`. A special case is when  $A = 0$ , the equation is of degree one. Otherwise, we replace  $B$  by  $B/A$  and  $C$  by  $C/A$ . We replace  $B$  by  $B/2$ . Thus, the equation to be solved is  $x^2 + 2Bx + C = 0$ . Let  $\Delta = B^2 - C$ . If  $\Delta < 0$ , there is no solution. Otherwise, the solutions are  $x_2 = -(\pm\sqrt{\Delta} + B)$ , and  $x_1 = C/x_2$ . Here the sign of the square root is the same as  $B$ .

- `\FPcsolve\Ca\Cb\Cc\Va\Vb\Vc\Vd`. Assume that the arguments are  $x_1, x_2, x_3$ , and  $A, B, C, D$ . This solves the equation

$$Ax^3 + Bx^2 + Cx + D = 0$$

and puts the result in  $x_1, x_2$ , and  $x_3$ . In `Tralics`, you will find the number of solutions in `\count0`. A special case is when  $A = 0$ , this is an equation of degree 2. Otherwise, let  $D' = (D/A)/2$ ,  $T = -B/(3A)$ ,  $p = C/(3A) - T^2$  and  $q = -(BC)/(6A^2) + D - T^3$ . Let

$\delta = p^3 - q^2$ . If  $\delta > 0$ , there is a single solution. Let  $u = \sqrt{\delta} - q$  and  $v = -(\sqrt{\delta} + q)$ . The solution is

$$x = \sqrt[3]{u} + \sqrt[3]{v} + T.$$

Otherwise, let  $R = \pm\sqrt{|p|}$ , with the same sign as  $q$ . Let  $s = \arccos(q/R^3)/3$ , and

$$r_1 = -\cos(s) \quad r_2 = \cos(\pi/3 - s) \quad r_3 = \cos(\pi/3 + s).$$

The solutions are  $x_i = 2Rr_i + T$ . The implementation of the formulas are a bit different than in the initial T<sub>E</sub>X version, so that numerically, there can be some small differences.

- `\FPqqsolve\Ca\Cb\Cc\Cd \Va\Vb\Vc\Vd\Ve`. Assume that the arguments are  $x_1, x_2, x_3, x_4$  and  $A, B, C, D, E$ . This solves the equation

$$Ax^4 + Bx^3 + Cx^2 + Dx + E = 0$$

and puts the result in  $x_1, x_2, x_3$  and  $x_4$ . In Tralics, you will find the number of solutions in `\count0`. We can assume  $A \neq 0$ , otherwise this is an equation of degree 3. Then we divide by  $A$ , and assume  $A = 1$ . A special case is  $D = 0$ , i.e.  $Ay^2 + Cy + E = 0$ ,  $x^2 = y$ . We first find  $y$ . For each positive solution, we take the square root. A second special case is  $A = E$ ,  $B = D$ . This is

$$x^4 + Bx^3 + Cx^2 + Bx + 1 = 0.$$

If  $u = x + 1/x$ , we get  $u^2 + Bu + (C - 2) = 0$ . We solve this equation. For each solution  $u$  we solve  $x^2 - ux + 1 = 0$ . In the general case, we consider

$$B' = B/4, \quad T = -B', \quad E' = E - B'D + CB'^2 - 3B'^4, \quad D' = D - 2B'C' + 8B'^3, \quad C' = C - 6B'^2$$

The effect is to shift the roots by  $T$ . If

$$x^4 + C'x^2 + D'x + E' = 0$$

has roots  $r_i$ , then the initial equation has roots  $x_i = r_i + T$ . Consider

$$P = (x^2 + s'/2 + r - D/s')(x^2 - s'/2 + r + D/s').$$

This is the original polynomial if  $C = 2r - s'^2/4$  and  $E = r^2 - D^2/s'^2$ . Let  $s = 8r - 4C$ . The first equation is true if  $s'^2 = s$ . We must have  $s > 0$ . The second relation says that  $r$  must be a solution of

$$8y^3 - 4Cy^2 - 8Ey + 4CE - D^2 = 0.$$

This is called the resolvent. This, we solve this equation, compute the three roots, chose one with  $s > 0$  (if this is not possible the initial equation has no real root). After that  $P = 0$  becomes two equations of degree two.

## Alternate syntax

The command `\FPupn` implements a postfix language that allows you to write shorter code. Here is an example

```
\FPupn\foo{7 20 2 sub 100 2000 - add +}
\testeq\foo{1925.}
\FPupn\foo{20 2 div 100 2000 / add 3 mul 2 *}
\testeq\foo{180.}
\def\mthree{-3}%there is no unary minus in this language
\FPupn\foo{ 3 abs mthree abs 3 sgn 10 * mthree sgn 100 * + + +}
\testeq\foo{-84.}
```

```

\FPupn\foo{2 3 min 400 500 max +}
  \testeq\foo{502.}
\FPupn\foo{12.43745678 2 round 12.35745678 2 trunc -}
  \testeq\foo{-0.09}
\FPupn\foo{e 1.2 exp + 2.3 ln + 3 4 pow + 5 6 root +}
  \testeq\foo{72.302276955235951659}
\FPupn\foo{pi 0.7 - sin cos sincos - tan cot tancot +}
  \testeq\foo{-2.894412996263821897}
\FPupn\foo{0.3 arcsin 0.1 * arccos 0.1 * arcsincos -
  arctan arccot arctancot -}
  \testeq\foo{0.751779218345560029}
\FPupn\foo{3.4 seed random}
  \testeq\foo{0.000023479107778276}
\FPupn\foo{1.1 2.3 3.4 pop swap copy add sub}
  \testeq\foo{0.1}

```

The `\testeq` command can be used to test the code. It is an error if the two arguments are not the same. Some comments. Consider the last expression. We put 1.1, 2.3 and 3.4 on the stack. After that we pop an item. After that we swap. The stack holds 1.1 (top stack), followed by 2.3. Then we duplicate the top stack. Then we add. The topstack is now 2.2. After subtraction, we get 0.1. If you say ‘2 3 -’, the result is 1, because - and sub use arguments in a different order. The same is true for / and div. Note the order of 10 2 pow, this gives 1024. If strange words are seen, like ‘mthree’, they are replaced by `\mthree`. Note that ‘e’ and ‘pi’ are predefined.

If you don’t like postfix language, you can use `\FPeval`. Here are some examples.

```

\FPeval\foo{1000-100-10-1}
  \testeq\foo{889.}
\FPeval\foo{1000+100+10+1}
  \testeq\foo{1111.}
\FPeval\foo{1000-100+10+1}
  \testeq\foo{911.}
\FPeval\foo{1000+100-10+1}
  \testeq\foo{1091.}
\FPeval\foo{(20 - 2) + (2000-100) + 7}
  \testeq\foo{1925.}
\FPeval\foo{(20/2 + 2000/100)*3*2}
  \testeq\foo{180.}
\FPeval\foo{210/2/3/5}
  \testeq\foo{7.}
\FPeval\foo{210*2/3/5}
  \testeq\foo{28.}
\FPeval\foo{210/2*3/5}
  \testeq\foo{63.}
\FPeval\foo{210/2/3*5}
  \testeq\foo{175.}
\FPeval\foo{210*2*3/5}
  \testeq\foo{252.}
\FPeval\foo{210*2/3*5}
  \testeq\foo{700.}
\FPeval\foo{210/2*3*5}
  \testeq\foo{1575.}
\FPeval\foo{abs(3) + abs(-3) + (sgn(3)* 10) + (sgn(-3) * 100)}
  \testeq\foo{-84.}

```

```

\FPeval\xfoo{\min(2:3) + \max(400,500)}
  \testeq\xfoo{502.}
\FPeval\xfoo{\round(12.43745678,2) - \trunc(12.35745678, 2)}
  \testeq\xfoo{0.09}
\FPeval\xfoo{e + \exp(1.2) + \ln(2.3) + \pow(3, 4) + \root(5, 6)}
  \testeq\xfoo{72.302276955235951659}
\FPeval\xfooa{\sin(\cos(\sin(0.7 - \pi))) - \cos(\cos(\sin(0.7 - \pi)))}
\FPeval\xfoo{\tan (\cot(\tan(xfooa))) + \cot(\cot(\tan(xfooa)))}
  \testeq\xfoo{-2.894412996263821897}
\FPeval\xfooa{\arcsin (\arccos (\arcsin(0.3)*0.1)*0.1) -
              \arccos (\arccos (\arcsin(0.3)*0.1)*0.1)}
\FPeval\xfoo{\arctan(\arccot(\arctan(xfooa))) - \arccot(\arccot(\arctan(xfooa)))}
  \testeq\xfoo{0.751779218345560029}
\FPeval\xfoo{2+3*4+5*3^2}
  \testeq\xfoo{58.99999999999998665}
\FPeval\xfoo{3^2*5+4*3+2+1}
  \testeq\xfoo{59.99999999999998665}
\FPeval\xfoo{(+3+4)*(-5-6)}
  \testeq\xfoo{-77.}

```

If you wonder what happens, you can look the transcript file. You can see something like:

```

{\FPeval}
{FPpostfix 1 2 3 mul add 400 500 max sin 4 pow add}
{\FP@upn}
{FPupcmd ??}
{FPupcmd ??}
{FPupcmd ??}
{FPupcmd mul}
{FPread for \FP@upn=+3.}
{FPread for \FP@upn=+2.}
{FPupcmd add}
{FPread for \FP@upn=+6.}
{FPread for \FP@upn=+1.}
{FPupcmd ??}
{FPupcmd ??}
{FPupcmd max}
{FPread for \FP@upn=+500.}
{FPread for \FP@upn=+400.}
{FPupcmd sin}
{FPread for \FP@upn=+500.}
{FPupcmd ??}
{FPupcmd pow}
{FPread for \FP@upn=+4.}
{FPread for \FP@upn=-0.467771805322476126}
{FPupcmd add}
{FPread for \FP@upn=+0.522845423476396576}
{FPread for \FP@upn=+7.}
{FPread for \FP@upn=+7.522845423476396576}

```

The second line is the expression converted from infix to postfix. Each ‘??’ represents a string that does not start with a letter. This is generally a number.

## 5.9 Action before translation

Normally, translation applies only to what is between `\begin{document}` and `\end{document}`. This is a very special environment, in fact, it leaves the semantics stack pointer unchanged. There are two hooks. You can say

```
\AtBeginDocument{\foo}
\AtEndDocument{\xbar}
```

These commands remember the tokens in a special list, that is inserted in the input stream when `\begin{document}` or `\end{document}` is seen. After that, the meaning of the command changes: it becomes ‘evaluate now’, more precisely `\@firstofone`. The last action in the begin-document hook is to change the definition again, so that an error may be signaled, for instance *Can be used only in preamble: \AtBeginDocument*. On the other hand, the `\end{document}` command inserts a special marker that closes every open file, thus stopping translation at the end of the hook (the bibliography is translated after that). The command `\@onlypreamble` takes as argument a command name and adds it to the list of commands that become invalid after the preamble.

Before the begin of the document, you can use commands of the form

```
\documentclass[doc-opt]{doc-class}
\usepackage[pack-opt]{pack-name}
```

There are some differences with L<sup>A</sup>T<sub>E</sub>X, see next section. If ‘doc-opt’ contains ‘useallsizes’ this is the same as if a line in the configuration file has said to use all font sizes. If it contains ‘french’ or ‘english’, this defines the default language.

Before version 2.9, the name or options of the class could indicate the top-level section; for instance, book assumed ‘leadingpart’ and report assumed ‘leadingchapter’; these keywords are no more recognised. You have to say `\toplevelsection{\part}` in the class file if you want it to be ‘part’. The default is ‘part’ for a book, ‘chapter’ for a report, ‘section’ otherwise. If the top-level section is part, chapter, or section, the translation of `\subsection` is, respectively, a `<div3>`, `<div2>` or `<div1>` element. Moreover, an attribute pair `chapters=‘true’` or `part=‘true’` is added to the main element, so that a post-processor can decide that `<div1>` is subsection, section or chapter.

If the packages ‘calc’, ‘fp’ or ‘fancyhdr’ are loaded, then the meaning of some command changes, as explained elsewhere. If the ‘babel’ package is loaded, the following languages are recognized: english, american, british, canadian, UKenglish, USenglish (these have number 0), french, francais, frenchb, acadian, canadien (these have number 1) austrian, german, germanb, naustrian, ngerman (these have number 2). The first language in the list is the default language. If a package is named ‘french’ or ‘frenchle’ or ‘german’, the default language is also set. The default language can be used in the attribute list of the main document element. Setting the default language also set the current language (value of `\language`).

Note that the current version of the babel package accepts 63 options, which are language names, and if you specify an option not in the list, for instance ‘foo’, then `foo.ldf` is loaded if possible, so that the number of options could be greater. There are three other options that are recognised by certain languages and whose purpose is to make some characters active. We have shown in section 5.4 how Tralics handles double quotes in German. For instance, in spanish, if option ‘activeacute’ is given then ‘a’ is a shorthand for `\’a`, and this applies to 12 other characters. In catalan, you can also use option ‘activegrave’, this makes ‘a’ a shorthand for `\`a`, it applies only to A, E and O. Finally option ‘KeepShorthandsActive’ controls whether shorthands are activated by default. The ‘french’ package no longer exists, there are two versions ‘frenchpro’ (commercial) and ‘frenchle’ (free); there are two versions of the for German, ‘german’ and ‘ngerman’. All features of these packages can be found in babel (with possibly differences in the syntax). These packages have no options.



The ‘calc’ and ‘fancyhdr’ packages have no options. The ‘fp’ package has two options, ‘debug’ and ‘nomessages’ that are ignored by Tralics.

The standard configuration file contains lines like these:

```
on package loaded calc CALC = "true"
on package loaded foo/bar F001 = "true"
on package loaded *foo/bar F002 = "true"
on package loaded foo/*bar F003 = "true"
on package loaded *foo/*bar F004 = "true"
```

You can also say

```
on_package_loaded calc CALC = "true"
on_package_option calc CALC = "true"
on_class_option article CALC = "true"
on class option */* CALC = "+"
```

Before version 2.8, these lines of codes provoked some actions. They are now ignored. You should use classes and packages instead.

## 5.10 Classes and packages

We explain in this section how Tralics implements package and classes.

Assume that we have a file named myclass.ct, whose content is given here and will be explained later:

```
1 \ProvidesClass{mypackage}[2006/08/19 v1.0 myclass document class for Tralics]
2 \NeedsTeXFormat{LaTeX2e}[1995/12/01]
3 \DeclareOption{a}{\typeout{option A}}
4 \DeclareOption{b}{\typeout{option B}}
5 \DeclareOption{d}{\typeout{option D}}
6 \AtEndOfClass{\typeout{End of class}}
7 \typeout{Before execute options}
8 \ExecuteOptions{a}
9 \ProcessOptions\relax
10 \endinput
```

and a file named mypack.plt, containing this single line:

```
11 \ProvidesPackage{mypack}[2006/10/10 My package]
```

a file named mypack1.plt with

```
\ProvidesPackage{mypack1}[2006/10/10 My package]
\typeout{Loading file mypack1}
\DeclareOption{x}{}
\DeclareOption{y}{}
\DeclareOption{z}{}
\ProcessOptions \relax
\endinput
```

and finally a file named mypack2.plt

```
8 \ProvidesPackage{mypack2}[2006/10/10 My package]
9 \DeclareOption{e}{\typeout{Option E}}
10
11 \@ifpackageloaded{mypack}
12   {\typeout{Seen package mypack}}
13   {\typeout{Package mypack missing}}
```

```

14
15 \@ifpackagelater{mypack}{2006/11/11}
16   {\typeout{Seen good package mypack}}
17   {\typeout{Package mypack obsolete}}
18
19 \@ifpackagewith{mypack1}{x}
20   {\typeout{Seen mypack with x}}
21   {}
22 \@ifpackagewith{mypack1}{x,y}
23   {\typeout{Seen mypack with x and y}}
24   {}
25 \@ifpackagewith{mypack1}{x,y,z}
26   {\typeout{Seen mypack with x, y and z}}
27   {}
28
29 \ProcessOptions\relax
30 \endinput

```

Assume that we have a source document containing the following lines

```

31 \AtBeginDocument{\typeout{Begin Document}}
32 \documentclass[a,b,c,e]{myclass}[2007/03/05]
33 \typeout{In preamble}
34 \usepackage{mypack}
35 \usepackage[y,x,w]{mypack1}
36 \usepackage{mypack2}[2000/00/00]
37 \usepackage[y,x,w]{mypack1}
38 \usepackage[aa,bb]{mypack1}
39
40 \begin{document}
41 Text
42 \end{document}

```

When Tralics translates the document above, you will see

```

43 This is tralics 2.11.7, a LaTeX to XML translator
44 Copyright INRIA/MIAOU/APICS 2002-2008, Jos'e Grimm
45 Licensed under the CeCILL Free Software Licensing Agreement
46 Starting translation of file toto.tex.
47 Warning: class myclass claims to be mypackage.
48 Document class: mypackage 2006/08/19 v1.0 myclass document class for Tralics
49 Before execute options
50 option A
51 option A
52 option B
53 Warning: You have requested, on line 3, version
54 '2007/03/05' of class myclass,
55 but only version
56 '2006/08/19 v1.0 myclass document class for Tralics' is available
57 End of class
58 In preamble
59 Loading file mypack1
60 Unknown option 'w' for package 'mypack1'
61 Seen package mypack
62 Package mypack obsolete

```

```

63 Seen mypack with x
64 Seen mypack with x and y
65 Option E
66 Option clash in \usepackage mypack1
67 Old options: y,x,w.
68 New options: aa,bb.
69 Tralics Warning: Unused global option:
70     c.
71 Begin Document

```

The `\documentclass` command has three arguments, an optional one, that defines the class options, a required one that defines the class name, and an optional one that indicates a date. Tralics reads 8 digits, with some separators, but L<sup>A</sup>T<sub>E</sub>X is a bit more exigent, four digits for the year, two digits for the month, two for the year, with slashes as separator, see above.

Evaluating the command is complicated. In fact, Tralics reads the file with extension `.clt` (instead of `.cls`), in either the current directory or the directory containing other configuration files, compares dates, and evaluates options. The behavior of `\usepackage` is similar. There are two differences. The first is that the mandatory argument of `\usepackage` can contain a comma-separated list of files; the second is that class options that are not used by the class can be used by packages. These options are called *global options*.

The `\usepackage` command must be used after `\documentclass`, before `\begin{document}`, there is a synonym `\RequirePackage` that can be used before the `documentclass` (this subtlety is not implemented in Tralics, both commands are always defined the same). There is also `\LoadClass`; this behaves like `\documentclass`, with some exceptions: for instance, the options of this command are not global options; there has to be a single `\documentclass` (L<sup>A</sup>T<sub>E</sub>X has additional requirements).

The two commands `\LoadClassWithOptions` and `\RequirePackageWithOptions` behave the same as the commands without the `WithOptions` but they take only two arguments: you give only a file name and maybe a date, you do not give options, because current options are used. Finally, `\InputClass` is a command defined by Tralics, that behaves like `\input`, but: the file (with extension `.clt`) is looked at in the same place as class files, and it can contain option declarations that apply to the current class (outside a class or package, you cannot declare options).

The class file should contain an identification line. This is like line 1 above, starting with the command `\ProvidesClass`. You can also use `\ProvidesPackage`, the behavior is the same. You can also use `\ProvidesFile`; in this case the identification line is printed on the transcript file, nothing more happens. In the case of the two other commands, the line is printed on the transcript file (in the case of a class, on the terminal as well, see transcript, line 52), and the date is parsed and remembered. The argument of the command should be the same as the file name, or else a warning is printed (line 51).

You can use the commands `\AtEndOfPackage` or `\AtEndOfClass`. These commands take an argument, whose content is added to the list of commands to be executed at the end of the class or package. In fact, when the end of file is seen, Tralics will insert and evaluate these tokens (example line 61); moreover a warning will be issued if there are options and the package does not process them (either via `\ProcessOptions` or via `\PassOptionToPackage`). Finally, a warning will be issued if the class or package is obsolete, i.e., earlier than the date argument of the `usepackage` or `documentclass` command (lines 57 to 60).

The commands `\@ifclassloaded` or `\@ifpackageloaded` take three arguments, P, A and B; they evaluate the token list A in case the class or package P is loaded, the token list B otherwise (example line 65). The commands `\@ifclasslater` or `\@ifpackagelater` take four arguments, P, D, A and B; they evaluate the token list A in case the class or package P is loaded with a date more recent than D, the token list B otherwise (example line 66). The commands `\@ifclasswith`

or `\@ifpackagewith` take four arguments, P, L, A and B; they evaluate the token list A in case the class or package P is loaded with options L, the token list B otherwise (example line 67, 68). The order of elements in L is irrelevant, the test is true if the package has been loaded with additional options.

The two commands `\PassOptionToClass` and `\PassOptionToPackage` take two arguments: an option list and a class name (or a package name), they add the options to the list of options of the package (this is useless if the class or package is not loaded later).

The command `\DeclareOption` takes two arguments A and B, where A is an option name and B a token list, the action associated to the option. If the option is processed, this list is evaluated. The command `\ExecuteOptions` takes a list of options as argument, and processes them in the given order (it processes only options defined in the current file). In the transcript given above, line 54 is the result of such an action. The command `\ProcessOptions` executes all options relevant to the current file. In the example, we have four class options, a, b, c, and e. The class defines a, b, and d, you see the result on lines 55 and 56. Nothing special happens for an option like d that is defined in the class but never referenced. Undefined package options generally provoke a warning (line 64). Unused class options (like c and e) become global options. In fact, option e is defined in 'mypack2', see transcript line 69. Thus, we have a single unused global option (see lines 73 and 74). Note the order of evaluation: if a star follows `\ProcessOptions`, the order is defined by the user (main document) otherwise by the system (class or package). The L<sup>A</sup>T<sub>E</sub>X documentation claims that reading this star may provoke expansion of commands that follows<sup>7</sup>, hence advises to use `\relax` in the case where no star is given.

If you say `\DeclareOption*{\foo}`, the `\foo` command is applied to every options not specified elsewhere. The name of the option will be in `\CurrentOption`. You can use `\OptionNotUsed` if you want to say that an option is not used. The command `\NeedsTeXFormat` takes a normal argument and an optional argument, but Tralics does nothing with them. The command `\default@ds` is an alias for `\OptionNotUsed`. You should not use it.

We give now an extract of the transcript file. Notice that, usually, the optional date argument of the `\usepackage` command is omitted, and Tralics reads the start of the next line in order to see if it is there. For instance, the first token on input line 6 is read in order to see if it is an open bracket (transcript line 118, 119). The command is evaluated later (transcript line 130). Note that Tralics restores a variable, `cur_file_pos`, this is the index of the current file in the list of packages or classes; it is zero outside a class or package. You can also see that the category code of the '@' character is set to 11 (letter) at the start of the file, and restored later.

Read carefully lines 99 to 102. In non-verbose mode, only line 101 is printed. This line says that options a, b, a, and b are executed. In fact, if the package defines options A and B, uses options a and b, and we have global options x, y, there is a first pass; we mark all options from the list A, B that are in the list a, b, x, y (see transcript, lines 99 and 100, or 145, 146); when an option is marked, its code is put in the todo list, and its code is removed. As a side effect, options are executed only once (on the other hand `\ExecuteOptions` leaves the option unchanged, so that the code may be executed more than once; once options are processed, executing them is a no-op). If you say `\ProcessOptions*`, the loop is on the global option list x, y, options are marked if they are in A, B. There is then a second pass on the list a, b. If the element is in the list A, B, it will be executed (in the case where is no star in the command, the option has already been used; otherwise it will be added to the to-do list; notice how this defines the order of evaluation of the options). If the element is not in the list, a fall back behaviour is used; if `\DeclareOption*{\foo}` has been issued, then then `\def \CurrentOption {a} \foo` will be added to the to-do list. Otherwise a warning is printed in the case of a package (see line 147), the option is added to the list of global options otherwise. The todo list is executed at the end of this second loop.

<sup>7</sup>This seems untrue, it might depend on the version of L<sup>A</sup>T<sub>E</sub>X, so that you should use an explicit star

```
72 [3] \documentclass[a,b,c,e]{myclass}[2007/03/05]
73 {\documentclass}
74 ++ file myclass.clt exists.
75 ++ Input stack ++ 1 myclass.clt
76 ++ Made @ a letter
77 ++ Opened file myclass.clt; it has 15 lines
78 [1]
79 [2] \ProvidesClass{mypackage}[2006/08/19 v1.0 myclass document class for Tralics]
80 Warning: class myclass claims to be mypackage.
81 Document class: mypackage 2006/08/19 v1.0 myclass document class for Tralics
82 [3] \NeedsTeXFormat{LaTeX2e}[1995/12/01]
83 [4]
84 [5] \DeclareOption{a}{\typeout{option A}}
85 [6] \DeclareOption{b}{\typeout{option B}}
86 [7] \DeclareOption{d}{\typeout{option D}}
87 [8] \AtEndOfClass{\typeout{End of class}}
88 [9] \typeout{Before execute options}
89 Before execute options
90 [10] \ExecuteOptions{a}
91 {Options to execute->a}
92 {Options code to execute->\typeout{option A}}
93 option A
94 [11] \ProcessOptions\relax
95 Marking option a
96 Marking option b
97 {Options to execute->a,b,a,b}
98 {Options code to execute->\typeout{option A}\typeout{option B}}
99 option A
100 option B
101 [12] \endinput
102 ++ End of file myclass.clt
103 ++ Catcode of @ restored to 12
104 Warning: You have requested, on line 3, version
105 '2007/03/05' of class myclass,
106 but only version
107 '2006/08/19 v1.0 myclass document class for Tralics' is available
108 ++ cur_file_pos restored to 0
109 ++ Input stack -- 1 myclass.clt
110 {\typeout}
111 End of class
112 [4] \typeout{In preamble}
113 In preamble
114 [5] \usepackage{mypack}
115 [6] \usepackage[y,x,w]{mypack1}
116 ++ file mypack.plt exists.
117 ++ Input stack ++ 1 mypack.plt
118 ++ Made @ a letter
119 ++ Opened file mypack.plt; it has 1 lines
120 [1] \ProvidesPackage{mypack}[2006/10/10 My package]
121 Package: mypack 2006/10/10 My package
122 ++ End of file mypack.plt
123 ++ Catcode of @ restored to 12
```

```

124 ++ cur_file_pos restored to 0
125 ++ Input stack -- 1 mypack.plt
126 {\usepackage}
127 [7] \usepackage{mypack2}[2000/00/00]
128 ++ file mypack1.plt exists.
129 ++ Input stack ++ 1 mypack1.plt
130 ....
131 {\usepackage}
132 ++ file mypack2.plt exists.
133 ++ Input stack ++ 1 mypack2.plt
134 ++ Made @ a letter
135 ++ Opened file mypack2.plt; it has 27 lines
136 ...
137 [9] \usepackage[aa,bb]{mypack1}
138 {\usepackage}
139 [10]
140 Option clash in \usepackage mypack1
141 Old options: y,x,w.
142 New options: aa,bb.
143 {\par}
144 [11] \begin{document}
145 {\begin}
146 {\begin document}
147 +stack: level + 2 for environment
148 {\document}
149 +stack: ending environment document; resuming document.
150 +stack: level - 2 for environment
151 +stack: level set to 1
152 ++ Input stack ++ 1 (AtBeginDocument hook)
153 [1] \let\do\noexpand\ignorespaces
154 ++ End of virtual file.
155 ++ cur_file_pos restored to 0
156 ++ Input stack -- 1 (AtBeginDocument hook)
157 LaTeX Warning: Unused global option(s):
158     c.
159 atbegindocumenthook= \typeout{Begin Document}\let\AtBeginDocument
160     \@notprerr\let\do\noexpand\ignorespaces

```

No error is signaled if the class or package does not exist. If you compile the example below with `Tralics`, no error is signaled, if you compile with `LATEX`, the following errors are signaled:

- on line 4, *Unknown option 'foo' for package 'calc'*;
- on line 5, *File 'xcalc.sty' not found*, and you are asked for an alternate file name, with default extension `.sty`;
- on line 5, *Unknown option 'a' for package 'xcalc'*;
- on line 5, *Unknown option 'b' for package 'xcalc'*;
- on line 8, *Undefined control sequence \xbad*;
- on line 9, *Undefined control sequence \ybad*.

```

1 \documentclass{article}
2
3 \usepackage[foo]{calc}
4 \usepackage[a,b]{xcalc}
5 \usepackage[a]{xcalc}
6 \makeatletter
7 \@ifpackagewith{calc}{foo}{}\{bad}
8 \@ifpackagelater{calc}{2005/12/12}{}\{xbad}
9 \@ifpackagelater{xcalc}{\ybad}{}
10 \@ifpackagelater{xcalc}{2000/01/02}{\zbad}{}
11 \@ifpackagewith{xcalc}{a}{}\{tbad}
12 \begin{document}
13 \end{document}

```

Note that `\usepackage` takes an optional argument, not given here, so that the first four errors are signaled after looking ahead for one token, they correspond to commands on line 3 and 4. In the case of Tralics, if a package is not found, the `\usepackage` declaration is ignored, this explains why `\ybad` is not called; no error is signaled for unknown options; however, the options are remembered, so that the second `\usepackage` will not try to load the file again, but checks the options. Builtin packages (`calc`, `fp`, `fancyhdr`, `babel`, `french`, `frenchle`, `german`) behave in a special way. If no ‘`plt`’ file is found, they are remembered in the table with 2006/01/01 as date; since the `calc` package is v4.1b, dated 1998/07/07, this explains why Tralics and L<sup>A</sup>T<sub>E</sub>X disagree when asked whether or not the package is older than december 2005. On the other hand, the default date of a file is 0000/00/00, so that an inexistant package is never later than a non-null date; the same is true for a package that does not provide a date, or before the identification of the package is evaluated; thus `\zbad` is not called.

The `xkeyval` package is an extension to `keyval`. It provides three extensions, that can be used in a package or a class:

```

\DeclareOptionX{opF}{\def\opF{#1}}
\DeclareOptionX{Cw}{}
\ExecuteOptionsX{keya,keyb=1}
\ProcessOptionsX \relax

```

The main document can start with

```

\documentclass[Cu,Cv,Cw,foo=E,opF={\foobar,gee},unused,Unused=U]{article}
\usepackage[opA,opB=C,epW=5,opC=\foo,opE]{testkeyval}

```

Let’s make the following assumptions: the `testkeyval` package defines all keys used in the `executeoptionsX` command, and most of the keys used in the `usepackage` declaration. You will see the following

```

xkeyval: Unknown option ‘epW=5’
xkeyval: Unknown option ‘opE’

```

We have shown above that the package defines `opF` and `Cw`; these are global class options. The package sees option `oF` with value, formed a a brace, a command, a comma, three characters, and a closing brace. You will see

```

Tralics Warning: Unused global options
unused,Unused=U.

```

This mechanism is neither fully compatible with pure L<sup>A</sup>T<sub>E</sub>X, nor with the `xkeyval` extension, but should work in all practical cases.

## 5.11 Expandable commands

We give here the list of all expandable commands, and some examples. Missing in this list are: some commands defined in the next section, all commands defined via `\def` in the C++ code, see section 6.13.

- `\a`, and all commands described in section 5.4. Here is a little dialog that shows expansion:
 

```

> \expandafter\def\expandafter\foo\expandafter{\a'e}
> \show\foo
\foo=macro: ->\'e.
> \edef\foo{\a'e}
> \show\foo
\foo=macro: ->é.
> \edef\foo{\aa}
> \show\foo
\foo=macro: ->â.

```
- `\aa` and some special characters. Commands whose effect is to produce a character in the XML result, on which an accent can be put, are expanded. In a near future, more commands could be implemented in this way.
- `\@firstofone`, `\@firstoftwo`, `\@secondoftwo`. These commands read one or two arguments, and return the first or the second.
 

```

> \makeatletter
> \expandafter\def\expandafter\foo\expandafter{\@firstofone{1}}
> \show\foo
\foo=macro: ->{1}.
> \expandafter\def\expandafter\foo\expandafter{\@firstoftwo{1}{2}}
> \show\foo
\foo=macro: ->{1}.
> \expandafter\def\expandafter\foo\expandafter{\@secondoftwo{1}{2}}
> \show\foo
\foo=macro: ->{2}.

```
- `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark`, `\splitbotmark`. The expansion is empty, because Tralics does not create pages.
- `\topmarks`, `\firstmarks`, `\botmarks`, `\splitfirstmarks`, `\splitbotmarks`. These are extensions introduced by  $\epsilon$ -TeX; an integer is read, expansion is empty.
- `\detokenize`, `\unexpanded`. These commands are introduced by  $\epsilon$ -TeX, and explained in section 6.12. These commands behave like `\the`, in that the resulting token list is not expanded, even in a `\edef` or `\write`.
- `\xspace`. This command reads a token. It may insert a space unless a) the token is a character of category code 12, and one of `. , ! ? : ; / ' ) -`, b) the token is an open brace, a close brace, or c) a space or `\`.
- `\[`, `\[ \`, `\)`. Expansion is one or two dollar tokens.
- `\noexpand`, `\expandafter`, `\csname`, `\the`: The behaviour is the same as that of TeX.
- `\number`, `\romannumeral`, `\string`, `\meaning`, `\fontname`, `\jobname`. The expansion of these macros is a list of characters, implemented in Tralics the same as in TeX.



- `\tralicsversion`, `\eTeXrevision`, `\Romannumeral`, `\@arabic`, Extensions to the list above, the first two commands return a character string indicating the current version of Tralics, or the subversion of  $\epsilon$ -T<sub>E</sub>X (note that the version number is a read-only integer), `\Romannumeral` produces uppercase roman digits, `\@arabic` is the same as in L<sup>A</sup>T<sub>E</sub>X: the following code gives 5V.

```
\makeatletter
\count0=3
\count4=5
\@arabic{\count0}4
\Romannumeral\count4
```

- `\if`, `\ifcat`, `\ifnum`, `\ifdim`, `\ifodd`, `\ifvmode`, `\ifhmode`, `\ifmmode`, `\ifinner`, `\ifvoid`, `\ifhbox`, `\ifvbox`, `\ifx`, `\ifeof`, `\iftrue`, `\iffalse`, `\ifcase`: conditionals.
- `\fi`, `\or`, `\else`: More conditionals.
- `\ifdefined`, `\iffontchar`, `\ifcsname`:  $\epsilon$ -T<sub>E</sub>X additional conditionals.
- `\unless`. The next unexpanded token must be a boolean conditional (not `\ifcase`); the truth value of that conditional is reversed.
- `\@afterfi`, `\@afterelsefi`: These commands can be used in a if-then-else structure, they terminate the condition and re-insert the interesting tokens.
- `\input`, `\endinput`. The expansion of these macros is empty, but they change the list of tokens to be considered later.
- `\useverb` is an extension of the verbatim package.
- `\@car`, `\@cdr`: these give access to first or rest of a list terminated by `\@nil`.

```
> \edef\foo{\expandafter\@car\@cdr123\@nil\@nil}
> \show\foo
\foo=macro: ->2.
```

- `\zap@space` removes spaces in a list (see example below for how the list is terminated).
- `\strip@prefix` reads all characters up to a greater than sign, see example below.
- `\hexnumber@` reads a number, assumed to be between 0 and 15, and returns a base 16 digit.

```
> \edef\foo{\zap@space 12 34 56 \@empty}
> \show\foo
\foo=macro: ->123456.
\let\E\expandafter
> \E\def\E\xfoo\E{\strip@prefix ab>cd}
> \show\xfoo
\xfoo=macro: ->cd.
> \E\E\E\def\E\E\E\xfoo\E\E\E{\E\strip@prefix\meaning \foo}
> \show\xfoo
\xfoo=macro: ->123456.
> \edef\foo{\hexnumber@\numexpr 2*6\relax}
> \show\foo
\foo=macro: ->C.
```

- `\@ifempty`, `\@ifbempty`. These commands, provided by Tralics, take three arguments, A, B and C. They expand to B if A is empty, to C otherwise. For the second command, blank spaces are removed. The command `\@zifempty` defined below is the easy way to test that the argument is empty; if the argument is a token list that starts with an at-sign (of category 11), the test is true, and part of the argument is evaluated. The command `\@xifempty` is a bit more complicated: it uses an additional command, so that an argument consisting of spaces only is passed as empty, thus the behavior is the same as `\@ifbempty`. This command comes from the class `amsart`. In order to use these commands with a normal at-sign, we changed the name.

```

\makeatletter
\let\@xp=\expandafter
\long\def\@zifempty#1{\ifx @#1\@xp\@firstoftwo\else\@xp\@secondoftwo\fi}
\long\def\@yifempty#1{\@xifempty#1@..\@nil}
\long\def\@xifempty#1#2@#3#4#5\@nil{%
  \ifx#3#4\@xp\@firstoftwo\else\@xp\@secondoftwo\fi}
\let\ifempty\@ifempty\let\yifempty\@yifempty\let\bifempty\@ifbempty
\let\zifempty\@zifempty
\makeatother

```

The following piece of code produces two identical lines.

```

\def\First#1#2{#1}
\def\Second#1#2{#2}
\def\w\ww{}
Ifempty \ifempty{ }{1}{2}\ifempty{ }{1}{2}\ifempty{\w}{1}{2}%
\yifempty{ }{1}{2}\yifempty{ }{1}{2}\yifempty{\w}{1}{2}%
\bifempty{ }{1}{2}\bifempty{ }{1}{2}\bifempty{\w}{1}{2}%
\zifempty{ }{1}{2}\zifempty{ }{1}{2}\zifempty{\w}{1}{2}%
\ifempty{\First}{\Second}{1}{2}%
\ifempty{ }{\First}{\Second}{1}{2}%
\ifempty{\w}{\First}{\Second}{1}{2}%
\yifempty{\First}{\Second}{1}{2}%
\yifempty{ }{\First}{\Second}{1}{2}%
\yifempty{\w}{\First}{\Second}{1}{2}%
\bifempty{\First}{\Second}{1}{2}%
\bifempty{ }{\First}{\Second}{1}{2}%
\bifempty{\w}{\First}{\Second}{1}{2}%
\zifempty{\First}{\Second}{1}{2}%
\zifempty{ }{\First}{\Second}{1}{2}%
\zifempty{\w}{\First}{\Second}{1}{2} %
\yifempty{x@22}{4}{2}%
\yifempty{xy@22}{4}{2}%
\yifempty{xy@22@33}{4}{2}%
.
\if\ifempty{ }{1}{2}1\else\error\fi
\if\ifempty{\w}{1}{2}2\else\error\fi
\if\yifempty{ }{1}{2}1\else\error\fi
\if\yifempty{\w}{1}{2}2\else\error\fi
\if\bifempty{ }{1}{2}1\else\error\fi
\if\bifempty{\w}{1}{2}2\else\error\fi
\if\bifempty{ }{1}{2}1\else\error\fi
\if\yifempty{ }{1}{2}1\else\error\fi
\count\bifempty{\w}{1}{2}=1

```

```

\count\ifempty{\w}{1}{2}=1
\count\yifempty{\w}{1}{2}=1
\count\zifempty{\w}{1}{2}=1
Ifempty 122112112122122112112122 222.

```

## 5.12 Other expandable commands

Not all commands defined here can be expanded, see the list at the end of Chapter 2. Essentially, this describes the `ifthen` package, the `calc` package, the `newtheorem` mechanism, and some input-output commands.

- `\@iwhilenum`, `\@iwhiledim`, and `\@iwhilesw` are internal commands for implementing the next three commands; should not be used.
- `\@whilenum`, `\@whiledim`, `\@whilesw`, `\whiledo`. Consider the following example,

```

\newcounter{ca}\newcounter{cb}
\newcommand{\printgcd}[2]{%
  \setcounter{ca}{#1}\setcounter{cb}{#2}%
  Gcd(#1,\,#2) =
  \whiledo{\not\(\value{ca}=\value{cb}\)}%
    {\ifthenelse{\value{ca}>\value{cb}}%
      {\addtocounter{ca}{-\value{cb}}}%
      {\addtocounter{cb}{-\value{ca}}}%
      gcd(\arabic{ca},\,\arabic{cb}) = }%
  \arabic{ca}.}

```

This works only if the package `ifthen` is loaded. It should produce (we have inserted some `\allowbreak` commands in order to allow linebreaks in the very big formula):  $\text{Gcd}(144, 272) = \text{gcd}(144, 128) = \text{gcd}(16, 128) = \text{gcd}(16, 112) = \text{gcd}(16, 96) = \text{gcd}(16, 80) = \text{gcd}(16, 64) = \text{gcd}(16, 48) = \text{gcd}(16, 32) = \text{gcd}(16, 16) = 16$ .

The `ifthen` package implements `\whiledo` in terms of the L<sup>A</sup>T<sub>E</sub>X command `\@whilesw`. The Tralics implementation of `\whiledo` is: read arguments A and B. Evaluate A as the first argument of `\ifthenelse`. If false terminate. Otherwise, the expansion is `B\whiledo A{B}`. Using one version or the other produces the same result.

```

\makeatletter
\newif\if@whiledo
\long\def\whiledo#1#2{%
  \ifthenelse{#1}%
    {\@whiledotrue
     \@whilesw\if@whiledo\fi
     #2%
     \ifthenelse{#1}\@whiledotrue\@whiledofalse}}%
  {}%
}

```

This is the same code, without the `ifthen` package.

```

\newif\iftest
\newcommand{\printgcd}[2]{%
  \setcounter{ca}{#1}\setcounter{cb}{#2}%
  \testtrue
  \@whilesw\iftest gcd(\arabic{ca},\arabic{cb}) = \fi{%

```

```

\ifnum\value{ca}=\value{cb}\testfalse\else
\ifnum\value{ca}>\value{cb}%
\addtocounter{ca}{-\value{cb}}\else
\addtocounter{cb}{-\value{ca}}\fi
\fi}%
\arabic{ca}.}

```

We propose now a method that uses `\@whilenum`. We start with two commands that put in  $(c, d)$  the numbers  $(a, b)$ , in some order, starting with the smallest. We consider also a function that takes the integers, puts them in length registers.

```

\newlength\cC\newlength\cD
\newcounter{cc}\newcounter{cd}
\def\assigncounter{%
\ifnum\value{ca}<\value{cb}%
\setcounter{cc}{\value{ca}}%
\setcounter{cd}{\value{cb}}%
\else
\setcounter{cc}{\value{cb}}%
\setcounter{cd}{\value{ca}}%
\fi}
\def\assignlength{%
\ifnum\value{ca}<\value{cb}%
\setlength\cC{\value{ca}sp}%
\setlength\cD{\value{cb}sp}%
\else
\setlength\cC{\value{cb}sp}%
\setlength\cD{\value{ca}sp}%
\fi}

```

This puts  $(c, c - d)$  in  $(a, b)$ .

```

\def\subtractcounter{%
\setcounter{cb}{\value{cd}}%
\setcounter{ca}{\value{cc}}%
\addtocounter{cb}{-\value{cc}}}%
\def\subtractlength{%
\setcounter{cb}{\cD}%
\setcounter{ca}{\cC}%
\addtocounter{cb}{-\cC}}%

```

Computing the gcd is now obvious: it suffices to call `assign` and `subtract`, until  $c = d$  (this will happen), because the procedures leave the gcd invariant.

```

\let\printgcd\relax
\newcommand{\printgcd}[2]{%
\setcounter{ca}{#1}\setcounter{cb}{#2}%
\assigncounter
\@whilenum \value{cc}<\value{cd} Gcd(\arabic{cc},\arabic{cd}) = \do{%
\subtractcounter\assigncounter}%
\arabic{ca}.}
\printgcd{144}{272}

```

The same with dimensions

```

\let\printgcd\relax
\newcommand{\printgcd}[2]{%

```

```

\setcounter{ca}{#1}\setcounter{cb}{#2}%
\assignlength
\@whiledim \cC<\cD Xgcd(\arabic{ca},\arabic{cb}) = \do{%
  \subtractlength\assignlength}%
\arabic{ca}.}
\printgcd{144}{272}

```

In verbose mode, the transcript file of Tralics will contain lines of the form shown here. It indicates the expansion of \@whilenum and the internal command \@iwhilenum associated to it.

```

\@whilenum<- \ifnum \value {cc}<\value {cd} Gcd(\arabic {cc},\arabic {cd}) =
\relax \subtractcounter \assigncounter \relax \@iwhilenum {\value {cc}<\value
{cd} Gcd(\arabic {cc},\arabic {cd}) = \relax \subtractcounter \assigncounter
\relax }\fi

```

```

\@iwhilenum<- \ifnum \value {cc}<\value {cd} Gcd(\arabic {cc},\arabic {cd}) =
\relax \subtractcounter \assigncounter \relax \expandafter \@iwhilenum \else
\expandafter \@gobble \fi {\value {cc}<\value {cd} Gcd(\arabic {cc},\arabic
{cd}) = \relax \subtractcounter \assigncounter \relax }

```

This looks like the following (in the case of \@whiledim, there is no \relax after ‘ETC’, so that the code is compatible with L<sup>A</sup>T<sub>E</sub>X).

```

\@whilenum<- \ifnum TEST ACTION \relax ETC \relax
\@iwhilenum {TEST ACTION \relax ETC \relax }\fi

\@iwhilenum<- \ifnum TEST ACTION \relax ETC \relax
\expandafter \@iwhilenum \else \expandafter \@gobble \fi
{TEST ACTION \relax ETC \relax }

```

We can simplify this further to the following.

```

\@whilenum<- \ifnum CODE \@iwhilenum {CODE}\fi
\@iwhilenum<- \ifnum CODE
\expandafter \@iwhilenum \else \expandafter \@gobble \fi {CODE}

```

You can see that, if the test is true, the first \expandafter token is evaluated; as a consequence, everything between \else and \fi is removed and we are left with \@iwhilenum followed by {CODE}; if the test is false the \fi is expanded first, so that \@gobble sees the code to gobble.

- \loop. This is the command described in the T<sub>E</sub>X book. The implementation is less tricky. The following example should give 0291817161514131211101908070605040302010

```

\count0=0
\def\foo{}
\def\xbar#1#2{\xdef\foo{#2#1\foo}}
\loop \advance\count0by1 \edef\xx{\ifnum\count0<10 0\fi\the\count0 }%
\expandafter\xbar\xx \ifnum\count0<20 \repeat
\foo

```

- \setlength. This command takes two argument, say \foo and ‘bar’. The expansion is \foo bar\relax, unless the calc package is loaded, case where the result is \calc{\foo}{bar}.
- \addtolength. This command takes two argument, say \foo and ‘bar’. The expansion is \advance\foo bar\relax, unless the calc package is loaded, case where the result is \calc{\advance\foo}{bar}. Tralics tests that the first argument is a single token; an error is signaled if this is \skip\footins, although this is a valid length.

- `\setcounter`. This command takes two arguments, say ‘foo’ and ‘bar’. The expansion is `\global\c@foo bar\relax`, unless the `calc` package is loaded, case where the result is `\calc{\global\c@foo}{bar}`.
- `\addtocounter`. This command takes two arguments, say ‘foo’ and ‘bar’. The expansion is `\global\advance\c@foo bar\relax`, unless the `calc` package is loaded, case where the result is `\calc{\global\advance\c@foo}{bar}`. For these last two commands, Tralics tests that the first argument is a counter. For instance, if the first argument is `\bar`, you get *Invalid token \bar found in counter name*.
- `\widthof`, `\heightof`, `\depthof`: These commands take an argument, typeset it in a box, and return the dimension of the box. Not implemented in Tralics.
- `\ratio`. This command is defined by the `calc` package to be used as `x/\ratio{u}{v}` or `x*\ratio{u}{v}`. Arguments  $u$  and  $v$  are dimensions, interpreted by the `\calc` function. The expression that comes before the operator is divided by  $u$  and multiplied by  $v$  (or the contrary). In Tralics, the quotient is computed with 6 digits; this is not exactly the same algorithm as in L<sup>A</sup>T<sub>E</sub>X.
- `\real`. This command is defined by the `calc` package to be used as `x*\real{u}` or `x/\real{u}`. The argument  $u$  should be a real number as in ‘1.5’.
- `\calc`. Examples of the `calc` package are given in section 2.7. The rules are the following: the `\calc` command (a private command), takes two arguments; the first argument is a token list of the form `\global\advance\count0`. Said otherwise, a variable, preceded by optional prefixes. The variable must be something that reads an integer, a dimension, or glue. The second argument is evaluated and put in the variable (or added to the variable); assignment can be global. A primitive is either a parenthesized expression, a call to `\widthof`, or the result of the command `scanint`, `scandim` or `scanglue` (whichever is used depends on the type of the variable). As a result, if a command like `\setcounter{foo}{bar}` is valid without the `calc`, it remains valid with it, and the interpretation is the same. Moreover, expressions like  $a + b$ ,  $a - b$ ,  $a/b$  and  $a * b$  can be used, instead of `\advance`, `\multiply` or `\divide`. In the case of multiplication or division, the second argument must be an integer; as an extension, it can be `\real` or `\ratio`, as explained above, in this last case, the Tralics result may differ slightly from the L<sup>A</sup>T<sub>E</sub>X value.
- `\newboolean`, `\provideboolean`. These two commands take as argument a character string, for instance ‘foo’ and behave like `\newif\iffoo`. Nothing happens for `\provideboolean` if the command `\iffoo` exists, but an error is signaled by `\newboolean`.
- `\setboolean`. This command takes two arguments. The second argument should be, after translation, and modulo the case, ‘true’ or ‘false’. If the first argument is ‘foo’, then `\footrue` is executed, otherwise `\foofalse` is executed. This works well if the command has been defined by `\newboolean{foo}`.
- `\boolean`. This can be used only in an ‘ifthenelse’ test. It takes an argument, say ‘foo’. The test is true if `\iffoo` is `\iftrue`, false otherwise. This works as expected if `\setboolean` has been used to set the value of ‘foo’.
- `\equal`. This can be used only in an ‘ifthenelse’ test. It takes two arguments, puts them in commands, say `\tmpa`, and `\tmpb`, expands then via `\edef`, and compares the result via `\ifx`. The result can be true or false.
- `\isodd`. This can be used only in an ‘ifthenelse’ test. It takes an argument, say ‘foo’. The test is true if the argument is an odd number as seen by `\ifodd`. There are no such hacks as in the L<sup>A</sup>T<sub>E</sub>X version.

- `\isundefined`. This can be used only in an ‘ifthenelse’ test. It takes an argument, say `\foo`. The test is true if it is undefined (if the argument contains more than one token, only the first one is considered).
- `\lengthtest`. This can be used only in an ‘ifthenelse’ test. It takes an argument, say ‘1cm=2cm’. The test is true if `\ifdim` interprets this as true.
- `\not`, `\and`, `\or`, `\NOT`, `\AND`, `\OR`. These can be used only in an ‘ifthenelse’ test as boolean connectors (but could be used elsewhere with a different meaning). If you say `\not \foo \and \not \not \bar`, the condition is true if `\foo` is false and `\bar` is true. Note that if `\foo` is found true, the condition is false, whatever follows `\and`, and this is not evaluated. In the same fashion, if what precedes `\or` is true, what follows is not evaluated.
- `\(, \)`. This can be used to change the order of evaluation. For instance `\not \( \foo \and \bar\)` negates foo-and-bar.
- `\ifthenelse`. This takes three arguments. The first is evaluated using the command explained above. If true, the expansion is the second argument, otherwise the third. In a case like `\ifthenelse{1=3}{Y}{N}`, the argument is evaluated as in `\ifnum`.
- `\theorembodyfont`. This is a command that remembers its argument for use with theorems. Default value is empty.
- `\theoremstyle`. This is a command that remembers its argument for use with theorems. Default value is ‘plain’.
- `\theoremheaderfont`. This is a command that remembers its argument for use with theorems. Default value is `\bfseries`.
- `\newtheorem`. The implementation is as described in the first edition of [2]. An example can be found on page 256. Implementation uses the reserved commands `\@begintheorem`, `\@endtheorem`, `\@xbegintheorem`. We comment here a part of the trace:

```
1 [8] \theorembodyfont{\sl}
2 [9] \theoremstyle{break}
```

This is our first theorem. It defines an environment `Cor`. For some reason, `\theCor` is set to `\relax`.

```
3 [10] \newtheorem{Cor}{Corollary} \relax
4 {\newtheorem}
5 {\newtheorem Cor}
6 {\let \endCor \@endtheorem}
7 {\let \theCor \relax}
8 {\def \Cor ->\@begintheorem {Corollary}\theCor {\sl }{Cor}{Cor}{break}}
```

Here you see that a counter `Cor` is defined.

```
9 {\countdef \c@Cor=\count24}
10 {\def \theCor ->\arabic {Cor}}
```

Here we see that an optional argument can be used after the name. The counter `Exa` depends on the counter `section`, and `\theExa` uses `\thesection`.

```
11 [14] \newtheorem{Exa}{Example}[section]
12 {\newtheorem}
13 {\newtheorem Exa}
14 {\let \endExa \@endtheorem}
15 {\let \theExa \relax}
16 {\def \Exa ->\@begintheorem {Example}\theExa {\sl }{Exa}{Exa}{plain}}
17 {\countdef \c@Exa=\count25}
```

```

18 {newcounter_opt}
19 {newcounter_opt->\cl@section}
20 \cl@section ->\@elt {subsection}
21 {\def \theExa ->\thesection .\arabic {Exa}}

```

Here we have an optional argument before the name. There is no counter Lem, but `\theLem` is `\theCor`.

```

22 [17] \newtheorem{Lem}[Cor]{lemma}
23 {\newtheorem}
24 {\newtheorem Lem}
25 {\let \endLem \@endtheorem}
26 {\let \theLem \relax}
27 {\def \Lem ->\@begintheorem {lemma}\theLem {\sl }{Lem}{Cor}{marginbreak}}
28 {\def \theLem ->\theCor }

```

This is now the use of one theorem. We have a command that reads 6 arguments, plus the text.

```

29 [23] \begin{Cor}
30 {\begin}
31 {\begin Cor}
32 +stack: level + 2 for environment
33 \Cor ->\@begintheorem {Corollary}\theCor {\sl }{Cor}{Cor}{break}
34 {\@begintheorem}
35 [24] This is a sentence typeset in the theorem environment \Lenv{Cor}.
36 {Push theorem 1}

```

We first increment the counter (argument 5).

```

37 \refstepcounter->\global \advance \c@Cor 1\relax {\let \@elt \@stpelt \cl@Cor }
38 {\global}
39 {\global\advance}
40 +scanint for \c@Cor->1
41 ...

```

This produces ‘Corollary 1’ from arguments 1 and 2. A part of the transcript file has been removed, but you get the idea. The `\scshape` comes from `\theoremheaderfont`.

```

42 +stack: level + 3 for brace
43 {\scshape}
44 {\font change \scshape}
45 {\begin-group character {}}
46 \theCor ->\arabic {Cor}
47 \arabic->\number \c@Cor
48 +scanint for \number->1
49 {Text:Corollary 1 }
50 {\font restore }
51 +stack: level - 3 for brace

```

What follows is easy: the body is typeset using argument 3.

```

52 {\sl}
53 {\font change \slshape}
54 Character sequence: This is a sentence typeset in the theorem environment .
55 ...

```

If the configuration file defines `xml_theorem_name`, the translation is not the same, because the meaning of `\@begintheorem` has changed.

```

56 [23] \begin{Cor}
57 {\begin}
58 {\begin Cor}
59 +stack: level + 2 for environment

```



```

60 \Cor ->\@begintheorem {Corollary}\theCor {\sl }{Cor}{Cor}{break}
61 {\@begintheorem}
62 [24] This is a sentence typeset in the theorem environment \Lenv{Cor}.
63 {Push theorem 1}
    Arguments 1, 4 and 6 are grabbed, and put in an element or attributes. Note: arguments 5
    and 6 are identical.
64 {Push Head 2}
65 Character sequence: Corollary.
66 {Text:Corollary}
67 {Pop 2: document_v theorem_v Head_t}
68 {Push argument 2}
69 Character sequence: Cor.
70 {Text:Cor}
71 {Pop 2: document_v theorem_v argument_t}
72 {Push argument 2}
73 Character sequence: break.
74 {Text:break}
75 {Pop 2: document_v theorem_v argument_t}
    The counter is always incremented.
76 \refstepcounter->\global \advance \c@Cor 1\relax {\let \@elt \@stpelt \cl@Cor }
    End is as usual.
77 {Push p 2}
78 Character sequence: This is a sentence typeset in the theorem environment .

```

- `\begin{filecontents}{name} ... \end{filecontents}`.. The content of the environment is printed verbatim on the file name, unless the file exists. The file starts with a header, of the form

```

%% LaTeX2e file 'openinaux.tex' utf8-encoded
%% generated by the 'filecontents' environment
%% from source 'txt9' on 2006/09/06.

```

In the example that follows, we use the starred version, in which case the header is omitted. This gives a file with 6 lines and 30 characters. We shall use it in the `\read` example below, where it is important that no header be added. You can notice that the first line says that the file is UTF-8 encoded; this may be necessary in the case where the non-ascii characters are copied, because the encoding cannot be changed.

```

1 \begin{filecontents*}{openinaux.tex}
2 abc
3
4 \a \b {\c
5 } \d} \e
6 123
7
8 \end{filecontents*}

```

The name of the environment can also be `filecontents+`. In the case the header is omitted as before; moreover the file is only created virtually; this means that no file is created on the disk; moreover, if Tralics reads a file it first checks the list of virtual files, before scanning the disk. The philosophy is the following: the environment is a sort of provide-package: this allows me to distribute a T<sub>E</sub>X file that uses an experimental package, included in the source. This explains why the environment can be used only in the preamble of the document. On the other hand, some Tralics test files use some dummy packages (that provide no interesting feature), provided by this environment, and it is important that the new package be used.

- `\openin`. This command reads a small integer, an optional equals sign, and a file name, this is a sequence of characters not containing a space. It reads the content of the file. If the filename is 'foo', and no file named foo exists, then `foo.tex` is tried. You can read the content via `\read`. If the file is not found, empty, or completely read, it is marked as 'closed'.
- `\closein`. This command reads an integer, and closes the channel associated to the number.
- `\ifeof`. This command reads an integer, and checks if the channel is opened for reading.
- `\read`. This command reads an integer, the keyword 'to' and a command name. After that, it reads a line (more than one line can be read, because the result is always balanced according to braces), and puts in the command. The line is read from the file associated to the number, unless it is closed, case where it is read from the terminal. We give here an example:

```

{
  \openin 5=openinaux
  \endlinechar=-1
  \ifeof5 \badifeofatentry\fi
  \read 5 to \foo\show\foo
  \read 5 to \foo\show\foo
  \read 5 to \foo\show\foo
  \global\read 5 to \foo
  \closein5\relax
  \ifeof5\else\badifeofatexit\fi
  }\show\foo
\ifeof3\else \badifeofnonexists\fi

```

The transcript file contains the following lines. One purpose of the group opened here is to keep the modification to the endline character local.

```

1 [15] {
2 {\begin-group character {}}
3 +stack: level + 2 for brace

```

This shows the `\openin` command. You can see that the file was `openinaux.tex`, it contains six lines. Even in non-verbose mode, the transcript file contains a line *file foo (does/does not) exist* whenever a file is tested for existence.

```

4 [16] \openin 5=openinaux
5 {\openin}
6 +scanint for \openin->5
7 ++ file openinaux does not exist
8 ++ file openinaux.tex exists
9 ++ Opened file openinaux.tex; it has 6 lines

```

The `\endlinecharacter` command specifies the value of the character that should be inserted at the end of every line read from a file. An out-of-range value means that no character should be inserted.

```

10 [17] \endlinechar=-1
11 {\endlinechar}
12 +scanint for \endlinechar->-1

```

This is a test of `\ifeof`; the test should be 'false' if the file has been read.

```

13 [18] \ifeof5 \badifeofatentry\fi
14 +\ifeof1
15 +scanint for \ifeof->5
16 +iftest1 false
17 +\fi1

```

The first line of the file contains ‘abc’. Thus, `\foo` should contain these letters, plus the endline character (there is no endline character here). Tralics maintains in a special stack positions in all active file, so that each `\read` pushes and pops an item on this stack. When a package or a class file is read, some information (the options) is stored in a table, at location  $N$ , for some positive number  $N$ , in all other cases the number is zero. You see here that this number is reset to zero.

```

18 [19] \read 5 to \foo\show\foo
19 {\read}
20 +scanint for \read->5
21 ++ Input stack ++ 1 openinaux.tex
22 ++ cur_file_pos restored to 0
23 ++ Input stack -- 1 openinaux.tex
24 {\show}
25 \foo=macro: ->abc.

```

The second line of the file is empty. If we did not change the value of `\endlinechar`, we would have seen `\par` here.

```

26 [20] \read 5 to \foo\show\foo
27 {\read}
28 +scanint for \read->5
29 ++ Input stack ++ 1 openinaux.tex
30 ++ cur_file_pos restored to 0
31 ++ Input stack -- 1 openinaux.tex
32 {\show}
33 \foo=macro: ->.

```

The third line contains `\a \b {\c`, this is incomplete, so that line 4, containing `{\d} \e` must also be read.

```

34 [21] \read 5 to \foo\show\foo
35 {\read}
36 +scanint for \read->5
37 ++ Input stack ++ 1 openinaux.tex
38 ++ cur_file_pos restored to 0
39 ++ Input stack -- 1 openinaux.tex
40 {\show}
41 \foo=macro: ->\a \b {\c {\d } \e .

```

This is to show that `\global` applies to `\read`: the command `\foo` is globally defined.

```

42 [22] \global\read 5 to \foo
43 {\global}
44 {\global\read}
45 +scanint for \read->5
46 ++ Input stack ++ 1 openinaux.tex
47 ++ cur_file_pos restored to 0
48 ++ Input stack -- 1 openinaux.tex

```

We close the file. Since the end-of-line character is inactive, it cannot be used as a terminator for the number 5, thus the `\relax`.

```

49 [23] \closein5\relax
50 {\closein}
51 +scanint for \closein->5
52 ++ End of file openinaux.tex
53 {\relax}

```

Here we check that the file is effectively closed. The `\relax` is automatically inserted because the `\else` is seen before the number 5 has been completely read (the `\else` is re-inserted).

```

54 [24] \ifeof5\else\badifeofatexit\fi
55 +\ifeof2
56 +\else2
57 +scanint for \ifeof->5
58 +iftest2 true
59 {\relax}
60 +\else2
61 +\fi2

```

Here you can see what happens at the end of the group: the end-line character is restored, but `\foo` is retained.

```

62 [25] }\show\foo
63 {end-group character }}
64 +stack: retaining \foo
65 +stack: restoring integer value 13 for \endlinechar
66 +stack: level - 2 for brace
67 {\show}
68 \foo=macro: ->123.

```

- `\openout`. This command reads a small integer (between 0 and 15), an optional equals sign, and a file name, this is a sequence of characters not containing a space. It opens the file for writing. In  $\TeX$ , this is an extension, and can be prefixed by `\immediate`; in *Tralics*, the action is always immediate.
- `\closeout`. This command reads a small integer, and closes the channel with this number. No error is signaled if the integer is not in the range 0-15. In  $\TeX$ , this is an extension, and can be prefixed by `\immediate`; in *Tralics*, the action is always immediate. If you write something on a file, and do not close it, it is unclear what is saved on the disk when *Tralics* quits.
- `\write`. This command reads an integer  $N$ , and a token list and writes the argument, after full expansion, to the stream. If  $N < 0$ , this is the transcript file; if  $N$  is a valid channel, to which a file is associated via `\openout`, printing will go there; otherwise to the terminal and transcript file. A special case is when the  $N = 18$  and the ‘shell-escape’ switch has been given: in this case, the command is evaluated by a shell. We use the same encoding as for the terminal and transcript file for non-ASCII characters.

Here is an example. Note how we inhibit expansion of commands. Note also that each `\write` command produces a single line; but newline characters can be added via `^^J`.

```

\immediate\openout 6=tout.tex
\immediate\write6{\def\noexpand\foo{\noexpand\endinput}}
\immediate\write6{\noexpand\foo^^J\noexpand\bar}
\immediate \closeout6 \relax
\input{tout}

```

This is part of the transcript file. It shows that the file has 3 lines, a first line that defines `\foo`, a line with `\foo` and `\bar`. Since `\foo` is `\endinput`, the line with `\bar` is not read.

```

1 [12] \input{tout}
2 {\input}
3 {Push argument 1}
4 Character sequence: tout.
5 {Text:tout}
6 {Pop 1: document_v argument_v}
7 {\input tout}
8 ++ Opened file tout.tex; it has 3 lines
9 [1] \def \foo {\endinput }

```

```

10  {\def}
11  {\def \foo ->\endinput }
12  [2] \foo
13  \foo ->\endinput
14  {\endinput}
15  ++ End of file tout.tex

```

- `\IfFileExists`. This command takes three arguments, say  $A$ ,  $B$ , and  $C$ . The quantity  $A$  must be a character string after translation. If there is a file with name  $A$ , the test is true. Otherwise, if there is a file with name ‘ $A.tex$ ’, the test is true. Otherwise, it is false. If the test is true, the expansion is ‘ $B$ ’, otherwise ‘ $C$ ’.
- `\InputIfFileExists`. This command takes three arguments, say  $A$ ,  $B$ ,  $C$ . The behavior is like `\IfFileExists {A} {B\Input{A}} {C}`. The procedure is a bit optimised: first all arguments are read; then the existence of the file is checked. If the file does exist, it is opened; this has as consequence that all unread characters on the current line are saved on a special stack, they will be read again at the end of the current file. Then one of the token lists  $B$  or  $C$  is inserted into the list of unread tokens.  
You can put a star after the name of the command. This will have as a consequence that the character ‘@’ is a letter while reading the content of the file. Its old category code will be restored.
- `\include`. This command takes one argument, otherwise behaves like `\input`.
- `\input`. This command reads a filename as in `\openin` or `\openout`. It is redefined by L<sup>A</sup>T<sub>E</sub>X to accept an argument. ‘`\input foo\input{bar}`’ is valid. In the first case, tokens are read and expanded until either a space, a non-expandable token, or `\input` is found.
- `\Input`. Like the L<sup>A</sup>T<sub>E</sub>X version of `\input`, but an optional star can be used after the name of the command. Note that `\input*foo` opens file ‘\*foo’ while `\Input*foo` complains about missing braces in the argument and opens file ‘f’. If a star is given, the @ sign character is of category code 11 while reading the file (L<sup>A</sup>T<sub>E</sub>X uses `\@pushfilename` and `\@popfilename` for managing, among others, the category code of @ while reading packages, Tralics uses this command).
- `\endinput`. This command has as effect to close the current file (in Tralics, the file is read once and for all, closing the file means ignoring all unprocessed lines). Here is an example. This assumes that there are no files `nohope`, `nohope.tex` neither `X.tex`, but a file `taux2.tex` that contains a call to `\mytypeout`.

```

{
  \IfFileExists{nohope}{\errmessage{bad1}}{}
  \IfFileExists{\jobname}{\errmessage{bad2}}{}
  \IfFileExists{X.tex}{\errmessage{bad3}}{}
  \def\bad{\errmessage{BAD}}\let\ybad\bad
  \def\mytypeout#1{\def\bad{\xbad}}
  \def\foo{\ifx\bad\ybad\else\let\xbad\relax\fi\let\bad\ybad}
  \def\IIFE#1{\InputIfFileExists{#1}{}{}}
  \IIFE{taux2}\IIFE{nohope}\foo\IIFE{taux2}
  \bad
}

```

We can define `taux2.tex` by the following code

```

\begin{filecontents}{taux2.tex}
% aux file for testing tralics

```

```

% this file contains nothing useful
\mytypeout{in file taux2.tex}
\endinput
The file should finish with a \endinput, but not on the last line.
\end{filecontents}

```

Assume now that `taux1.tex` contains

```

% aux file for testing tralics
% this file contains nothing useful
\mytypeout{in file taux1.tex}
\input taux2
% the file should finish with a comment

```

The following lines should work:

```

\let\mytypeout\typeout
\ifnum \tracingcommands=0 \def\mytypeout{\write -1 }\fi
\input taux1.tex\input taux2.tex
\input{taux1.tex}\input{taux2.tex}
\def\foo{\input taux1 \input taux2 }
\foo

```

The transcript file contains lines like the following. Note the special `\relax` token inserted by the second `\input` if the first is scanning its arguments (remember: `\input` is expandable).

```

[2380] \input taux1.tex\input taux2.tex
{\input}
{insert \relax for \input}
{\input taux1.tex}
++ file taux1.tex exists
++ Input stack ++ 1 taux1.tex
++ Opened file taux1.tex; it has 5 lines

```

Later on, at the end of the first file, we can see that the unread tokens are read in the right order (first the `\relax`, then the `\input` and finally the unread characters `taux2.tex`).

```

++ End of file taux1.tex
++ cur_file_pos restored to 0
++ Input stack -- 1 taux1.tex
{\relax}
{\input}
{\input taux2.tex}

```

## 5.13 Other non-expandable commands

- `\ignorespaces`. Tokens are fully expanded, until a non-space character is seen. Spaces are ignored.
- `\mark`. An expanded token list is read, the result is discarded.
- `\penalty`. An integer is read, but nothing happens.
- `\lastkern`, `\lastpenalty`, `\lastskip`. Since `Tralics` does not insert kerns, penalties, or glue in the boxes, it is not possible to fetch the last such quantity. In `TeX` you can get an error of the form *You can't use '\lastpenalty' in vertical mode*. In `Tralics` the message is: *Read only variable \lastpenalty*. You can use `\the` to access the variable, but the result is zero (depending on the command, this is an integer, a dimension, or glue).

- `\unpenalty`, `\unkern`, `\unskip`. These are assumed to remove the last penalty, kern or glue. They are defined as a no-op.
- `\shipout`. This reads a box number and signals an error `\shipout is undefined`.
- `\setlanguage`. This command reads an integer and does nothing.
- `\@@end`. This command can be used to stop translation. It closes all files. The end-document hooks are not applied.
- `\message`. This is like `\write-1`. The argument is expanded and printed.
- `\typeout`. This is like `\write17`. The argument is expanded and printed on the terminal and the transcript file.
- `\errmessage`. This is as above, but an error is signaled. For instance `\errmessage {This \iftrue can \else cannot \fi happen}` will give: *Error signaled at line 1: This can happen*.
- `\noboundary`. Unimplemented.
- `\insert{arg}`. Unimplemented, an error is signaled.
- `\vadjust`. Unimplemented, an error is signaled.
- `\unhbox`, `\unvbox`, `\unhcopy`, `\unvcopy`. These commands read a box number, say N. The `\unhbox` command calls `\leavevmode` (in L<sup>A</sup>T<sub>E</sub>X, `\leavevmode` uses `\unhcopy`). A copy of the box register N is put on the main tree. In the case of `\unhbox` and `\unvbox`, the box is cleared.
- `\centering`. This command changes the current value of the internal centering number, and adds an attribute to the current paragraph (the numbers mean normal, centered, flush left, flush right, quoted).
- `\nocentering`. This sets the internal centering number to ‘normal’. The next paragraph will not have a centering attribute.
- `\href`. The command takes two arguments, say ‘foo’ and ‘bar’. The translation is an `<xref>` element, containing `\bar`, with an attribute `url`, with value foo. The tilde character can be inserted in the first argument using `\~` (in the second argument, it produces an accent).
- `\htmladdnormallink`, `\Hhref`. Like `\href`, but the order of arguments is changed.
- `\htmladdnormallinkfoot`. Like the previous (but the link is not placed in a footnote).
- `\url`. This takes one argument. Normally `\url{foo}` is the same as `\href{foo}{foo}`. However, the element is translated using the value of `\urlfont`, which is empty by default. In a construction like `\href{foo}{\url{bar}}`, the `\url` command does nothing, as well as in the case `\url{\rrrt{foo}}`. Characters tilde, underscore and sharp sign in the argument are handled as ordinary characters.
- `\rrrt`. This command takes one argument. Its translation is `\url{prefix/foo.html}`, if the argument is foo, where the prefix is ‘http://www.inria.fr/rrrt’. This was useful for the Raweb, but since 2006, no new documents are put on that server.
- `\caption`. This command takes an optional argument and a required argument. The optional argument is ignored. The result is a `<caption>` element. It can contain paragraphs. An implicit `\nocentering` command is executed.

- `\footnote`. This command takes one argument. The result is a `<note>` element. It can contain paragraphs. An implicit `\nocompiling` command is executed. If the result contains a single `<p>` element, the `<p>` will be replaced by its content (unless in hack mode).
- `\thanks` can be used to attach a footnote to an author on a title page; is the same as `\footnote` in `Tralics`.
- `\anchor`. Translation is an empty `<anchor>` element that can be used as target of a `\label`. If you say `\label{xx}` followed by `\ref{xx}`, there will be a link from the ref to the anchor. The question is: how to implement this? there is no notion of anchor in `TeX`, the only possibility is to use the `\special` command, a whatsit, that is interpreted by dvi processors. In `Tralics`, an anchor is an element that has an attribute `id` with some value, and the link is a `<ref>` element whose `target` attribute is this value; in `HTML`, a link is a `<a>` element with a `href` attribute, and an anchor is `<a>` element with a `name` attribute, or any element with an `id` attribute `id`.

In `LATEX`, an anchor is defined by a number and a page, but the `hyperref` package makes this more complicated; for the label associated to the current section, we have five values, the number, the page, the title of the section, an identifier ‘section.5.12’ and an empty list. These quantities are written in the aux file, and read again. The command `\refstepcounter` defines (locally) `\@currentlabel`, and this contains the number used above. The page number is computed when the whatsit is effectively written in the dvi. In the pdf version, if you click on the number, you will go to the page containing the anchor, i.e., the title of the section.

This mechanism is different in `Tralics`, because we store only one quantity, not two or five. We assume that section numbers, equation numbers, etc., are computed by the XML processor (thus, there is no need for two passes and an auxiliary file). A potential anchor is defined for each section, equation, footnote, etc., this is the same as the action of `\refstepcounter`. When a document like this one is converted from XML to HTML, the style sheet can compute the equation number, section number, etc., depending on the type of the anchor.<sup>8</sup> If a precise location is needed, you can use `\pageref`, a command that was not originally accepted by `Tralics`, since no page number is constructed. If you want to refer, in the XML version, to a precise point, you must also add an anchor at the right place, using `\anchor`. Here are some examples: a link to this item 5.13 (`Tralics` adds an anchor to each item, in `LATEX`, this works only for enumerations, in fact, environments that increment a counter), a link to this anchor 5.13 (defined in the non-`LATEX` version at the start of this sentence), a link to the footnote 8, a citation [4], and a reference to a section 5.13; and two uses of `\pageref`: 182 and 180. In this case, the anchors are the same, but not the page numbers, this is strange.

- `\label`. This command takes an argument, and defines it as the name of an anchor to be used by `\ref`. The anchor is an `id` attribute of the element associated to a section command, a footnote, an item in an enumeration, a formula, a figure, etc. The translation is empty.
- `\eqref`. Like `\ref`, but parentheses are put around the reference.
- `\ref`. This takes an argument, that must be defined by `\label`. The translation is a `<ref>` element.
- `\pageref`. Like `\ref`, but the element has an attribute `rend` with value ‘page’.
- `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection` `\paragraph`, `\subparagraph`. To each command is assigned an integer between 0 and 6. It is decremented by 1 if the leading section is ‘chapter’, by 2 if it is ‘section’ (negative numbers are replaced by 0). If this gives `N`, the result is a `<divN>` element. The command takes an argument, the title,

---

<sup>8</sup>Dummy footnote here



that is translated in a <head> element. Before the title, an optional argument is ignored. The title is printed in the terminal if this is at highest level. A section can be the target of a \label. The element will have the attribute pair rend='nonnumber', in the case of a starred section (i.e. if either a star has been given after the command name, or in case of \chapter in \frontmatter or \backmatter).

- \endsec. This command takes an argument. Nothing happens if it is empty. If the first argument is one of the 7 section commands described above, it will be terminated; more precisely, if this section command has number  $N$ , then all divisions with number  $N$  and greater will be terminated.
- \mainmatter, \frontmatter, \backmatter. Translation is an element with the same name, one of <mainmatter>, <frontmatter>, <backmatter>. Using these commands sets a global variable that says whether or not chapters have a default star. These commands execute \endsec\part. In L<sup>A</sup>T<sub>E</sub>X, some additional actions are performed (like: start a new page, change pagENUMBERING, etc). No counter is modified. Example:

```

\mainmatter
\section{A}\subsection{B}
\paragraph{C} x \label{a1}
\paragraph{D} \label{a2}x\label{a3}
\endsec{\subsection}
\paragraph{E} x\footnote{x\label{a4}}
\endsec{\part}
\paragraph{F} x
\ref{a1} \ref{a2}\ref{a3}\ref{a4}
\section{G}
\paragraph{H} x
\backmatter
\chapter{I} y

```

In the configuration file, we have given alternate names for all elements. The book class is used, so that \section is a div2. When translating, Tralics prints *Translating section command div2: A*, then *Translating section command div2: G*, then *Translating section command div1: I*. After that, it will not print section titles anymore (however, all titles are printed in the transcript file). Translation is

```

<MainMatter>
  <Section id='uid1'><Head>A</Head>
    <Subsection id='uid2'><Head>B</Head>
      <Paragraph id='uid3'><Head>C</Head>
        <p>x</p>
      </Paragraph>
      <Paragraph id='uid4'><Head>D</Head>
        <p>x</p>
      </Paragraph>
    </Subsection>
    <Paragraph id='uid5'><Head>E</Head>
      <p>x<Footnote id='uid6' Place='as a note'><p>x</p>
        </Footnote></p>
    </Paragraph>
  </Section>
  <Paragraph id='uid7'><Head>F</Head>
    <p>x <ref target='uid3' />
      <ref target='uid4' /><ref target='uid4' /><ref target='uid6' />

```

```

    </p>
  </Paragraph>
  <Section id='uid8'><Head>G</Head>
    <Paragraph id='uid9'><Head>H</Head>
      <p>x</p>
    </Paragraph>
  </Section>
</MainMatter>
<BackMatter>
  <Chapter id='uid10' rend='nonnumber'><Head>I</Head>
    <p>y</p>
  ...

```

- `\aparaitre`, `\toappear`. These two commands are for the Raweb. They translate as ‘à paraître’ (with the correct spelling) or ‘to appear’, depending on the current language.
- `\xmllatex`. This is a command that takes two arguments, and ignores the second. In the first, commands are replaced by their names. For instance `\xmllatex{\&\#x25;}{etc}` translates as `&\#x25;`. Normally, the translation of `\&` is `&amp;`.
- `\xmlelt`. This command takes two arguments, say `foo` and `bar`. The translation is a `<foo>` element containing the translation of `bar`. The command calls `\leavevmode`.
- `\xmleptyelt`. This command takes one argument, say `foo`. The translation is an empty `<foo/>` element (in fact a string that prints like an element, you cannot add attributes to it). The command does not call `\leavevmode`.
- `\xbox`. This command takes two arguments, say `foo` and `bar`. The translation is a `<foo>` element containing the translation of `bar`. This command behaves like `\hbox`, in that it does not read the second argument, so that category code changes are allowed. The token list `\everyxbox` is inserted in the input stream after the first argument and opening brace of the second one have been read.
- `\vbox`, `\hbox`. These commands read their argument as described in the `TEX` book, but specification arguments are ignored; the result is an unnamed XML element. The token list `\everyvbox` or `\everyhbox` is inserted in the input stream after the opening brace has been read.
- `\leaders`, `\cleaders`, `\xleaders`. These commands read a box or a rule, that is supposed to be followed by some glue. Two non trivial examples are (`TEX`book exercise 21.7 and 21.8)

```

  \null\nobreak\leaders\hrule\hskip10pt plus1filll\ \par
  $$\hbox to 2.5in{\cleaders
    \vbox to .5in{\cleaders\hbox{\TeX}\vfil}\hfil}$$

```

Let’s explain the second example first. We have vertical and horizontal leaders; the result is rectangle of size 2.5 times 0.5 inches containing four rows and nine columns of the `TEX` logo. The size occupied by the logos is defined by the size of the glue; in this example, the glue is `\vfil`, or `\hfil`, the size of which is the size of the box, as computed by `TEX` when the box is packaged; in the example, the size of the box is explicit. Due to limitations of MathML, when `Tralics` sees a box in math mode, it signals an error for every non-character in the box; for this reason the example will not compile, and we give below the translation without the dollar signs. As explained above, `Tralics` reads the width of the box, but ignores it; in the same manner, it is unable to compute the size of the logo, thus cannot know how many copies to insert.

The first example is interesting: the idea is to create a rule, that has at least ten points, reaches the right margin (as a consequence the size of the rule is known only when the current paragraph is split into lines). For this reason, the glue specification is: ten point plus enough to go to fill the line. The non trivial point is that the last line of a paragraph does in general non reach the right margin: this is because T<sub>E</sub>X adds `\parfillskip` glue, so that a fill with three Ls is required; moreover, since an end-of-paragraph is generally obtained by a blank line, and the end-of-line character of the line before the blank one produces some glue, T<sub>E</sub>X inserts `\unhskip`, even if the end-of-paragraph is given by an explicit `\par`; this one can remove the leader glue, so that a backslash-space is inserted in the example. For all these reasons, implementing leaders properly is nearly impossible; the leader command interpreter of Tralics does not read the glue that follows (and you will never see an error like *Leaders not followed by proper glue*). The translation given here shows that translation of glue commands could be better.

```
<leaders><rule depth='0.0pt' height='0.4pt' /></leaders>
  <p>&#xA0;&#xA0;&#xA0;</p>
<cleaders><cleaders><TeX/></cleaders><vfil/></cleaders><hfil/>
```

- `\moveleft`, `\moveright`. These commands read a dimension and a box; but the box is not moved. Compare with `\raise` and `\lower`.
- `\vtop`. This is the same as `\vbox`.
- `\vsplit`. You can say `\vsplit0 to 3cm`. The result is the same as `\copy0`.
- `\lastbox`. This returns the last element on the stack, unless the top-stack is a character, case where the result is the empty box. Example:

```
\def\dupbox{\setbox0=\lastbox \copy0 \box0 }
\abox{foo}{ok} and \dupbox; \abox{bar}{ok}\dupbox.
```

The translation is:

```
<foo>ok</foo><p>and ; <bar>ok</bar><bar>ok</bar>.
```

- `\begin{rawxml} ... \end{rawxml}`. This is like a verbatim environment, but special characters like `&` and `<` are not escaped. For instance

```
let\verbatimfont\relax
\begin{verbatim}
{\let\rm\bf \bf totoé}
<!--this is a comment -->
&Dollar; not &Equals; &Euro;
\end{verbatim}
% See comment below
\begin{rawxml}
{\let\rm\bf \bf totoé}
<!--this is a comment -->
&Dollar; not &Equals; &Euro;
\end{rawxml}
```

translates (with the `'nozerowidthspace'` switch) to

```
<p noindent='true'><hi rend='tt'
  >{\let\rm\bf&#xA0;\bf&#xA0;toto&#xe9;}</hi></p>
<p noindent='true'><hi rend='tt'
  >&lt;!--this&#xA0;is&#xA0;a&#xA0;comment&#xA0;--&gt;</hi></p>
<p noindent='true'><hi rend='tt'>&amp;Dollar;&#xA0;not&#xA0;&amp;
```

```

Equals; &#xa0; & Euro; </hi></p>
<p noindent='true'>{\let\rm\bf \bf toto&#xe0;}
<!--this is a comment -->
&Dollar; not &Equals; &Euro;</p>

```

- `\begin{xmlelement} ... \end{xmlelement}`. Like `\xbox`, but this is an environment. You can put a star after the name of the environment. The effect is the following. If no star is used, then horizontal mode is entered before creating the element. Otherwise, the element just created is typeset in vertical mode. If a '+' sign is used instead of '\*', then `\leavevmode` is called, translation starts in vertical mode. For instance

```

\begin{xmlelement}{foo1} A \end{xmlelement}\par
\begin{xmlelement*}{foo2} B \end{xmlelement*}\par
\begin{xmlelement+}{foo3} C \end{xmlelement+}\par

```

translates as

```

<p> <foo1> A </foo1> </p>
  <foo2><p>B </p> </foo2>
<p> <foo3><p>C </p></foo3></p>

```

- `\AddAttToCurrent`. This command takes two arguments, and constructs an attribute pair that will be added to the current element.
- `\AddAttToLast`. This command takes two arguments, and constructs an attribute pair that will be added to the last created element.
- `\addattributestodocument`. This command takes two arguments, and constructs an attribute pair that will be added to the main document element. This is an example:

```

\begin{xmlelement}{foo}
\begin{xmlelement}{subelt1}
texte1
\end{xmlelement}
\begin{xmlelement}{subelt2}
texte2
\end{xmlelement}
\AddAttToLast{sb2-att}{value1}%
\AddAttToLast{sb3-att}{}%
\AddAttToCurrent{foo-att}{att-value''}%
\end{xmlelement}

```

The translation is

```

<foo foo-att='att-value&apos;&apos;'><subelt1>texte1
</subelt1>
<subelt2 sb3-att='' sb2-att='value1'>texte2
</subelt2>
</foo>

```

The commands described above can be followed by a '\*'. In that case, an existing attribute will be overwritten. Example:

```

\AddAttToCurrent{x}{1} % ignored
\AddAttToCurrent{y}{2} % ok
\AddAttToCurrent*{x}{3} % ok
\AddAttToCurrent{y}{4} % ignored

```

- `\XMLaddatt`. This command takes an optional star, an optional integer, and two arguments, and constructs an attribute pair that will be added to the element defined by the number. If no number is given, the current element is used. The main XML element has id number one, the bibliography has number three, the table of contents has number four, the index has number five. You can add an attribute to these elements before their creation (for instance in a package). Never use number two.

The following example shows usage.

```

{\everyxbox{Vest}
\setbox0=\xbox{foo}{1\xbox{bar}{2} %
  \XMLaddatt[\the\xMLlastid]{x}{1}%
  \XMLaddatt[\the\xMLcurrentid]{y}{2}%
  \XMLaddatt{y}{22}3}
\showbox0
}
{\everyxbox{West}
\setbox0=\xbox{foo}{1\xbox{bar}{2} %
  \XMLaddatt*[\the\xMLlastid]{x}{1}%
  \XMLaddatt*[\the\xMLcurrentid]{y}{2}%
  \XMLaddatt*{y}{4}3}
\showbox0
}

```

it should print

```

<foo y='2'>Vest1<bar x='1'>Vest2</bar> 3</foo>
<foo y='4'>West1<bar x='1'>West2</bar> 3</foo>

```

- `\begin{glossaire} ... \end{glossaire}` This is a command for the raweb. It contains `\glo` commands.
- `\usecounter`. An error is signaled if the argument is not the name of a counter. Otherwise this becomes the counter to be used by the current list. It is reset to zero.
- `\labelitemi`, `\labelitemii`, `\labelitemiii`, `\labelitemiv`. This is used by L<sup>A</sup>T<sub>E</sub>X as default value of optional argument of `\item` in an `itemize` environment. Not used by Tralics.
- `\begin{itemize}`, `\begin{description}`, `\begin{enumerate}`, `\begin{list}`. These commands behave as in L<sup>A</sup>T<sub>E</sub>X. The semantics changed in version 2.8. A counter may be used, via `\usecounter`. This is `enumi`, `enumii`, `enumiii`, `enumiv`, depending on the enumeration level (the last counter is used in case the nesting level is too high). No counter is used in other cases. If the environment is `list`, then two argument are read. The first argument is put in `\@itemlabel`, the second is read again, and evaluated. It might contain a call to `\usecounter`. If the environment is not `list`, then `\@itemlabel` is relax. The translation is `<list>` element, with an attribute `rend='simple'`. It could be `description` or `enumerate`. In the case of `list`, it is `description`, but this can be changed in a configuration file. The second argument of the environment could also change the value of the attribute.

We give here an example, to be used later. The first argument of the list environment explains how to typeset the value of the counter. The second argument uses `\usecounter`. The outer list environment saves/restores its counter into the `exx` counter.

```

\newcounter{exx}
\newcounter{xnumi}
\newcounter{xnumii}
\def\exe{\begin{list}{\thexnumi}{\usecounter{xnumi}}%

```

```

\setcounter{xnumi}{\value{exx}}
\def\endexe{\setcounter{exx}{\value{xnumi}}\end{list}}
\def\xlist{\begin{list}{\thexnumi.\alph{xnumii}}{\usecounter{xnumii}}
\def\endxlist{\end{list}}

```

- `\item`, `\@item`. This command takes an optional argument, that will produce a `<label>` element. What follows `\item` is translated in a `<item>` element. The end of the element is defined by a following item or the end of the list (thus, you should use this command only in a list).

In the case where `\usecounter` has defined a counter for the current list, it will be globally incremented. If no optional argument is given, but if `\@itemlabel` is not `\relax`, the value of this command (which had better be a user-defined command without argument) is used instead. It will be translated in a group. This means that in a case like `\item[{\bf x}]`, only `x` is in bold face. The translation is something like `<label>foo</label>`. In the case of `\item`, this is added to the XML tree before the `<item>` element. In the case of `\@item`, the content is added as attribute label to the item element (font changes are forbidden in a case like that). Example

```

\makeatletter\let\item\@item\makeatother
\begin{exe}
\item a
\item b
\end{exe}
\begin{exe}
\item c
\begin{xlist}
\item c1\label{xx}
\item c2
\end{xlist}
\item d
\end{exe}

```

Translation is given here.

```

<list type='description'>
<item id='uid1' label='1'><p noindent='true'>a</p>
</item>
<item id='uid2' label='2'><p noindent='true'>b</p>
</item></list>
<list type='description'>
<item id='uid3' label='3'><p noindent='true'>c</p>
<list type='description'>
<item id='uid4' label='3.a'><p noindent='true'>c1</p>
</item>
<item id='uid5' label='3.b'><p noindent='true'>c2</p>
</item></list>
</item>
<item id='uid6' label='4'><p noindent='true'>d</p>
</item></list>

```

If you make a reference to label `xx`, this will produce a link to the element with id `uid4`. It is very easy to associate some text (say `3.a`) to this reference: just take the label of the element. If the example had used the original `\item` command, this would have been harder. For this reason, if the configuration file says `alternate_item="true"`, the `\let` assignment shown

above is automatically done. Original definition can be found in `\@item`. The raweb DTD says that an item label should be an element that comes before the `<item>`, and the style sheet uses this construct. The style sheet for converting this document to HTML assumes on the other hand that labels are attributes.

- `\lhead`, `\rhead`, `\chead`, `\cfoot`, `\lfoot`, `\rfoot`. These commands are available only if the `fancyhdr` package is loaded. They take an optional argument, say `A`, and a required argument, say `B`. The default value of `A` is `B`. The result is the same as `\@IF{elh}{A}` followed by `\@IF{olh}{B}`, where ‘lh’ must be replaced by the first two characters of the command. Here `\@IF` is a pseudo-command described below.
- `\fancyhead`, `\fancyfoot`, `\fancyhf`. These commands are available only if the `fancyhdr` package is loaded. They take two arguments, an optional one `x`, and a required argument `y`. The argument `x` is a sequence of letters, separated by commas. In each sequence, only letters `e`, `o`, `l`, `c`, `r`, `h`, and `f` are considered. If none of `e` or `o` are given, both are added; if none of `l`, `c`, `r` and given, all three are added; for `\fancyhead`, `h` is added, for `\fancyfoot`, `f` is added; if none of `h` or `f` are given, both are added. After that we consider all sequences `XYZ`, where `X` is one of `e`, `o`, `Y` is one of `l`, `c`, `r`, `Z` is one one `h`, `f`, the letters being in the list so completed (at least one sequence matches). For each match we call `\@IF{XYZ}{y}`. Here `\@IF` is a pseudo-command described below.
- `\@IF`. This is a pseudo command, that takes two arguments, say `x` and `y`. The translation is `\fancyinternal {x} {\let \thepage \inert\thepage y}`.
- `\inert\thepage`. The translation is `<thepage/>`.
- `\fancyplain`. Not yet implemented. It takes two arguments, that are conditionally evaluated, depending on whether this is a “plain” page or not.
- `\fancyinternal`. The command takes two arguments `x` and `y`. It constructs a `<headings>` element, containing `y`, with an attribute `type` with value `x`. Example:

```
\newcommand\rightmark{\xmlelement{rightmark}}
\newcommand\leftmark{\xmlelement{leftmark}}
\newcommand\fancyplain[2]{#2}
\lhead[\fancyplain{}{\bf\thepage}]{\fancyplain{}{\sl\rightmark}}
\rhead[\fancyplain{}{\sl\leftmark}]{\fancyplain{}{\bf\thepage}}
\cfoot[\fancyplain{\bf\thepage}{}]{\fancyplain{\bf\thepage}{}}
\chead{} \lfoot{} \rfoot{}
```

The translation is

```
<headings type='olh'><hi rend='slanted'><rightmark/></hi></headings>
<headings type='elh'><hi rend='bold'><thepage/></hi></headings>
<headings type='orh'><hi rend='bold'><thepage/></hi></headings>
<headings type='erh'><hi rend='slanted'><leftmark/></hi></headings>
<headings type='ocf' />
<headings type='ecf' />
<headings type='och' />
<headings type='ech' />
<headings type='olf' />
<headings type='elf' />
<headings type='orf' />
<headings type='erf' />
```

- `\@ifundefined`. The command takes three arguments. The first argument should be a sequence of letters, for instance ‘foo’. If `\foo` is undefined, the expansion of the command is

the value of the first argument, otherwise, the value of the second. Note: `\csname` is used to construct the command, so that ‘undefined’ is the same as being `\relax`. If the first argument is `\par`, an error is signaled, of the form *Invalid token \par found while scanning command for \@ifundefined*. If the argument is `\xpar`, the error is *Undefined command \xpar*, and `\relax` gives: *Invalid token \relax found while scanning command for \@ifundefined*. Note that Tralics may complain with an extra `\endcsname`.

As an example, this should give 2.

```
\makeatletter
\def\Foo{foo}\def\Bar{bar}
\@ifundefined{\Foo}{\@ifundefined{\Bar}{1}{2}}{3}
```

- `\epsfbox`, `\epsffile`. These commands do exactly the same thing. They take one argument, say X. It behaves as `\includegraphics[Y]{X}Z`, where Y contains ‘width=w’ and ‘height=h’, where w and h are the values of `\epsfxsize` and `\epsfysize`, provided they are not zero, and Z is the code that sets these values to zero.
- `\psfig`, `\epsfig`. The command takes an argument, which is an association list. It must contain `file=X`, or `figure=X`. If Y is the remaining of the list, the command is equivalent to `\includegraphics[Y]{X}`.
- `\includegraphics`. The command takes an optional argument and a required argument. The optional argument is an association list, with names can be clip, angle, height, width, scale. The required argument is a file name. There is no difference between ‘foo’, ‘foo.ps’ and ‘foo.eps’. Not more than one dot is allowed in the file name. See examples section 6.15.
- `\lsc`, `\fsc`. These commands are like `\textsc`.
- `\@ifstar` The command takes two arguments. After that, it looks at the next character (ignoring spaces). If the character is a star, it is read, the expansion is the first argument, otherwise the second. The following code produces B1A2.

```
\makeatletter
\def\foo{\@ifstar{A}{B}}
\foo1\foo*2
```

- `\@ifnextchar`. The command takes three arguments, a character token, say w, and two commands say A and B. After that, it looks at the next character (ignoring spaces). If the character is w, the expansion is A, else B. The character is not read. The following should produce: ‘With opt AABBBWithout opt CC’.

```
\makeatletter
\def\ifbracket{\@ifnextchar[]}
\makeatother
\def\wopt{\ifbracket\xwopt\ywopt}
\def\xwopt[#1]#2{With opt #1#2}
\def\ywopt#1{Without opt #1}
\wopt[AA]{BB}\wopt{CC}
```

- `\batchmode`, `\nonstopmode`, `\scrollmode`, `\errorstopmode`. These commands do nothing in Tralics, because the only supported mode is ‘batchmode’. In T<sub>E</sub>X, they change an internal variable, so that they can be preceded by the prefix `\global`.
- `\@gobble`, `\@gobbletwo`. These commands read one or two argument. The result is empty.
- `\discretionary`. This command reads two arguments but does nothing with it. In T<sub>E</sub>X, there is a third argument, that is typeset.



- `\HTMLset` (Raweb only command). This command reads two arguments but does nothing with them.
- `\RALabel` (Raweb only command). This command reads one argument but does nothing with it.
- `\special`, `\patterns`, `\hyphenation`. These command read one argument but do nothing with it.
- `\protect` behaves like `\relax`.
- `\htmlimage` was once defined in the kernel (is now is the html package).
- `\centerline`, `\leftline`, `\rightline`: these commands reads an argument, typeset it, and return a `<line>` element with an attribute (left, right, center).
- `\clearpage`, `\cleardoublepage`, `\allowbreak`: an empty element is returned.
- `\sloppy`, `\subitem`, `\immediate`, `\break`, `\nobreak`, `\@`, `\/`, These command are currently defined as doing nothing.
- `\begin{latexonly} ... \end{latexonly}`, `\begin{xmlonly} ... \end{xmlonly}`, `\begin{subequations} ... \end{subequations}`. The environment is ignored, not the content.
- `\begin{htmlonly} ... \end{htmlonly}`, `\begin{comment} ... \end{comment}`, `\begin{rawhtml} ... \end{rawhtml}`, `\begin{LaTeXonly} ... \end{LaTeXonly}`, The environment is ignored, the content also.
- `\centering`, `\raggedright`, `\raggedleft`. These commands are associated to the environments: ‘center’, ‘flushleft’, ‘flushright’. There are two other environments ‘quote’ and ‘quotation’. This gives an integer with values between 1 and 5, the normal value being 0. Whenever a new paragraph is started, the value of the integer considered. It is non-zero, an attribute is added (for instance `rend='center'`). If you say `\begin{center}`, this will terminate the current paragraph (and the next one will be centered). If you say `\end{center}`, this will terminate the current paragraph (and the next one will not be centered). If you say `\centering`, this will add an attribute to the current paragraph.
- `\error`. This command takes an argument and signals an error.
- `\ClassError`, `\ClassErrorNoLine`, `\PackageError`, `\PackageErrorNoLine`, `\GenericError`, `\ClassInfo`, `\PackageInfo`, `\GenericInfo`, `\ClassWarning`, `\ClassWarningNoLine`, `\PackageWarning`, `\PackageWarningNoLine`, `\GenericWarning`, `\@latex@error`, `\@latex@warning`, `\@latex@warning@no@line`, `\@latex@info`, `\@latex@info@no@line`. All these commands take an error message (that could be a warning, or some other piece of information), in which `\MessageBreak` is interpreted as a special new line character. The first argument can be a package or class name, or some generic prefix. Information is printed on the transcript file only, errors and warnings and also printed on the terminal. The last two arguments of `GenericError` are ignored, and the last arguments of other errors commands are ignored (these are generally of the form: see the class or package for explanation, and written by L<sup>A</sup>T<sub>E</sub>X, on the terminal when the user asks for help). Tralics automatically adds the current line and file information, unless the adequate prefix is given (or, as a special case, if the last item in the message is `\@gobble`). Here is an example

```

\PackageError{mypack}{you cannot use the command\MessageBreak
\string\foobar\space here}{unused}
\PackageWarning{mypack}{you should not use the command\MessageBreak
\string\foobar\space here}{unused}

```



```

\makeatletter
\notrivialmath=1
\def\foo#1{x#1x}
\def\xbar{\catcode'\$=12\catcode'\^=12 \ybar}
\def\ybar#1{y#1y}
\newenvironment{wbar}{\catcode'\$=12\catcode'\^=12w}{w}
\newcommand\Fct{\@reevaluate\foo\xbar}
\newenvironment{Env}{\@reevaluate*{center}{wbar}}{}

```

the translation of

```

{\Fct{$1^{\er}$}}
\begin{Env}$3^{\eme}$ \end{Env}

```

is

```

<p>x1<hi rend='sup'>er</hi>xy$1^{\er}$y</p>
<p rend='center'>3<hi rend='sup'>e</hi></p>
<p>w$3^{\eme}$ w</p>

```

This is a part of the transcript file showing the expansion of the command.

```

[11] {\Fct{$1^{\er}$}}
{begin-group character}
+stack: level + 2 for brace
\Fct ->\@reevaluate \foo \xbar
{\@reevaluate}
{Reeval: \foo{$1^{\er}$}}%
\xbar{$1^{\er}$}%
}

```

This shows the expansion in the case of a starred command. Note that the current environment is terminated; then everything up to `\end{whatever}` is read.

```

[12] \begin{Env}$3^{\eme}$ \end{Env}
{\begin}
{\begin Env}
+stack: level + 2 for environment
\Env ->\@reevaluate *{center}{wbar}
{\@reevaluate}
+stack: ending environment Env; resuming document.
+stack: level - 2 for environment
{Reeval: \begin{center}$3^{\eme}$ \end{center}}%
\begin{wbar}$3^{\eme}$ \end{wbar}%
}

```

## 5.15 Trees

We explain here some commands from the `tree-dvips` package by Emma Pease. A tree is defined by some nodes and connectors. Each node has a name, whose scope is limited to the current page (Tralics does no validity test for the names). A connector can be attached to the top, bottom, left or right of a node (abbreviation is one character of ‘tblr’), or a corner (two letter, one of ‘tb’ followed by one of ‘lr’).

- `\node{N}{V}` creates a `<node>` element, whose content is the translation of `V`, with a `name` attribute `N`.

- `\nodepoint{N}[h][v]` creates an empty `<node>`, with a name attribute N. It has optional attributes `xpos` and `ypos`, with value `h` and `v`.
- `\nodeconnect[f]{F}[t]{T}` creates a `<nodeconnect>` element, with attribute `nameA` equal to `F`, attribute `nameB` equal to `T`, attributes `posA` and `posB` equal to `f` and `t`. These must be positions; if omitted, or invalid syntax, then `'t'` and `'b'` are used (bottom of first node is connected to top of second node).
- `\anodeconnect`. Same as above, but the element is named `<anodeconnect>` (it has an arrow from the first node to the second).
- `\barnodeconnect[d]{F}{T}` creates a `<barnodeconnect>` element, with attribute `nameA` equal to `F`, attribute `nameB` equal to `T`, attribute `depth` equal to `d`; this should be a dimension (not tested by Tralics).
- `\abarnodeconnect`. Same as above, but the element is named `<abarnodeconnect>` (it has an arrow from the first node to the second).
- `\nodecurve[f]{F}[t]{T}{d1}[d2]` is like `\nodeconnect`, but produces a `<nodecurve>` element, with two additional attributes `depthA` and `depthB`, containing the value of `d1` and `d2` (default value of `d2` is `d1`).
- `\anodecurve`. Same as above, but the element is named `<anodecurve>`.
- `\nodetriangle{F}{T}` creates a `<nodetriangle>` element, with the two names.
- `\nodebox{T}` creates a `<nodebox>` element, with a single name, it adds a decoration to the node.
- `\nodeoval{T}` creates a `<nodeoval>` element, with a single name, it adds a decoration to the node.
- `\nodecircle[d]{T}` creates a `<nodecircle>` element, with a single name, and attribute `depth` with value `d`; it adds a decoration to the node.

For instance

```

\node{a}{Value of node A}
\nodepoint{b} \nodepoint{c}[3pt]\nodepoint{d}[4pt][5pt]
\nodeconnect{a}{b}
\nodeconnect[tl]{a}[r]{c}
\nodeconnect{a}{b}
\nodeconnect[tl]{a}[r]{c}
\barnodeconnect[3pt]{a}{d}
\nodecurve{a}{b}{2pt} ?
\nodecurve[l]{a}[r]{b}{2pt}[3pt]
\nodetriangle{a}{b}
\nodebox{a}
\nodeoval{a}
\nodecircle[3pt]{a}

```

Translation

```

<node name='a'>Value of node A</node>
<node name='b' />
<node xpos='3pt' name='c' /><node ypos='5pt' xpos='4pt' name='d' />
<nodeconnect nameA='a' nameB='b' posA='b' posB='t' />
<nodeconnect nameA='a' nameB='c' posA='tl' posB='r' />

```

```

<anodeconnect nameA='a' nameB='b' posA='b' posB='t' />
<anodeconnect nameA='a' nameB='c' posA='tl' posB='r' />
<barnodeconnect nameA='a' nameB='d' depth='3pt' />
<nodecurve nameA='a' nameB='b' posA='b' posB='t' depthB='2pt' depthA='2pt' />?
<nodecurve nameA='a' nameB='b' posA='l' posB='r' depthB='3pt' depthA='2pt' />
<nodetriangle nameB='b' nameA='a' />
<nodebox nameA='a' />
<nodeoval nameA='a' />
<nodecircle nameA='a' depth='3pt' />

```

## 5.16 Linguistic macros

The `gb4e` package allows you to input the following (extract of the thesis of C. Romero)

```

\begin{exe}
\ex \label{agen1}
\gll ... \th et hit er {\bf \textit{ahte}}.\.
... that OBJ-it already PRET-possessed.\.
\glt \textit{... that (he) already owned it.} (CMLAMBX1,31.377)

\ex \label{agen2}
\gll ... the love that men to hym {\bf \textit{owen}}.\.
... the love that SUBJ-men to OBJ-him PRES-owe.\.
\glt \textit{... the love that men owe him.} (CMCTPARS,313.C2.1087)
\end{exe}

```

The `exe` environment is used for numbered examples; it is implemented as a `list` environment, the `\ex` command behaves like `\item` (each item is numbered, the item number is saved in a global counter). The L<sup>A</sup>T<sub>E</sub>X source of the package (as used by Tralics) can be found in the distribution. The non-trivial part in the example above is the `\gll` command. It takes three lines of text (there is also `\glll` that takes four lines), the first line is a sequence of words (here in old English), the second line another sequence (translated literally, with possible annotations), and the last line is the translation of the whole, with a bibliographic reference. Words in the first two lines are vertically aligned. The algorithm (by Marcel R. van der Goot) is the following; the list is split into words (a space acts as a word separator), each word is typeset via:

```
\hbox{#2\strut#3 }% adds space
```

where `#3` is the word, and `#2` is `\eachwordone` for the first line, `\eachwordtwo` for the second line, and `\eachwordthree` for the third line (case of `\glll`). These commands default to `\rm`. The words are put in a list (a `\vbox`, argument `#1`) like this

```
\setbox#1=\vbox{\hbox{XXX}\unvbox#1}
```

After that, the two or three lists are merged (the code uses `\unvbox` and `\lastbox` in order to get the next element of the list). The command `\vtop` is used to put two words one above the other, and these boxes are merged together using the following code

```
\setbox\gline=\hbox{\unhbox\gline \hskip\glossglue \vtop{XXX}}
```

The glue between the boxes is 0pt plus 2pt minus 1pt (remember that each `hbox` is terminated by some glue). The Tralics implementation is the following. There are two commands `\cgloss@gll` and `\cgloss@glll` written in C++, and the package renames them to `\gll` and `\glll`. It is not clear what the translation should be (a list of boxes containing boxes?) In the current implementation, we use a table. This means that the resulting XML is easy to interpret; the only drawback is that we lose linebreaks (from the `\glossglue`). This is the translation of the example.

```

<list type='description'>
<item id='uid1692' label='650'>
<table rend='inline'><row><cell halign='left'>...</cell>
<cell halign='left'>pet</cell>
<cell halign='left'>hit</cell>
<cell halign='left'>er</cell>
<cell halign='left'><hi rend='bold'></hi><hi rend='it'>
  <hi rend='bold'>ahte</hi></hi><hi rend='bold'>.</cell>
</row><row><cell halign='left'>...</cell>
<cell halign='left'>that</cell>
<cell halign='left'>OBJ-it</cell>
<cell halign='left'>already</cell>
<cell halign='left'>PRET-possessioned.</cell>
</row></table>
<p noindent='true'><hi rend='it'>... that (he) already owned it.</hi>
  (CMLAMBX1,31.377)</p>
</item>
<item id='uid1693' label='651'>
<table rend='inline'><row><cell halign='left'>...</cell>
<cell halign='left'>the</cell>
<cell halign='left'>love</cell>
<cell halign='left'>that</cell>
<cell halign='left'>men</cell>
<cell halign='left'>to</cell>
<cell halign='left'>hym</cell>
<cell halign='left'><hi rend='bold'></hi><hi rend='it'>
  <hi rend='bold'>owen</hi></hi><hi rend='bold'>.</cell>
</row><row><cell halign='left'>...</cell>
<cell halign='left'>the</cell>
<cell halign='left'>love</cell>
<cell halign='left'>that</cell>
<cell halign='left'>SUBJ-men</cell>
<cell halign='left'>to</cell>
<cell halign='left'>OBJ-him</cell>
<cell halign='left'>PRES-owe.</cell>
</row></table>
<p noindent='true'><hi rend='it'>... the love that men owe him.</hi>
  (CMCTPARS,313.C2.1087)</p>
</item></list>

```

## 5.17 Special parsing rules

In the  $\text{T}_{\text{E}}\text{X}$ book, chapter 24, you will find the definition of `(general text)`. This rule explains that  $\text{T}_{\text{E}}\text{X}$  expects a brace-delimited list of tokens, where the starting brace can be either a character, or a token like `\bgroup`; it can be preceded by optional spaces and `\relax` tokens. We give here a list of all cases where this rule can be applied.

- (old behaviour) The procedure that scans a math subformula skips over optional `\relax` commands, and if the token found is not a character (it can be a generalized character like `\mathchar`), it uses this rule. As a result, in the case of `$a_\par b$`, you get a missing opening brace error, and in the case of `$a\par$`, this rule is not applied, you get a missing

dollar error (this dollar marks the end of the formula.) In a case like  $\$a\_relax b\$,$  Tralics removes the `\relax` token before attaching the subscript to the kernel, so that the T<sub>E</sub>X hack is useless (no missing brace/dollar error is signaled, but Tralics may signal a *Bad math expression* involving a `\par` cmd).

- In no-mathml mode, if a token, say `\foo` has the same meaning as `\relax`, it will appear under its name in the result. An expression like  $\$a\_foo b\,$  is valid, so that relax tokens are removed lately.
- In order to simplify error recovery, `\par` tokens are forbidden in math mode; moreover a closing delimiter is added (in the example above, it is a dollar sign that terminates the formula, and a second error will be signaled later)
- The procedure that scans the four arguments of `\mathchoice` uses this rule. You can say `\sqrt\relax{x}` or  $\$a\_relax{b}\,$  and the `\relax` is ignored. Replacing `\relax` by `\par` gives a missing brace error. Note that `\relax` is allowed even in the case where the argument is not delimited by braces, such as in  $\$a\_relax b\,$ .
- The rule applies for commands that produce an accent, like `\bar`; it does not apply for commands like `\frac`; so that `\frac{A}{B}` constructs a fraction with `A` as numerator, and `B` as denominator.
- `\discretionary`. Quoting Knuth “The routine that scans the four mlists of a `\mathchoice` is very much like the routine that builds discretionary nodes.” Tralics ignores first two arguments and translates the last one inside a group. See example below.
- `\hyphenation`, `\patterns`, `\special`. Argument is ignored by Tralics. The rule applies.
- `\insert`, `\vadjust`. These commands are not implemented by Tralics. This means that an error is signaled. However, arguments are scanned as in plain T<sub>E</sub>X; this means that a register number must be given for `\insert`. No error is signaled if this number is 255.
- In the case of `\hbox` or `\vbox`, there can be some keywords (read and ignored by Tralics). The content of the box is defined by this rule.
- Scanning of `\noalign` uses this rule. This is not implemented.
- This rule is used when T<sub>E</sub>X uses the tokens from `\output`. Not implemented in Tralics.
- This rule is used when scanning the token list of `\uppercase`, `\lowercase`, `\message`, `\write`.

Examples.

```
{DISC:\discretionary \relax{1}\relax{2}\relax \bgroup \bf 3}\relax{4}}
\hyphenation\relax\bgroup12}3\patterns\relax\bgroup 45}6
\insert14\relax{\bf A}B\vadjust\relax{\bf C}D
```

Translation

```
<p>DISC:<hi rend='bold'>3</hi>4
36
<hi rend='bold'>A</hi>B<hi rend='bold'>C</hi>D</p>
```





## Chapter 6

# Running Tralics

### 6.1 Introduction

There is a number of ways to alter the translation of your T<sub>E</sub>X document. One solution consists in using a `ult` file: this is a T<sub>E</sub>X file that Tralics loads automatically before the main source. The file has the same name as the main source, with a different extension, and is in the same directory.

All other configuration files are searched in a list of directories (default being `confdir`). There are four such types: file with extensions `clt` and `plt` are T<sub>E</sub>X files, that contains code associated to classes and packages (the `u` in `ult` stands for user, the other letters are for L<sup>A</sup>T<sub>E</sub>X and Tralics).

The file `.tralics_rc` is known as the default configuration file, its use is considered obsolete. Configuration files of this kind consist in a sequence of subfiles, and a rule for choosing a *Type*, that is, either a subfile or an external file, for instance `ra2007.tcf`. The suffix `tcf` stands for Tralics configuration file, there structure and use is explained here. The default value for the *Type* is the current document class. In the description of command line arguments below, some options are marked ‘Raweb only’, this means that they are meaningful only when the *Type* (after removal of trailing digits) is `ra`.

The `tcf` file defines the DOCTYPE: this is the second line of the XML output; if the doctype is `foo+bar.dtd`, this means that the `dtd` file is `bar.dtd` and the root element is `<foo>`. The DOCTYPE can also be given as a command line argument or in the T<sub>E</sub>X source using a special syntax.

The `tcf` file may contain a sequence of assignments. Some of them control the attributes of the root element, but in general they alter the name of XML elements and attributes. These names can also be given as command line argument, or in the T<sub>E</sub>X source.

The `tcf` file may contain some T<sub>E</sub>X code. In fact, the file `ra.tcf` contains code that ought to be in `ra.clt`, and exists only for historical reasons.

Finally, a `tcf` file can contain a `TitlePage` block: this is a description of how commands like `\maketitle` can be translated using meta-data (title, author, keywords, etc) defined earlier.

### 6.2 The command line arguments

If you call Tralics without arguments, you will see something like

```
This is tralics 2.11.7, a LaTeX to XML translator, running on macarthur
Copyright INRIA/MIAOU/APICS 2002-2008, Jos'e Grimm
Licensed under the CeCILL Free Software Licensing Agreement
Say tralics --help to get some help
```

In any case, the first three lines are printed. The version number may vary; we shall describe here the behavior of version 2.12 (released in April 2008). Command line arguments are read and interpreted from left to right. If an argument does not start with a hyphen, it is the name of the source file (only one input file is accepted); otherwise it is called an option. Some option names are shown with a hyphen, it is optional (in fact, dashes and underscores are ignored in option names), so that `-help` and `--help` are synonyms. Some options take no argument, for instance `-version` (whose effect is to print the version number and quit); others, for instance `-input-file`, take an argument. The argument is the character string that follows, preceded by an optional equals sign. Example

```

tralics -foo = bar gee      #1
tralics -foo= bar gee
tralics -foo =bar gee
tralics -foo=bar gee      #4
tralics -foo = "bar gee"  #5
tralics -foo = bar\ gee
tralics -foo bar\ gee
tralics -foo = " bar gee" #8
tralics -foo = \ bar\ gee
tralics -foo \ bar\ gee
tralics "-foo = bar gee"  #11
tralics -foo\ =\ bar\ gee\

```

We assume here that a command line interpreter (usually called a shell) reads the line you type, converts it in character strings, finds the executable program associated to the first string, and calls it with all these strings as arguments. There are five arguments on the first line (the first argument is the name of the program, it is currently ignored). We assume here that spaces can be inserted into an argument by either enclosing the string in quotes, or by escaping the space with a backslash, and that characters after a sharp sign are ignored. Assume that `-foo` is a `Tralics` option that takes a value; then the previous line are interpreted as follows.

The first three examples are similar but for spaces around the equals sign. Cases 1 and 4 are equivalent, the argument of `-foo` is `bar`, and there is a second option `gee`. In case 2, the argument of `-foo` is empty, and there are two options `bar`, `gee`. In case 3, the optional equals sign is omitted, hence the argument is `=bar`, and there is a second option. Thus you should either put no space or two spaces surrounding the equals sign.

Remaining examples show what happens if you put spaces in the argument. In cases 5, 6 and 7, the argument is `bar-space-gee`. In cases 8, 9, 10 it is `space-bar-space-gee`. Lines 11 and 12 are the same, except for the trailing space. Since `Tralics` removes spaces before and after the equals sign, the argument is `bar-space-gee` (plus space in the last case).

Here is the list of all options, in alphabetic order.

- `all` (Raweb only) is obsolete and not described anymore.
- `check` (Raweb only) can be used to check the document for the Raweb semantics. No translation is started. In 2007, we removed the preprocessor and its checks, and this option becomes an error it can only be used if the year of the document (that matches the document class) is 2006 at most. Option is ignored in 2008.
- `confdir=prefix` specifies an alternate location for configuration files (with extension `tcf`, `ult`, `plt`, and `clt`). If you specify A, B, and C, in this order, then `Tralics` checks C, B, A, D in this order (where D is a default value).
- `config=SomeFile`: configuration file is `SomeFile`. You can use `'config-file'` instead of `'config'`.

- **default-class=MyClass**. Assume that you compile a file that has `\documentclass{foo}`; in this case, Tralics tries to read the file `foo.clt`. No error is signaled if the file cannot be opened. If this option is used, then `MyClass.clt` is loaded if it exists; the command `\CurrentClass` will hold the name of the document class (here `foo`), so that `MyClass.clt` knows how it was called.
- **dir=path** (Raweb only); path will be the “raweb dir”, the directory containing lots of stuff for the Raweb mode. In particular, it contains a subdirectory `confdir`<sup>1</sup> with the configuration file. If this option is not used, the value of the shell variable `TRALICSDIR` will be used instead. Is obsolete in 2008, option `confdir` should be instead.
- **distinguish-refer-in-rabib=x** sets a special flag. The value can be `yes`, `no`, `true`, `false`, otherwise it is ignored. Default is `yes`. The flag indicates whether, for the Raweb, the refer bibliography database should be considered a normal file, or a subfile of the year database.
- **doc-type=A-B** This specifies the DOCTYPE of the resulting XML document. Here A is the name of the root element, B is the DTD file, these two names are separated by a space, a dash, a comma or a plus sign.
- **entnames=yes/no** applies to some commands like `\alpha` that translate as `'&alpha;'` or `'&#x3B1;'` (before version 2.9, it applied also to text commands like the euro sign, that are now represented internally as a Unicode character). Default is `yes`: the XML resulting document contains a name rather than a number.
- **etex** enables  $\epsilon$ -T<sub>E</sub>X extensions. This is the default. Option `noetex` disables these extensions.
- **external-prog=pgm** (Raweb only) tells Tralics to use `pgm` instead of `'rahandler.pl'` as interpreter for the Raweb actions defined by the `xmlXXX` switches.
- **hack-notitle** (Raweb only). Tralics may replace `\section{}` (sectioning command with empty argument) by `\section{Introduction}` (same command with a dummy name). This is implied in raweb mode until 2006. The option was withdrawn in 2007: using it has no effect on the translation. You should never use a sectioning command with an empty argument.
- **find-words** asks for printing on the file `words` the list of all words of the XML tree.
- **help** prints the list of all options.
- **input-dir=dir** specifies where source files are to be found. This is a colon separated list of directories. Current directory is defined by an empty slot, or a single dot; it will be added to the end of the list, unless present.
- **input-path=dir** is the same as `input-dir`.
- **input-file=foo** specifies the source to be `foo` (extension `.tex` is added if needed).
- **interactivebib**: withdrawn in version 2.9.
- **interactivemath**: special mode, where no input file is required, expressions are read from the terminal. The translation of every math expression is printed on the terminal. The value of `\jobname` will be `'texput'` (so that the XML result is in `texput.xml`, the transcript file is `texput.log`).
- **latin1**: source files are assumed to be latin1 encoded, unless the first line of a file contains `'utf8-encoded'` (if `'utf8'` and `'latin1'` are given, the last option wins, if none is given latin1 is assumed).

---

<sup>1</sup>Was `src` in previous versions

- `leftquote=num` specifies translation of the left quote (or backquote) to be character number `num`; see below under `rightquote`.
- `log-file = myfile`. This tells `Tralics` to put all messages and warnings in `myfile` (extension `.log` added if not given). The file can be in the directory specified by the `output-dir` option.
- `math-variant`: By default, or if you say ‘no-math-variant’, a math character in a font, for instance `\mathcal A` translates into `<mi>&#x1D49C;</mi>` or `<mi>&Ascr;</mi>`, a character from Unicode plane one. Using this option changes the translation to be an ASCII character with a math variant property, hence `\mathfrak B` gives `<mi mathvariant='fraktur'>B</mi>`.
- `no-bib-year-error` (Raweb only): no error is signaled for a bibliography reference with missing or wrong year.
- `no-bib-year-modify` (Raweb only): no special hacks are applied if an entry is wrongly marked ‘refer’ (otherwise, the entry is moved to another category)
- `no-config`: no configuration file will be used.
- `no-entnames` is the same as ‘`-entnames=no`’.
- `no-etex` disables  $\epsilon$ -TeX extensions.
- `no-mathml` sets the `\@nomathml` counter to minus one; default is zero.
- `no-math-variant` is the converse of ‘`mathvariant`’ described above.
- `no-trivialmath` sets the `\notrivialmath` counter to 0 (default is 1), thus even trivial math formulas such as `$0$` are translated as a `<formula>` element.
- `no-undef-mac`: no error is signaled when you use an undefined command like `\foo`. Instead, the command is automatically defined to be `\foo`.
- `no-straight-quotes`: the apostrophe translates into character U+B4, as `\textasciicute`. However, the normal value is used in verbatim mode, when reading a file name, in an URL, or in a construct like `\char`.
- `no-xml-error` inhibits insertion of `<error>` elements in the XML tree. By default, an error element containing (at least the current line number) is added to the tree, for every error.
- `no-zerowidth-elt` controls what should be inserted in a verbatim string in order to inhibit ligatures when converting XML to Pdf. For this purpose the zero-width space character, namely `&#x200B;`, is used. This character appears sometimes as a normal-width space in a HTML file. For this reason, an element `<zws/>` is used; it can be replaced by a style sheet according to its destination: a special marker for Pdf, nothing for HTML. The default translation is an element, but using the flag changes it to a character.
- `no-zerowidth-space` inhibits insertion of a special marker in a case like `\verb+--+`.
- `oe8`, `oe1`, `oe8a`, or `oe1a` specify the output encoding (used in the XML file), one of UTF8 or latin1. If the letter a is given, then all non-7 bits characters are printed as character references. Thus, the only difference between option `oe8a` and `oe1a` is the XML header line. Default encoding is `oe1`.
- `output-dir = mydir` specifies the directory in which output files are to be stored (this concerns the main XML file, the transcript file, and other files).

- `output-file = myfile` tells Tralics to put the result in `myfile` (extension `.xml` added if not given). In the case of the Raweb, this option is ignored.
- `param = foo=bar`: `'foo="bar"'` is virtually added at the end of the configuration file. The equals sign can be replaced by a space.
- `ps` (Raweb only): Tralics creates a file `'foo.tex'` from the file `'foo2006.tex'`, calls L<sup>A</sup>T<sub>E</sub>X, then `dvips` in order to produce a PostScript file. The document class of the new file is `'raweb.cls'` instead of `'raweb2006.cls'`. This option exists for compatibility, and may be withdrawn. This is the only case where no XML file is created.
- `ra-debug` (Raweb only): parsing of the Raweb source continues after the first error.
- `rightquote=num` defines the translation of the right quote (or apostrophe). The problem is the following: the default translation of the left and right quote are often wrong; but you cannot make these characters active, because the scanner assumes that they are of category 12. The behaviour is the following: in verbatim mode, (as well as in URLs) these characters behave normally; if doubled translation is U+201C and U+201D. Otherwise you can change the translation. If you say `-leftquote=2018` and `-rightquote=2019` then characters U+2018 and U+2019 are used. Only base16 digits are allowed; the value should be a number between 1 and 2<sup>16</sup> (otherwise default value is used).
- `shell-escape` makes `\write18{rm \jobname.tex}` remove the file you are translating.
- `silent` make the program silent (less lines are printed on the terminal).
- `te8`, `te1`, `te8a`, or `te1a` specify the encoding used in transcript files. In the case of `te8` or `te8a`, characters are printed using UTF-8 format; in the case of `te1` or `te1a`, characters are printed using latin1 encoding. Characters are printed using the `^^^abcd` notation in case: the value is greater than 255, and one of `-te8a`, `-te1a` is given, or the character is not in proper range (32-126 plus 160-155) and `te1` is given, the character is smaller than 32. Note: horizontal tabulation, line-feed and carriage-return do not use the hat-hat notation. Default value is the output encoding (option `oe8`, etc.).
- `tpa-status=flag` controls what is to be translated if the configuration file specifies a titlepage (see for instance the `'titlepage.html'` document on the Web). If the value is `'all'`, then the whole document is translated; if the value is `'title'`, only the titlepage is translated; if the value is `'config'`, action depends on the configuration file. Otherwise, translation stops in case of an error, continues otherwise. Only the first character of the value is tested. Capital letters are allowed.
- `trivialmath=N` sets the `\notrivialmath` counter to N (default is 1). Thus trivial math formulas such as `$_alpha$` are translated as non-math.
- `type=mytype` specifies the configuration file to use. If you say `'tralics -type ra hello'`, this will read the `ra.tcf` file, and you will enter Raweb mode, and this is wrong because the name of the file contains no year. If you say `'tralics -type article miaou2003'`, you are not in Raweb mode, the preprocessor is not called, and 157 strange errors are signaled because of different reasons. If you say `'tralics -type ra miaou2007'`, the compilation will fail because `ra.tcf` was designed for year 2006 and `ra2007.tcf` should be used instead. If you say `'tralics -type ra2007 miaou2003'`, compilation fails because the `tcf` file contains an instruction declaring `\declaretopics` to be ignored, and this declaration is declared illegal by the raweb checker.
- `use-quotes` has as effect to convert double quotes into a pair of single quotes, either left quotes, or right quotes, that in turn can be replaced by other characters (for instance French guillemets).

- `utf8`: sources files are assumed to be UTF-8 encoded, unless the first line of a file contains ‘iso-8859-1’.
- `utf8output` is equivalent to options ‘oe8’ and ‘te8’.
- `v` and `verbose` make the program rather verbose (it prints a lot of lines in the transcript file). The equivalent of the command `\tracingall` is executed.
- `V` or `verbose-doc`: the program is rather verbose (see option `v` above), but only when `\begin{document}` is seen. The equivalent of the command `\tracingall` is then executed.
- `version`: the program stops after printing the banner.
- `xml`, `xmlfo`, `xmlhtml`, `xmllint`, `xmltex`, `xmlall` (Raweb only). These options are incompatible with ‘check’ and ‘ps’. They ask Tralics to produce an XML file, and maybe more (a HTML file in the case `xmlhtml`, a FO file in the case `xmlfo`, a postscript file in the case `xmltex`) and check the validity of the translation against the Raweb DTD (case `xmllint`).
- `year=nb` (Raweb only) specifies the default year (year 2005 starts at May, 1st, because people start writing the Raweb2004 in September 2004, and everything should be finished by March 2005).

Example. Assume that we have a file, named `xii.tex`, containing

```
\let~\catcode~'76~'A13~'F1~'j00~'P2jdefA71F~'7113jdefPALLF
PA'~'FwPA; ;FPAZZFLaLPA//71F71iPAHHFLPAzzFenPASSFthP;A$$FevP
A@@FfPARR717273F737271P;ADDFRgniPAWW71FPATTFvePA**FstRsamP
AGGFRruoPAqq71.72.F717271PAY7172F727171PA??Fi*LmPA&&71jfi
Fjfi71PAVVFjbigskipRPWGAUU71727374 75,76Fjpar71727375Djifx
:76jelse&U76jfiPLAKK7172F7117271PAXX71FVln0SeL71SLRyadR@oL
RrhC?yLRurtKFeLPFovPgaTLtReRomL;PABB71 72,73:Fjif.73.jelse
B73:jfiXF71PU71 72,73:Pws;AMM71F71diPAJJFRdriPAQQFRsreLPAI
I71Fo71dPA!!FRgiePbt'el@ 1TLqdrYmu.Q.,Ke;vz vzLqip.Q.,tz;
;Lql.IrsZ.eap,qn.i. i.eLlMaesLdRcna,; ;!;h htLqm.MRasZ.ilK,%
s$;z zLqs'.ansZ.Ymi,/sx ;LYegseZRyal,@i;@ TLRlogdLrDsW,@;G
LcYlaDLbJsw,SWXJW ree @rzchLhzw,;WERcesInW qt.'oL.Rtrul;e
doTsW,Wk;Rri@stW aHAHHFndZPppar.tridgeLinZpe.LtYer.W,:jbye
```

If you call Tralics, with the option ‘find-words’, you can see that the XML file contains once drumming and drummers, twice piping and pipers, 3 times leaping and lords, 4 times dancing and ladies, 5 times milking and maids, 6 times swimming and swans, 7 times laying and geese, 8 times rings and gold, 9 times calling and birds, 10 times hens and french, 11 times doves, turtle and ‘and’, 12 times tree, pear, in, partridge, me, to, gave, love, true, my, christmas, of, day, the, on. There are 45 words with a single letter. The words: twelve, eleven, ten, nine, eight, seven, six, five, four, three, two, appear  $x$  times, where  $13 - x$  is the value of the word. The words first, second, third, fourth, fifth, sixth, seventh, eighth, ninth, tenth, eleventh, twelfth appear once. Amazing isn’t it? The file was written by D. Carlisle, it is available on the CTAN. This is not really a  $\LaTeX$  file, so that some features cannot be applied (for instance, there is no at-begin-document hook). The `\bye` command was implemented in Tralics for this example to compile without error.

### 6.3 Configuration files

A configuration file is a way to alter the translation, using a special syntax. It contains some rules that define a *Type* and some *tcf* blocks, where a *tcf* block is identical to the content of a *tcf* file, and a *Type* is the name of a *tcf* file (*tcf* stands for Tralics configuration file). The *Type* can be

given as a command line argument, or in the main source, provided that the following magic line appears near the beginning of the document (the tcf file name is between quotes):

```
% Tralics configuration file 'test0.tcf'
```

A tcf file may contain some blocks: for instance, a TitlePage block, described later, or a Command block, that contains L<sup>A</sup>T<sub>E</sub>X commands inserted at the start of the document; it contains also assignments of various types. In particular, it contains the Document Type used in the XML output. As already mentioned, the Document Type information can be given as a command line argument; it may also be given in the main source file, if a magic like the following appears near the start of the document (the DTD is classes.dtd, with <book> as root element):

```
% Tralics DOCTYPE = book classes.dtd
```

We explain here the default configuration file (that has little use anymore), the old default configuration file (in use before 2006), the tcf file for the Raweb, a tcf and plt file for Research Reports (we will show how the same document can be compiled in two different ways).

### 6.3.1 The standard configuration file

We give here the content of the standard configuration file. As you can see, there are lots of comments. There is one assignment, this is a rule that says that the Type to use is the document class of the input file. There is a block that says that if this Type is report, book, article, minimal, and if no tcf file is found for this Type, then std.tcf should be used instead; this block says also that torture1 and torture2 are aliases for torture (used only for debugging). Finally, there is a block defining std.tcf: it says that the Doctype to use is classes.dtd, and the root element is <std>.

Lines 2 and 3 were modified: we added the letter 'x' after the dollar sign, for otherwise RCS would replace the identification of the original file by the identification of the L<sup>A</sup>T<sub>E</sub>X file.

```
# This is a configuration file for tralics.
# $xId: tralics_rc,v 2.24 2006/07/24 08:23:17 grimm Exp $
## tralics ident rc=standard $xRevision: 2.24 $

# Copyright Inria. Jos\`e Grimm. Apics. 2004/2005 2006
# This file is part of Tralics.

% Some comments: comments start with % or #

% this means: take the documentclass value as type name
Type = \documentclass

## First type defined is the default. Since version 2.8, there is only
## one type.

BeginType std#          standard latex classes
  DocType = std classes.dtd
End

BeginAlias
  torture torture1 torture2
  std report book article minimal
End
```

Comments: If no configuration file is found, default rules apply. In particular, the default Type is the document class, and if no tcf file is found, the DocType to use is unknown from unknown.dtd,

unless it is a standard L<sup>A</sup>T<sub>E</sub>X class, case where `std` from `classes.dtd` is used. This means that the standard configuration file has become useless.

### 6.3.2 The old configuration file

We shall describe here the old configuration file, used before the notion of `tcf` files was invented.

Lines starting with a sharp sign or percent sign are comment lines (ignored). Some lines start with ‘Begin’, and others with ‘End’. To each Begin, there should be an associated End. Blocks can be nested. Characters after ‘End’ are ignored, so that you can say ‘BeginFoo’ followed by ‘EndBar’, although it is not recommended. All other lines should be comment lines, empty, or indented.

Note the ‘x’ after the dollar sign; it does not appear in the source file, (see comment in the previous subsection). The third line is a bit special: when `Tralics` loads the file, it prints the revision number on the terminal.

```

1 # This is a configuration file for tralics.
2 # $xId: tralics_rc,v 2.15 2005/08/02 09:22:56 grimm Exp $
3 ## tralics ident rc=standard $xRevision: 2.15 $

```

This is the Copyright notice. In the current version, the semantics of the RA is in the `ra.tcf` file (described later).

```

4 # Copyright Inria. José Grimm. Apics. 2004/2005
5 # This file is part of Tralics.
6 # (See the file COPYING in the Tralics main directory)
7 # If you modify this file, by changing the semantics of type RA,
8 # please remove the ‘standard’ on the ‘tralics ident’ line above,
9 # or replace it by ‘non-standard’.
10
11 % Some comments: comments start with % or #

```

A configuration file is split into main sections, one for each type. We start with the RA, or `raweb`.

```

12 ## configuration for the RA (Inria’s Activity Report)
13 ## First type defined is the default
14
15 BeginType RA      % Case RA
16   DocType = raweb raweb3.dtd
17   DocAttrib = year \specialyear
18   DocAttrib = creator \tralics

```

This comment explains how to parametrize some element or attribute names that were built-in in a previous version of `Tralics`. We shall see later how `Language` can be used (default value is ‘`language`’), the same for `lang_en` and `lang_fr` that have ‘`english`’ and ‘`french`’ as default value. Translation of a `\caption` produces an element `<caption>`, whose name will be changed to `<head>` by the post-processor of figures (it will be left unchanged if the caption is not in a figure or a table). The variable `xml_caption_name` can be used to change the first name, and `xml_scaption_name` can be used to change the second name. The title of a ‘`topics`’ (defined by `\declaretopics`) is in a `<t_title>` element, the name can be changed by `xml_topic_title`. A reference to a topic uses the `num` attribute; this attribute name can be changed by `att_topic_num`. The identification of an Inria Team is in `<accueil>`, this can be changed via `xml_accueil_name`. It is formed of a long name in `<projetdeveloppe>` and a short name in `<projet>`, the name of these elements can be changed via `xml_project_name` or the ‘`expanded`’ version. The section with the composition of the team is `<composition>`, its name can be changed by `xml_composition_ra_name`.



```

19 # (new)
20 # Language = "xml:lang"
21 # lang_en = "en"
22 # xml_scaption_name= "caption"
23 # xml_topic_title=""
24 # xml_project_name = "title"
25 # xml_expanded_project_name = "longtitle"
26 # xml_accueil_name = "identification"
27 # xml_composition_ra_name = "team"
28 # att_topic_num = "id"

```

Processing of the Raweb needs converting the XML output of Tralics into XSL/Format, HTML, etc., via some external commands like ‘xsltproc’, ‘latex’, etc. Originally, Tralics was in charge of these commands, and the configuration file explains how to call these tools. These lines are not needed anymore.

```

29 makefo="xsltproc --catalogs -o %A.fo %B %C";
30 makehtml = "xsltproc --catalogs %B %C";
31 call_lint = "xmllint --catalogs --valid --noout %C"
32 makepdf = "pdflatex -interaction=nonstopmode %w"
33 %makedvi = "latex -interaction=nonstopmode %w"
34 % makedvi et dvips pour marie-pierre
35 %dvitops = "dvips %w.dvi -o %w.ps"
36 %makedvi = "latex -interaction=nonstopmode %w"
37 generatedvi = "latex -interaction=nonstopmode %t"
38 % old latex: "latex \nonstopmode\input{%t}"
39 generateps = "dvips %t.dvi -o %t.ps"

```

This defines the list of valid Raweb sections, themes and URs (research units). If you change these lines please: a) remove the ‘standard’ on line 3, or b) make sure that it matches the official list, or c) make sure that this remains a private copy. A star after a section name says that topics are not allowed<sup>2</sup>.

```

40 #these are new in version 2.0
41 theme_vals = "com cog num sym bio"
42 section_vals = "composition*/presentation*/fondements/domaine/logiciels/"
43 section_vals = "+resultats/contrats*/international*/diffusion*/"
44 ur_vals = "Rocquencourt//Sophia/Sophia Antipolis/Rennes//Lorraine//";
45 ur_vals = "+RhoneAlpes/Rhône-Alpes/Futurs//"

```

Due to some inertia, people continue using the obsolete environment. We make sure an error is signaled.

```

46 BeginCommands
47 \newenvironment{body}{\obsoleteEnvBody The body environment is %
48   obsolete since 2003}
49 {End of obsolete environment body}
50 \newenvironment{abstract}{\obsoleteEnvAbstract The abstract %
51   environment is obsolete since 2003}
52 {End of obsolete environment abstract}
53 EndCommands
54
55 End

```

This is an example of titlepage environment; it will be discussed later. In fact, we shall give below the content of the RR.tcf file, it is identical.

<sup>2</sup>This is new in version 2.5 (pl7)

```

56 ## configuration for the RR (Research Report of Inria)
57 ## not yet complete
58
59 BeginType RR#      Case RR
60 ...
89 EndType
    A short definition for standard classes.
90 BeginType std#      standard latex classes
91   DocType = std classes.dtd
92   xml_biblio = "bibliography"
93 End
    Some aliases.
94 # (types Article and slides are not defined, hence this is useless)
95
96 BeginAlias
97   Article report
98   slides inriaslides foiltex
99 End
    This command has to be outside any block.
100 % this means: take the documentclass value as type name
101 Type = \documentclass
    More aliases. Note that toto matches RR (first in list) and report matches std (because 'un-
    known' is undefined).
105 BeginAlias
106   RR toto# ra2001
107   RA ra toto ra2001x%/etc
108   torture torture1 torture2
109   unknown report
110   std report book article minimal
111 End
    For fun.
112 ## an empty type
113 BeginType MP
114 EndType
    This is used for testing Tralics.
115 BeginType torture
116   DocAttrib = creator \tralics
117   DocType = remain raweb.dtd
118   on package loaded calc CALC = "true"
119   on package loaded foo/bar F001 = "true"
120   on package loaded *foo/bar F002 = "true"
121   on package loaded foo/*bar F003 = "true"
122   on package loaded *foo/*bar F004 = "true"
123   url_font = "\large "
124   no_footnote_hack = "false"
125   on class loaded calc CALC="true"
126

```

```

127 use_font_elt = "true"
128 xml_font_small = "font-small"
    A bunch of declarations omitted here. The list of all options is given later, in test.tcf.
154 xml_underline_name = "font-underline"
155
156 BeginCommands
157   % These commands are inserted verbatim in the file
158   \def\recurse{\recurse\recurse}
159 EndCommands
160 EndType
    This may be used for typesetting a bibliography, exactly like the Raweb.
161 BeginType rabib   % Case RA
162   DocType = raweb raweb3.dtd
163   DocAttrib = year \specialyear
164   DocAttrib = creator \tralics
165
166 BeginCommands
167   % These commands are inserted verbatim in the file
168   \newcommand\usebib[2]{\bibliography{#1#2,#1_foot#2+foot,#1_refer#2+refer}}
169 EndCommands
170 EndType

```

### 6.3.3 The ra.tcf file

This is the tcf file used for the Raweb2006. Read carefully the copyright notice.

```

1 # This is a configuration file for tralics, for the Raweb
2 # $xId: ra.tcf,v 2.3 2006/07/25 16:29:39 grimm Exp $
3 ## tralics ident rc=standard-ra $xRevision: 2.3 $
4
5
6 # This file is part of Tralics.
7 # Copyright Inria. Jos'e Grimm. Apics. 2004/2005, 2006
8 # (See the file COPYING in the Tralics main directory)
9 # If you modify this file, by changing the semantics of the RA,
10 # please remove the 'standard-ra' on the 'tralics ident' line above,
11 # or replace it by 'non-standard'.

```

A tcf file is a like the configuration file, but it applies to a single type of document; for this reason, there is no need to explain how the type is computed (no 'Type' declaration), neither to what type a block applies (there is no 'BeginType' block). These three lines are the same as before. Note that the 2007 DTD is raweb7.dtd.

```

12 DocType = raweb raweb3.dtd
13 DocAttrib = year \specialyear
14 DocAttrib = creator \tralics
    These line are as before, without commented out lines.
15 makefo="xsltproc --catalogs -o %A.fo %B %C";
16 makehtml = "xsltproc --catalogs %B %C";
17 call_lint = "xmllint --catalogs --valid --noout %C"
18 makepdf = "pdflatex -interaction=nonstopmode %w"

```

```

19 generatedvi = "latex -interaction=nonstopmode %t"
20 generateeps = "dvips %t.dvi -o %t.ps"

```

This values are the same as those shown above.

```

21 theme_vals = "com cog num sym bio"
22 ur_vals = "Rocquencourt//Sophia/Sophia Antipolis/Rennes//Lorraine//";
23 ur_vals = "+RhoneAlpes/Rh\^one-Alpes/Futurs//"

```

In 2006, section\_vals has the same value as shown above; in 2007 it is replaced by the following lines.

```

24 fullsection_vals = "/composition/Team/presentation/Overall Objectives/\
25 fondements/Scientific Foundations/domaine/Application Domains/\
26 logiciels/Software/resultats/New Results/\
27 contrats/Contracts and Grants with Industry/\
28 international/Other Grants and Activities/diffusion/Dissemination"

```

New in 2006 are the two lists affiliation\_vals and profession\_vals. The syntax is the same as for other lists. The value given here is an example; the real names are in French.

```

29 affiliation_vals = "Inria//Cnrs//University//ForeignUniversity//"
30 affiliation_vals = "+Public//Other//"
31 profession_vals = "Scientist//Assistant//Technical//PHD//"
32 profession_vals = "+PostDoc//StudentIntern//Other//"

```

We have the same obsolete environments as before. Moreover, we declare that `\keywords` is the same as `\motscle`; this is needed because we removed the `\keywords` command (for the Raweb, this is an environment, using it as a command will fail in a very strange manner).

```

33 BeginCommands
34 \let\keywords\motscle
35 \newenvironment{body}{\obsoleteEnvBody The body environment is %
36     obsolete since 2003}
37     {End of obsolete environment body}
38 \newenvironment{abstract}{\obsoleteEnvAbstract The abstract %
39     environment is obsolete since 2003}
40     {End of obsolete environment abstract}
41 EndCommands

```

This is the command block for the ra2007. The last line does not appear in the file, but is automatically added in Raweb mode; the command uses the values saved by `\theme`, `\UR` and its aliases, `\project` and its alias, `\isproject`. Some are defined as doing nothing (like `\maketitle`, `\loadbiblio`, `\declaretopic`, `\TeamHasHdr`). The `\module` command is redefined: if the last argument is empty, a default value is used instead.

```

42 BeginCommands
43 \makeatletter
44 \def\declaretopic#1#2{} %% obsolete in 2007
45 \def\TeamHasHdr#1{} %% temporary
46 \def\theme#1{\def\ra@theme{#1}}
47 \def\UR#1{\def\ra@UR{#1}}
48 \def\isproject#1{\def\ra@isproject{#1}}
49 \let\ResearchCenterList\UR
50 \let\ResearchCentreList\UR
51 \def\projet#1#2#3{\def\ra@proj@a{#1}\def\ra@proj@b{#2}\def\ra@proj@c{#3}}
52 \let\project\projet
53 \def\moduleref#1#2#3{\ref{mod:#3}}
54 \let\oldmodule\module %% Compatibility

```

```

55 \renewcommand\module[4] [] {\oldmodule{#2}{#3}{\@ifbempty{#4}{(Sans Titre)}{#4}}
56 \let\htmladdnormallinkfoot\@href@foot
57 \let\htmladdnormallink\@href
58 \makeatother
59 \let\maketitle\relax
60 \let\loadbiblio\relax
61 \let\keywords\motscle
62 %%% \AtBeginDocument{\rawebstartdocument} %%% pseudo line
63 EndCommands

```

New in 2008 is the following list. The argument of the `catperso` environment must be one of `XX`, `YY`, `ZZ`, interpreted as `xx`, `YY` and `zz`. If the declaration is omitted, there is no restriction on the argument. Whether or not there will be such a restriction in the file `ra2008.tcf` is still undecided.

```

64 catperso_vals = "XX/xx/YY//ZZ/zz"

```

### 6.3.4 The RR.tcf file

We indicate here the content of the `RR.tcf` file, it defines commands for the title page of a Research Report.

```

1 ## tralics ident rc=RR.tcf $Revision: 1.29 $
2 ## configuration for the RR (Research Report of Inria)
3
4
5 DocType = rr raweb.dtd
6 BeginTitlePage
7 \makeRR <RRstart> "" "type = 'RR'"
8 alias \makeRT "" "type = 'RT'"
9
10 <UR> -
11 \URSophia ?+<UR>
12 \URRocquencourt ?+<UR>
13 alias \URRocq
14 \Paris ?<UR> <Rocquencourt>
15 \URRhôneAlpes ?+<UR>
16 \URRennes ?+<UR>
17 \URLorraine ?+<UR>
18 \URFuturs ?+<UR>
19
20 \RRtitle q<title> "pas de titre"
21 \RRetitle q<etitle> "no title"
22 \RRprojet <projet> "pas de projet"
23 \motcle <motcle> "pas de motcle"
24 \keyword <keyword> "no keywords"
25 \RRresume p<resume> "pas de resume"
26 \RRabstract p<abstract> "no abstract"
27 \RRauthor + <author> <auth> "Pas d'auteurs"
28 \RRdate <date> "\monthyearvalfr"
29 \RRNo <RRnumber> "?????"
30
31 \RRtheme <> +"pas de theme" % CES
32 <Theme> - % E

```

```

33   \THNum ?+<Theme>           % CE
34   \THCog ?+<Theme>          % CE
35   \THCom ?+<Theme>          % CE
36   \THBio ?+<Theme>          % CE
37   \THSym ?+<Theme>          % CE
38
39   %% \myself \RRauthor "grimm" % CCS
40   %% \cmdp <cmdp> +"nodefault" % CES
41   %% \cmda <cmdA> A"\cmdAval" % CES
42   %% \cmdb <cmdB> B"\cmdBval" % CES
43   %% \cmdc <cmdC> C"\cmdCval" % CES
44
45   End
46
47   BeginCommands
48     \let\RRstyisuseful\relax
49   End

```

### 6.3.5 The RR.plt file

We indicate here the content of the RR.plt file, it also defines commands for the title page of a Research Report. This is a T<sub>E</sub>X file, loaded whenever the package ‘RR’ is used. Note that, if the RR.tcf file is loaded, the line 48 above defined a command that is checked on line 4 below, so that the file is ignored. We shall explain later how these two files can be used.

```

1  % -*- latex -*-
2  \ProvidesPackage{RR}[2006/10/03 v1.1 Inria RR for Tralics]
3
4  \ifx\RRstyisuseful\relax\endinput\fi
5
6  \newcommand\RRtitle[1]{\let\\\ \xbox{ftitle}{#1}}
7  \newcommand\RRetitle[1]{\let\\\ \xbox{title}{#1}}
8  \newcommand\RRauthor[1]{\xbox{author}{#1}}
9  \newcommand\RRprojet[1]{\xbox{inria-team}{#1}}
10 \newcommand\RRdate[1]{\xbox{date}{#1}}
11 \newcommand\RRNo[1]{\xbox{rrnumber}{#1}}
12 \newcommand\RRtheme[1]{\xbox{theme}{#1}}
13 \newcommand\keyword[1]{\xbox{keyword}{#1}}
14 \newcommand\motcle[1]{\xbox{motcle}{#1}}
15 \newcommand\THNum{THnum}
16 \newcommand\THCom{THcom}
17 \newcommand\THCog{THcog}
18 \newcommand\THSym{THsym}
19 \newcommand\THBio{THbio}
20 \newcommand\URSophia{\xbox{location}{Sophia Antipolis}}
21 \newcommand\URLorraine{\xbox{location}{Lorraine}}
22 \newcommand\URRennes{\xbox{location}{Rennes}}
23 \newcommand\URRhôneAlpes{\xbox{location}{Rhône-Alpes}}
24 \newcommand\URRocq{\xbox{location}{Rocquencourt}}
25 \newcommand\URFuturs{\xbox{location}{Futurs}}
26 \newcommand\RRresume[1]{\begin{xml element*}{resume}#1\end{xml element*}}
27 \newcommand\RRabstract[1]{\begin{xml element*}{abstract}#1\end{xml element*}}

```

```

28
29 \let\makeRT\relax
30 \let\makeRR\relax

```

### 6.3.6 Sample files

The Tralics distribution comes with a bunch of test files. There are two directories: the Test directory contains source files, and the Modele directory contains the translation. In particular, the file tpa2.tex explains how to use a tcf file to change the names of most XML elements.

## 6.4 The action before translation

As explained at the start of the Chapter, Tralics first reads all options. Some of them are marked ‘Raweb only’; this means that they are not used, unless the Type is ra (i.e. you are translating the Raweb, see next section); this section describes how the Type is computed.

Unless Tralics is called with option `interactive-math`, an input file name is required. The program is aborted if more than one input name is given. It must be the name of a  $\TeX$  file: an extension `tex` is added if needed, so that `foo` and `foo.tex` are the same. As an exception `foo.xml` is also equivalent to `foo.tex`. We consider two examples, the `xii.tex` shown above, and the following  $\LaTeX$  file `hello1.tex`:

```

\documentclass{article}
\begin{document}
Hello, world!
\end{document}

```

### 6.4.1 Files and Paths

The standard way to use Tralics is to type ‘tralics filename’ in a terminal, example:

```

grimm@macarthur$ tralics hello1
This is tralics 2.12, a LaTeX to XML translator, running on macarthur
...
Output written on hello1.xml (179 bytes).
No error found.
(For more information, see transcript file hello1.log)
grimm@macarthur$ ls hello1*
hello1.log      hello1.tex      hello1.xml

```

The `ls` command shows the source, the result of the translation and the transcript file. If the file `hello1.ult` were present, it would be been read by Tralics, and if the source were a bit more complicated the files `hello1.img` and `hello1_.bbl` might have been created. All these files are in the same directory, and this paragraph explains what you can do if input or output files are elsewhere.

Consider now a graphical interface to Tralics, where you drag and drop the  $\TeX$  source; in such a case there is no shell anymore, hence no current directory; what Tralics gets is an absolute path name (that may be of the form `/users/somebody/somewhere`). In early versions, such an absolute path was a fatal error. Currently, only Unix-like pathnames are implemented.

Consider now a system, like the Raweb, where the XML file produced by Tralics is converted to another XML file (with a different DTD), and further processed. Thus a great number of files are created, and managing all these becomes uneasy. As the example below shows, you can ask Tralics to put the files it creates in another directory, you can chose the name of the XML output

(so that `Tralics` can create `foo-t.xml` from `foo.tex`, and this file can be processed again into `foo.xml`), and you can also chose the name of the transcript file.

```
grimm@macarthur$ tralics hello1 -o h2 -logfile=h3 -output_dir=./Test
This is tralics 2.12, a LaTeX to XML translator, running on macarthur
...
Output written on ../Test/h2.xml (179 bytes).
No error found.
(For more information, see transcript file ../Test/h3.log)
```

The input path is a colon separated list of directories. For instance `../foo/A:/bar/B/::gee:` contains five elements, two of them being empty. The empty slot represents the current directory, it will be added at the end if omitted. The current directory may also be given as a single dot. A final slash is silently removed. In this example, the path means: search in subdirectory A of the sibling directory foo, then if the subdirectory B of the directory bar that is is at the root, then in the current directory, then in the subdirectory gee of the current directory, then in the current directory again; this rule does not apply if a file starts with a dot or a slash.

A special case is when the main input file name starts with a dot or a slash, for instance `/usr/grimm/home/hello` or `./Test/hello.tex`. In this case, the name is split into pieces. One piece is the entry name, say `hello`, another one is the directory name (everything before the final slash), and the last part is the extension (here `.tex`). If no output directory is given on the command line, the directory of the input file is used. In the same fashion

You can also specify the name of the transcript file; By default, this is the entry name. If for instance you use `/foo/bar`, then input file will be `/foo/bar.tex` and the transcript file will be `/foo/bar.log`; you may change the name of the transcript file, so as to get `/foo/myfile.log`; you may change the directory of the transcript file, so as to get `/mydir/bar.log`; you may change both.

Consider again the case where the input is `/foo/bar`. If no input path is given, then `Tralics` behaves as if the file was `bar`, and the input path was `'foo:'`. This has as consequence that, if `bar` inputs another file, say `bar1`, it is first searched in the same directory as `bar`, and then in the current directory. Moreover, if no output directory is specified, files written by `bar` are put in this directory, thus can be read again. If the user gives an input path, it will be left unchanged, and the input path is not considered for the main path. Example: Directories `foo` and `foo1` contain files `bar` and `bar1`; `bar` inputs `bar1`, input path contains `foo1`. If the main file is `/foo/bar`, it will input `/foo1/bar1`. If the input path contains both `foo` and `foo1`, and the main file is `bar`, you will get either `/foo/bar` and `/foo/bar1` or `/foo1/bar` and `/foo1/bar1`, depending on the order.

## 6.4.2 Finding the configuration

There are some options that tell `Tralics` not to produce an XML file, we shall not explain them. Thus, after parsing all arguments, `Tralics` reads the complete source (main input file). It opens the transcript file, and print on the terminal a line like *Starting translation of file hello1.tex*. The transcript file will contain a bit more information, namely

```
Transcript file of tralics 2.12 for file hello1.tex
Copyright INRIA/MIAOU/APICS 2002-2008, Jos'e Grimm
Tralics is licensed under the CeCILL Free Software Licensing Agreement
Start compilation: 2008/04/19 18:27:18
OS: Apple, machine macarthur
Starting translation of file hello1.tex.
Using iso-8859-1 encoding (idem transcript).
Left quote is ' right quote is '
Input path (../FO:../Test:)
++ Input encoding is 1 for ../Test/hello1.tex
```



After that, Tralics reads the configuration file. You can use the `-noconfig` option, this inhibits reading a configuration file. In this case the transcript file contains

```
No configuration file.
No type in configuration file
Seen \documentclass article
Potential type is article
Using some default type
dtd is std from classes.dtd (standard mode)
OK with the configuration file, dealing with the TeX file...
```

The first line says that no configuration file is considered, so that an empty one will be used instead. The  $\text{\TeX}$  source is scanned for a document class. If this is a standard one (book, article, report, minimal, the DTD is std from classes.dtd, otherwise unknown from unknown. Consider now the same file, without the `-noconfig` option. We get

```
++ file .tralics_rc does not exist.
++ file ../confdir/.tralics_rc exists.
Configuration file identification: standard $ Revision: 2.24 $
Read configuration file ../confdir/.tralics_rc.
Configuration file has type \documentclass
Seen \documentclass article
Potential type is article
Defined type: std
++ file article.tcf does not exist.
++ file ../confdir/article.tcf does not exist.
Alias torture does not match article
Potential type article aliased to std
Using type std
dtd is std from classes.dtd (standard mode)
```

There are some lines starting with a double plus sign. Whenever Tralics searches if a file exists, it will print such in line in the transcript file. The first two lines that do not start with a double plus are also printed on the terminal (this is an easy way to check that that right configuration file has been seen). The standard configuration file says that they Type is the document class (here article). This is a true type, provided that it is defined, and the configuration file does not define it. It could be defined in `article.tcf`. But you can see that there is no such file. As a consequence, the behavior is the same as if no configuration file has been given.

This is what happens if the option `config=rabib` is given

```
Trying config file from user specs: rabib.tcf
++ file ../confdir/rabib.tcf exists.
Configuration file identification: rabib.tcf $ Revision: 2.2 $
Read configuration file ../confdir/rabib.tcf.
Using tcf type rabib
dtd is raweb from raweb3.dtd (standard mode)
```

You can notice that a tcf file is being searched in the `confdir` directory. If the name starts with a slash or a dot, no extension is added, and the file is not searched in the configuration path. Assume that the source file contains a line of the form

```
% Tralics configuration file 'test0.tcf'
```

and you neither specify a configuration file, nor inhibit loading one. Then you will get

```
Trying config file from source file 'test0'
++ file test0.tcf does not exist.
++ file ../confdir/test0.tcf exists.
```

```

Read configuration file ../confdir/test0.tcf.
Using tcf type test
catperso_vals: AA -> BB
catperso_vals: CC -> CC
catperso_vals: XX -> xx
dtd is unknown from unknown.dtd (standard mode)

```

As you can see, tcf extension is added, and the file is searched in the current directory first, then in the configuration path.

You can call Tralics with option `type=rabib`. This just says that the name of the tcf file should be rabib, instead of the document class; it is thus useless if the name of the tcf file to use has been given as shown above. It can be useful in the case of a plain T<sub>E</sub>X file, that has no document class. In the example that follows, we say that the type is ra12.

```

++ file .tralics_rc does not exist.
++ file ../confdir/.tralics_rc exists.
Configuration file identification: standard $ Revision: 2.24 $
Read configuration file ../confdir/.tralics_rc.
Configuration file has type \documentclass
Seen \documentclass article
Potential type is ra12
++ file ra12.tcf does not exist.
++ file ../confdir/ra12.tcf does not exist.
++ file ra.tcf does not exist.
++ file ../confdir/ra.tcf exists.
Configuration file identification: standard-ra $ Revision: 2.3 $
Read tcf file ../confdir/ra.tcf
Using type ra
...
dtd is raweb from raweb3.dtd (mode RAWEB2007)

```

Note that no file ra12.tcf was found, and ra.tcf was used searched for. As a consequence, the effective type is ra, and Raweb mode is entered; this is an error, since current file is not a ra file. In fact, you will be faced to *Fatal error: Input file name must be team name followed by 2007*. Note that you can compile a file named foo2006 in Raweb mode, as long as this matches the year option (if used) and the document class is ra2006.

### 6.4.3 Old behaviour

The algorithm is the following.

1. If you say `tralics -noconfig`, then no configuration file is read at all.
2. If you say `tralics -configfile=foo`, then Tralics will print *Trying config file from user specs*, and try to use this file.
3. If you say `tralics -configfile=foo.tcf`, then Tralics will print the same as above; it will also search the file in the ‘confdir’ directory.
4. If the source file contains ‘% tralics configuration file ‘foobar’’, then Tralics will print *Trying config file from source file*, and try to use this file. In case of failure, and if the name ‘foobar’ contains no dot, the suffix .tcf is added, and the next rule is applied.
5. If the source file contains ‘% tralics configuration file ‘foobar.tcf’’, then Tralics will print the same as above; it will also search the file in the ‘confdir’ directory.

6. The default configuration file is named `.tralics_rc` (or `tralics_rc` on Windows). The current directory is looked at first, then the `tralicsdir`, finally the home directory.
7. If you say `tralics -dir TOTO`, then `TOTO/src/.tralics_rc` is the second try.
8. The home directory, or its `src` subdirectory, is searched next. (Depending on the operating system, this can fail, because there is no standard way of defining the home directory of the user).
9. If you set the shell variable `TRALICSDIR` to `somedir`, or `RAWEBDIR` to `somedir`, then `somedir/src/.tralics_rc` is the last try. If neither variable is set, then some default location will be used.
  - If a configuration file has been given on the command line, but not found according to rules 2 and 3, then no configuration file will be used.
  - For rules 4 and 5, only the first hundred lines of the source file are considered, the line must start with a percent character; you can use an upper case T in the string “tralics”; the configuration file name is delimited by single quotes; the line can hold additional characters that are ignored.
  - If a configuration file has been given in the source file, but not found according to rules 4 and 5, then the default configuration file will be used.
  - In the case of rules 3 and 5, a `tcf` file is looked at, say `foo.tcf`. It is first searched in the current directory, then in all directories specified by the `confdir` options, then in a default directory. For instance, if you say `tralics -confdir=mydir/` (final slash optional), the second try would be `mydir/foo.tcf`. If this file is not found, a default location, for instance `../confdir/foo.tcf`, is tried. If you specify `dir1` then `dir2`, then the order will be: current directory, `dir2`, `dir1`, default directory.

In current version, rules 4, 7, 8 and 9 have been removed.

#### 6.4.4 Preparing the translation

Let's consider again file `hello1`, compiled with option `type=rabib`. The transcript file contains the following lines.

```
OK with the configuration file, dealing with the TeX file...
There are 4 lines
Starting translation
\notrivialmath=1
{\countdef \count@=\count255}
```

After that, there is a bunch of lines of the form ‘`countdef x=y`’, and in verbose mode, the bootstrap code, as explained later. The meaning of the last line shown here is: all bootstrap lines have been correctly read.

```
{changing \countref395332=0 into \countref395332=1}
[1] %% Begin bootstrap commands for latex
[2] \@flushglue = Opt plus 1fil
...
[47] %% End bootstrap commands for latex
++ Input stack empty at end of file
```

Our configuration file contains a block of T<sub>E</sub>X code. The transcript file shows them

```
[19] % These commands are inserted verbatim in the file
[20] \newcommand\usebib[2]{\bibliography{#1#2,#1_foot#2+foot,#1_refer#2+refer}}
```

Our configuration file contains also

```
DocAttrib = variable "va'&quot;lue"
DocAttrib =Foo \World
DocAttrib =A \specialyear
DocAttrib =B \tralics
DocAttrib =C \today
```

The effect is to add an attribute to the main element. The normal syntax is: `DocAttrib = foo "bar"`. The attribute name must contain only ASCII letters, the value can consist of any character. An apostrophe is replaced by `&apos;`, double quotes must be given as `&quot;`, as well as some other special characters. Using a command name instead of a string means that the value of the command should be used. The value `\tralics` is replaced by a string of the form ‘Tralics version 2.9’, and `\specialyear` is replaced by the year as used by the Raweb (the current year, in general). The command `\Word` is undefined, and an error is signaled.

Before translating the document, the `ult` file is checked first. Here, the star says that the `@` character should be of category letter while loading the file, and the plus sign says that the file should be searched in the same directory as the main file, and not elsewhere. We finish by showing the class file is found.

```
[1] \InputIfFileExists*+{hello1.ult}{-}{-}
++ file hello1.ult does not exist.
[1] \documentclass{article}
[2] \begin{document}
++ file article.clt does not exist.
++ file ../confdir/article.clt exists.
```

## 6.5 Translating the Raweb

Raweb mode is entered if a configuration file is found that says that the type to use is ‘RA’ or ‘ra’. The document class should be `ra97`, `ra98`, or, for later years, `ra1999`. The example has `ra2003`. This must match the name of the input file, which is `miaou2003`. The document can be translated in one of three versions: first, you may try `latex`, this gives `miaou2003.dvi`; then we have a mode in which `miaou2003.tex` is converted into `miaou.tex`, and `latex` can produce `miaou.dvi`. Finally, `Tralics` may produce `miaou.xml`, and this can be compiled into `wmiaou.dvi`.

Historically, we had a Perl script for the conversion, this was extended to a translator, then re-written in C++. You could edit the script and change it (for instance, if a non-standard name for the `LATEX` executable is needed). Since `Tralics` is nowadays a binary file, you cannot edit it. For this reason the configuration file contains some lines (see old configuration file, lines 29 to 39) that can be modified. These are copied into `user_param.pl` and, after `Tralics` has produced a XML file, it calls an external program (defined by the ‘externalprog’ switch, default being `rahandler.pl`). If the current year (2003 in the example below) is 2007 or more, simplified `ra` mode is entered, not postprocessor is called, and no file `user_param.pl` is created. Here is an example:

```
1 $::makefo='xsltproc --catalogs -o %A.fo %B %C';
2 $::makehtml='xsltproc --catalogs %B %C';
3 $::checkxml='xmllint --catalogs --valid --noout %C';
4 $::makepdf='pdflatex -interaction=nonstopmode %w';
5 $::makedvi='';
6 $::dvitops='';
7 $::generate_dvi='latex -interaction=nonstopmode %t';
8 $::generate_ps='dvips %t.dvi -o %t.ps';
9 $::raweb_dir='/user/grimm/home/cvs/raweb';
```

```

10 $::raweb_dir_src='/user/grimm/home/cvs/raweb/src/';
11 $::ra_year='2003';
12 $::no_year='miaou';
13 $::tex_file='miaou';
14 $::todo_fo=0;
15 $::todo_html=0;
16 $::todo_tex=0;
17 $::todo_lint=0;
18 $::todo_ps=0;
19 $::todo_xml=1;
20 1;

```

Here is an example of a source file, valid in 2003.

```

1 \documentclass{ra2003}
2 \theme{Num}
3 \isproject{YES} % \isproject{OUI} works also
4 \projet{MIAOU}{Miaou}{Mathématiques et Informatique de
5   l'Automatique et de l'Optimisation pour l'Utilisateur}
6 \def\foo{bar}
7 \UR{\URSophia\URFuturs}
8 \declaretopic{abc}{Topic abc}
9 \declaretopic{def}{Topic def}
10 \begin{document}
11 \maketitle
12 ...
13 \begin{module}{composition}{en-tete}{}
14 \begin{catperso}{Head of project team}
15 \pers{Laurent}{Baratchart}[DR INRIA]
16 \end{catperso}
17 \end{module}
18 \begin{module}{diffusion}{dif-conf}{Conferences and workshops}
19 \begin{glossaire}\glo{A}{B\par C}\glo{A1}{B1\par C1}\end{glossaire}
20 \begin{participants}
21 \pers{Laurent}{Baratchart},
22 \pers{José}{Grimm}
23 \end{participants}
24 \begin{motscle}
25 meromorphic approximation, frequency-domain identification,
26 extremal problems
27 \end{motscle}
28 \end{module}
29 \loadbiblio
30 \end{document}

```

This is what Tralics prints, for the full miaou2003 document, in verbose mode

```

This is tralics 2.5 (pl7), a LaTeX to XML translator
Copyright INRIA/MIAOU/APICS 2002-2005, Jos'e Grimm
Licensed under the CeCILL Free Software Licensing Agreement
Starting xml processing for miaou2003.
Configuration file identification: standard $ Revision: 2.14 $
Read configuration file /user/grimm/home/cvs/tralics/.tralics_rc.

```

The lines that follow show the assignments from the configuration file. Note that the year in the mode reflects the compilation data, not the year in the source file.

```

makefo=xsltproc --catalogs -o %A.fo %B %C
makehtml=xsltproc --catalogs %B %C
makepdf=pdflatex -interaction=nonstopmode %w
generatedvi=latex -interaction=nonstopmode %t
generateps=dvips %t.dvi -o %t.ps
theme_vals=com cog num sym bio
dtd is raweb from raweb3.dtd (mode RAWEB2005)

```

Following lines are specific to the Raweb. You can see a summary of all the tests done by the program that converts `miaou2003.tex` to `miaou.tex`. The statistics (number of environments, keywords, etc) are computed by a preprocessor, that has been removed in 2007.

```

Ok with the config file, dealing with the TeX file...
Activity report for MIAOU (Miaou)
Mathématiques et Informatique de l'Automatique et de l'Optimisation pour l'Utilisateur
There are 138 environments
Checked 15 keyword env with 60 keywords (52 unique)
Checked 8 catperso and 31 participant(es) envs with 146 \pers
There were 2 topics
Sections (and # of modules): 1(1) 2(1) 3(2) 4(6) 5(5) 6(13) 7(4) 8(5) 9(3).

```

Whenever a section or a chapter is translated, a line is printed on the terminal. There is a complaint at the end, about a lonely module without title. A title is invented, namely '(Sans Titre)'. A non-trivial task for the post-processor is to remove it (it should not appear on the HTML pages). In 2007, this has become an error.

```

Translating section composition
Translating section presentation
Translating section fondements
Translating section domaine
Translating section logiciels
Translating section resultats
Translating section contrats
Translating section international
Translating section diffusion
Bib stats: seen 57 entries
Seen 64 bibliographic entries
(SansTitre) Only one module seen in the section
Problem with sans titre 1
There was 1 NoTitle not handled

```

Tralics prints now statistics.

```

Used 1756 commands
Math stats : formulas 503, non trivial kernels 299, cells 10227,
  special 1 trivial 149, \mbox 5 large 0 small 118.
List stats: short 0 inc 10 alloc 43456
Buffer realloc 41 string 15750 size 610086; merge 7
Macros created 80 deleted 0
Save stack +1582 -1582
Attribute list search 7539(1494) found 3154 in 5616 elements (1401 after boot)
Number of ref 92, of used labels 36, of defined labels 73, of ext. ref. 19
Modules with 24, without 16, sections with 9, without 15
There were 6 images.
Output written on miaou.xml (250758 bytes).
No error found.
(For more information, see transcript file miaou2003.log)

```

Here you can see the call to the post-processor.

rahandler.pl v2.12, (C) 2004 INRIA, José Grimm, projet APICS  
 Postprocessor did nothing

Since 2006, the syntax of the `\pers` command in a ‘catperso’ environment has changed. Example:

```
\begin{catperso}{Category test}
\pers{Jean}{Dupond}{Scientist}{Inria}
\pers{Jean}{Dupont}{Assistant}{Cnrs}[] [yes]
\pers{Jean}{Durand}{Technical}{University}[] []
\pers{Jean}{Durant}{PHD}{ForeignUniversity}[with a T]
\pers{Jean}{Dumas}{PostDoc}{Public}[with a S]
\pers{Jean}{Dumat}{StudentIntern}{Other}[bla bla ] [no]
\pers{Jean}{Dumont}{ Other }{Other}[bla bla ] [no]
\end{catperso}
```

Here are the commands specific to the Raweb:

- The document class should be ‘ra2003’ (the number must match the suffix of the file name). The document should start with `\maketitle` (see line 11) and end with `\loadbiblio`, see line 29. These commands are set to `\relax` by the translator.
- Some commands can be placed before `\begin{document}`. They will be evaluated in order. Then all commands outside a ‘module’ environment are executed (modules may be re-ordered)<sup>3</sup>. Strange results can be obtained... so that it is better not to put anything between modules.
- You use `\project`, or `\projet`, with three arguments. The first argument is the name of the Team. After converting to lower case letters, this must be the name of the file. The second argument can be empty (in this case the name of the Team is used). Any L<sup>A</sup>T<sub>E</sub>X command is allowed (remember: this is the official name of the Team). Then comes the long name (see example, lines 4-5).
- You use `\theme` to specify the theme. The argument must be one of those defined by the configuration file. The case is irrelevant. See example, line 2.
- You use `\isproject`, to say whether this is a “Project” or not (an argument that starts with y, Y, o or O is true). See example, line 3. Note that the Miaou team has been dissolved on 12/31/2003, and replaced by Apics, a team that had not the status of “project” until 01-01-2005.
- The command `\localisation` is obsolete.
- You use `\UR` for explaining in which UR the team is localized. Line 7 gives an example of a team in ‘Sophia’ and ‘Futurs’. The list of valid URs are defined by the configuration file (see example, lines 45, 46). The ‘UR’ prefix is not in the configuration file. The translator will see a single command, instead of all these four; in the example it is:
 

```
\RAstartprojet{num}{1}{Miaou}{Mathématiques et Informatique de l’Automatique
et de l’Optimisation pour l’Utilisateur}{Sophia Antipolis --- Futurs}
```
- You can use `\declaretopic`<sup>4</sup>. At most ten topics can be declared. The first argument will be replaced by a unique id. For instance, the translator will see the following, instead of lines 8 and 9.

```
{\declaretopic{1}{Topic abc}\declaretopic{2}{Topic def}}
```

<sup>3</sup>This re-ordering mechanism has been removed in 2007

<sup>4</sup>It is ignored since 2007

- You can use the module environment. Modules cannot be nested. A module is the equivalent of a `\section`, so that you cannot use the `\section` command; you can use `\subsection`. The first argument of the environment is the section name. It can be empty, meaning the same section as the previous module. The list of valid sections is defined by the configuration file. The first section cannot have more than one module. A module can have an optional argument (before the section name), a topic reference (the translator will see the numeric value)<sup>5</sup>. This is ignored, in the case a star follows the section name in the configuration file<sup>6</sup>. Then comes the module name and the module title. The module name must be unique, and consists only in letters, digits, and dashes. The module title should not be empty (module in the ‘composition’ section are special; this is an awful hack, because the translator does not know which section is the composition section.) The translator will see

```
\begin{module}{MIAOU}{1}{en-tete}{module1}{module\textit{title}!}
... \end{module}
```

Here ‘MIAOU’ is the name of the team, ‘1’ the section number, and ‘module1’ the unique identifier associated to the module.

- You can use the ‘catperso’ environment, only in a module in section one. You can use ‘participants’ in other sections. These environments contain `\pers` commands (separated by commas in a ‘participants’).
- You can use `\pers` only as indicated above. The program checks a lot a things<sup>7</sup>. Between the first name and the last name, there can be an optional argument, a particle. After the last name, there can be an optional argument. For 2006: after the last name is a mandatory argument that must match the ‘profession\_ vals’ defined by the configuration file, a second mandatory argument that must match the ‘affiliation\_ vals’ defined by the configuration file, an optional argument, a second optional argument that says if the person has a *habilitation* or equivalent. The example shown above is invalid (official keywords are in French).
- You should not use the ‘abstract’ and ‘body’ environments.
- You can use the ‘motscle’ environments for keywords.
- You can use a glossary (see line 19).
- You can use `\moduleref` for a reference to a module.<sup>8</sup>

As a consequence, each module has an implicit label; the statistics shown above say that, of the 73 labels defined, only 36 were used; in fact, 40 of them were implicit.

- The environments ‘glossaire’, ‘moreinfo’, ‘participants’ and ‘keywords’ are designed for modules only. The statistics say : *Modules with 24, without 16, sections with 9, without 15*. You have to interpret this as: there are 24 modules and 9 other sectioning commands (like `\subsection`) with such an information; there is a total of 40 modules (in accordance with *Sections (and # of modules): 1(1) 2(1) 3(2) 4(6) 5(5) 6(13) 7(4) 8(5) 9(3)*).

More information is available on the Web page.

The following commands can be used in any document, but they are specific to the Raweb.

- `\RAstartprojet{a}{b}{c}{d}{e}`. The translation is a `<accueil>` element. You should not use this command, because its translation depends on parameters that are not arguments.

<sup>5</sup>It is ignored since 2007

<sup>6</sup>Feature introduced in version 2.5 (p17)

<sup>7</sup>Most of them removed since 2007

<sup>8</sup>This is an obsolescent command; it may one day become obsolete



- `\begin{RAsection}{domaine} ... \end{RAsection}`. The translation is a `<domaine>` element; it can be the target of a `\ref`, if a `\label` is used, it can also be the target of a `\moduleref`, but this is a bad idea, because the section may disappear, if the Raweb is viewed as a sequence of topics. The first argument can be any character sequence. Note: This is automatically generated, it should not appear in a Raweb source, otherwise the result is not conforming to the Raweb DTD.
- `\begin{module}{MIAOU}{9}{dif-conf}{module28}{Some title} ... \end{module}`  
The syntax is a bit different from what is in the input source. The translation is  
`<module id='uid75' html='module28'><head>Some title</head>...`  
A module can be the target of a `\ref`, or a `\moduleref` via its name (in the example ‘dif-conf’). The name does not appear in the XML result. The value ‘module28’ is computed by the preprocessor. The first two arguments are ignored.
- `\begin{participants} ... \end{participants}`. The name of the environment can be ‘participants’, the final S can be omitted. The translation is an element with the same name. Locally `\par` is redefined to do nothing. No text is allowed. If you look at the example, the comma at the end of line 21 is necessary, but should not be translated.
- `\begin{catperso}{title} ... \end{catperso}`. This is like ‘participants’, but it takes an argument. No commas should be given here. The configuration file may impose restrictions on the value of the title.
- `\declaretopic{a}{b}`. The translation is a `<topic>` element, with attribute `num` equal to `a`, containing an element `<t_titre>` containing the translation of `b`.
- `\begin{moreinfo} ... \end{moreinfo}`. The translation is a `<moreinfo>` element. Paragraphs are allowed.
- `\moduleref[y]{t}{s}{m}`. Argument `y` is optional; only empty values are allowed. Argument `t` is unused (this must be the name of the team in uppercase letters). If `m` is empty, this is reference to section `s`, otherwise to module `m`.
- `\pers`. There are two variants, `\persA` and `\persB`. The `\persA` command (used outside a ‘catperso’ environment) takes two arguments, first name and last name, with an optional argument between them (a particle), and an optional argument after the name (info). The particle is merged with the last name (it exists only for historical reasons). If the last name contains a footnote, it is moved to the info argument. After that, we have three arguments. Initial and final spaces are removed. The command `\@persA` is called on these.
- `\persB` is the variant used, since 2006, in a ‘catperso’ environment. The syntax is: a required argument, the first name of the person, an optional argument, the particle, a required argument, the last name of the person, a required argument, the professional category of the person, a required argument, the organization of the person, an optional argument, the info field, and an optional argument, the HDR flag. Example `\pers {Rose} {Dieng-Kuntz} {Scientist} {Inria} [Chevalier de la légion d'honneur] [yes]`. The process is the same as for a normal `\persA`: The particle is merged with the last name (it exists only for historical reasons). If the last name contains a footnote, it is moved to the info argument. After that, we have six arguments. Initial and final spaces are removed. The function `\@persB` is called on these.
- `\@persA{First}{Last}{Info}`. Initial and final spaces are removed from the arguments. An optional comma after the `\persA` command is removed. The result is a `<pers>` element with two attributes `nom` and `prenom` that contain the first and last names, the value of the element being the info argument. The command is only allowed in a ‘catperso’ or ‘participants’ environment. For instance

```

\begin{catperso}{List of Very Important Persons}
\makeatletter
\def\@persA#1#2#3{\xbox{pers}{\xbox{firstname}{#1}%
\xbox{lastname}{#2}\xbox{info}{#3}}}
\let\pers\persA
\pers{Jean}[de la]{Fontaine}
\pers{Donald}{Knuth}[author of \TeX]
\pers{Leslie}{Lamport}
\end{catperso}

```

translates as

```

<catperso><head>List of Very Important Persons</head>
<pers><firstname>Jean</firstname><lastname>de la Fontaine</lastname>
  <info/></pers>
<pers><firstname>Donald</firstname><lastname>Knuth</lastname>
  <info>author of <TeX/></info></pers>
<pers><firstname>Leslie</firstname><lastname>Lamport</lastname>
  <info/></pers>
</catperso>

```

- `\@persB`. The command takes six arguments, it does all checks required by the Raweb. The command is only allowed in a ‘catperso’ environment.

You can redefine it. Example

```

\begin{catperso}{List of Very Important Persons 2}
\makeatletter
\def\@persB#1#2#3#4#5#6{\xbox{pers}{\xbox{firstname}{#1}\xbox{lastname}{#2}%
\xbox{main-interest}{#3}\xbox{nationality}{#4}\xbox{info}{#5}\xbox{hdr}{#6}}}
\pers{Jean}[de la]{Fontaine}{Tales}{French}[] [phd]
\pers{Donald}{Knuth}{Math}{American}[ author of \TeX]
\pers{Leslie}{Lamport }{Computer Science}{American}
\end{catperso}

```

Translation

```

<catperso><head>List of Very Important Persons 2</head>
<pers><firstname>Jean</firstname><lastname>de la Fontaine</lastname>
  <main-interest>Tales</main-interest>
  <nationality>French</nationality><info/><hdr>phd</hdr></pers>
<pers><firstname>Donald</firstname><lastname>Knuth</lastname>
  <main-interest>Math</main-interest><nationality>American</nationality>
  <info>author of <TeX/></info><hdr/></pers>
<pers><firstname>Leslie</firstname><lastname>Lamport</lastname>
  <main-interest>Computer Science</main-interest>
  <nationality>American</nationality><info/><hdr/>
</pers>
</catperso>

```

## 6.6 Tracing commands

In some cases,  $\TeX$  or Tralics produce wrong results, incomprehensible error messages, and so on. In these cases, you must use specialized commands to see what happens. Since the internal structure of  $\TeX$  is not the same as Tralics, the results in the transcript file may be different.

We have explained the command `\show`: it prints the meaning of a command (useful for a user defined command) and `\showthe` (this shows the value of a variable, counter, a token list, etc). We have also mentioned that `\showbox` prints the content of a T<sub>E</sub>X box or XML element. There is a command `\showlists`; its effect is to indicate the global context; this is not implemented in Tralics. The typical example is from the T<sub>E</sub>Xbook. Given the test file:

```
\tracingcommands=1
\hbox{
$
\ vbox{
\ noindent$$
x\showlists
$$}\bye
```

This is the result of the `\showlists` command.

```
### display math mode entered at line 5
\mathord
.\fam1 x
### internal vertical mode entered at line 4
prevdepth ignored
### math mode entered at line 3
### restricted horizontal mode entered at line 2
\glue 3.33333 plus 1.66666 minus 1.11111
spacefactor 1000
### vertical mode entered at line 0
prevdepth ignored
```

This example does not compile in Tralics: you cannot put a `\vbox` in a math formula. You cannot put a display math formula in a formula.

T<sub>E</sub>X provides 9 commands of the form `\tracingXXXX` described earlier. Each variable defines an integer (in general, positive means verbose). There is a command `\tracingall` that turns everything on. In Tralics, it sets `\tracingmacros`, `\tracingoutput`, `\tracingcommands` and `\tracingrestores` to 1. Only these variables are useful in Tralics (the command `\tracingmath` is new in version 2.11, it controls the math printing). For instance, `\tracingonline` controls whether or not anything is printed on the terminal; for Tralics, debugging information is only printed on the transcript file. Variables like `\tracingparagraphs` and `\tracingpages` show line-break and page-break calculations, performed by T<sub>E</sub>X but not by Tralics. The command `\tracingoutput` shows boxes when they are shipped out (in Tralics, the whole XML tree is printed at the end; if the command is positive, lines are printed, whenever used by the scanner), `\tracinglostchars` indicates all characters not found in the fonts (Tralics never looks at font properties).

The command `\tracingstats` indicates that T<sub>E</sub>X should gather all statistical information available; in Tralics, statistics are always computed; if you call it with the ‘silent’ switch, statistics are not printed on the terminal. Note that the ‘verbose’ switch calls `\tracingall`.

There are three remaining commands: `\tracingmacros` is used whenever a user command is expanded, `\tracingrestores` whenever things are popped from the save stack, and finally, `\tracingcommands` for all other commands. Let’s start with the example given on page 23. This is what you see if `\tracingoutput` is positive:

```
[4] \def\foo#1{\xbar#1}
[5] \def\xbar#1{{\itshape #1}}
[6] \foo{12}
```

It shows the input. This is what you see if `\tracingmacros` is positive:

```

\foo #1->\xbar #1
#1<-12
\xbar #1->{\itshape #1}
#1<-1

```

This is what you see if `\tracingrestores` is positive:

```

+stack: level + 2 for brace
{Push p 1}
{font restore }
+stack: level - 2 for brace

```

This is now what you see if `\tracingcommands` is positive: As you can see, some commands produce more than one line in the transcript file. For instance, a line is printed for `\def` when the command is seen, another one when the whole definition is read.

```

{\def}
{\def \foo #1->\xbar #1}
{\def}
{\def \xbar #1->{\itshape #1}}
{begin-group character {}}
{\itshape}
{font change \itshape}
Character sequence: 1.
{end-group character }}
{Text:1}
Character sequence: 2 .
{Text:2}
}

```

This is the start of the trace on page 26:

```

[61] \begin{x}a b c \end{x}
{\begin}
{\begin x}
+stack: level + 3 for environment

```

What you can see is that `\begin` produces three lines, the second line holds the name of the environment; the last line explains that the stack pointer was changed from 2 to 3; the system remembers that the change comes from an environment, so a closing brace of a `\endgroup` command is illegal. Later on, the trace says:

```

Character sequence: ZbAY c .
{\end}
{Text:ZbAY c }
{\end x}
\endx ->by\end {y}ay

```

In  $\text{\TeX}$ , instead of the first line, you would have seen:

```

{the letter Z}
{blank space }
{the letter c}
{blank space }

```

`\tracingrestores` shows all characters it translates; it puts them on a single line. The character sequence is printed on the transcript file, because the command `\end` wants to be logged. After that, we have a line that contains ‘Text’ in braces. The text is added to the current XML element; a line is printed whenever the buffer is flushed. The buffer is flushed here because it might be used by the internal routine that scans the argument of `\end`. The transcript file contains also:

```

Character sequence: ay.
{\endgroup (for env)}
{Text:ay}
+stack: ending environment x; resuming document.
+stack: level - 3 for environment
Character sequence: .

```

Normally, each line of the form ‘level +3’, is followed by a line ‘level -3’, after that the current level is 2. The last line contains *Character sequence*, followed by a colon, a space, some characters, a period. Instead of ‘some characters’, you see only a space, it could be any character token with the category code of a space. In our case, it is the new line character that marks the end of the line. If the example is followed by an empty line, you will see:

```

[62]
{\par}
{Text:
}

```

What you see here is: open brace, *Text*, colon, some text, close brace. Here ‘some text’ is the space above, shown as a new line character. What the `\par` command does is: a) flush the buffer, so that the text is printed, b) remove space at end of paragraph, c) terminate the element and unwind the XML stack. The next line shows this ‘pop’. The integer 1 is the number of elements on the stack after the pop; you see the content of the stack, just before it is popped, in case the topstack is wrong. After the underscore, there is a suffix that indicates the mode (here `p_v` means that vertical mode will be entered after the pop).

```
{Pop 1: document_v p_v}
```

This is an example from page 31:

```

\E ->\expandafter
{\expandafter \E \E}
\E ->\expandafter
\E ->\expandafter
{\expandafter \expandafter \def}
{\expandafter \def \toto}
\toto ->\titi !
{\def}
{\def \titi !->7}

```

This shows that `\expandafter\foo\bar` shows all three tokens. This can be interesting if these tokens come from the expansion of other tokens. For instance, in a case like this

```

\def\y{yy}
\def\foo{\textit\y}
\expandafter\expandafter\foo

```

it is interesting to know that `\y` is expanded before `\textit`.

The next one is from page 45.

```

[346] \skip\count0=2pt plus \parindent \relax
{\skip}
+scanint for \count->0
+scanint for \skip->1
+scanint for \skip->2
+scandimen for \skip->2.0pt
+scandimen for \skip->3.0pt
{scanglue 2.0pt plus 3.0pt}
{\relax}

```

In T<sub>E</sub>X, there is a big recursive function that converts characters into integers, dimensions and glue. The interesting point is the following: We have two commands `\skip` and `\relax`. The purpose of `\relax` is to stop scanning the glue, because an optional ‘minus’ term. You will not see `\parindent` nor `\count`. The T<sub>E</sub>X output, in this case, consists of two lines. Tralics offers 6 more lines. The last line holds the glue that is effectively read and put in the register. There are three calls to the internal function ‘scanint’, the first is the number of the count register, the second is the number of the skip register, the last is the integer part of the dimension. There are two calls to the internal routine ‘scandimen’, one for each component of the glue (the shrink component is omitted, hence not read).

The next example comes from page 46.

```

1 [3506] \count0=2\ifnum\count0=\count13\fi4
2 {\count}
3 +scanint for \count->0
4 +\ifnum3532
5 +scanint for \count->0
6 +scanint for \ifnum->7
7 +\fi3532
8 +scanint for \count->13
9 +scanint for \ifnum->7
10 +iftest3532 true
11 +scanint for \count->2
12 {\relax}
13 +\fi3532
14 Character sequence: 4 .

```

We have already explained that ‘scanint’ is used to read something in case of assignment; as you can see, the procedure is also called in the case of a conditional. This example is a bit strange. Let’s explain what happens. A line of characters is read (see line L1), tokens are constructed, expanded and evaluated. The evaluator sees a first token, printed on line L2. This matches the rule:  $\langle \text{simple assignment} \rangle$ , in fact, the first clause, which is  $\langle \text{variable assignment} \rangle$ , that is defined as  $\langle \text{integer variable} \rangle \langle \text{equals} \rangle \langle \text{number} \rangle$ , and the first term is `\count` $\langle 8\text{-bit number} \rangle$ . There are two calls to ‘scanint’, the first with a range check. In order to makes things easier to understand, we have given an index to each call, like  $S_1$ ,  $S_2$ , etc.

The job of  $S_1$  is easy: there is one digit, printed on L3. The equals character is the first unread character. It is an  $\langle \text{equals} \rangle$ . After the equals sign an integer is read, via  $S_2$ . This sees the digit 2, then the conditional  $I_{3532}$ . This number was computed by Tralics, it is printed on line L4 to make debug easier. The `\ifnum` command reads two numbers, and a character between them, and compares the numbers. First number is read via  $S_3$ . In fact,  $S_3$  sees `\count` and calls  $S_4$ . Procedure  $S_4$  sees the number 0, followed by an equals sign. It prints that value on line L5. Now  $S_3$  knows that its value is in `\count0`, this is 7, printed on line L6. After that,  $I_{3532}$  has a first number 7, and sees the equal sign, and reads the second number via  $S_5$ . This sees `\count` and reads a number via  $S_6$ . This reads 13. Then comes `\fi`. The `\fi` command prints line L7 on the transcript file. This terminates  $I_{3532}$ . This is not possible: our conditional is still reading the second number. As a consequence, two tokens are pushed back, a `\fi` and a `\relax`, in this order: the `\relax` is read again first. This terminates expansion of the `\fi`.

Our procedure  $S_6$  is programmed to fully expand tokens, and read them as long as digits are seen (even if the result overflows). It is unaware of the fate of the `\fi` token. All it knows is that the first unexpandable token after the digit 3 is `\relax`. Thus,  $S_6$  has finished its jobs: the value is 13, printed on line L8. Then  $S_5$  knows its result: the value is `\count13`, hence 7, printed on line L9. After that,  $I_{3532}$  has two numbers, they are the same, the test is true, as can be seen on line L10. Normal expansion resumes; however the condition stack has a marker that tells that the next `\fi` or `\else` matches  $I_{3532}$ . If the test had been false, the expansion of the test would

have read, at high speed, all tokens up to the next `\fi` or `\else`. There are four unread tokens: a `\relax`, a `\fi`, the digit 4, and the newline character. Remember  $S_2$ : this is a procedure that has read a digit, and wonders what follows. It does not care how complicated the task of ‘expand’ may be. It just wants a non-expandable token. In fact, the current token is `\relax`. Thus,  $S_2$  knows it has read all digits; it prints the result of the transcript file L11. As a consequence, 2 is stored in `\count0`. The `\relax` token does nothing (let’s hope nobody has redefined it), see line L12. The conditional is terminated because of the inserted `\fi` token, line L13. After the last character on the line is translated, a new line is read, and a line of the form L1 will be printed; before a line is added to the transcript file, the internal buffer is flushed, this explains line L14. Note that the following sequence provokes an error in  $\text{\TeX}$  (if both counters are equal)

```
\edef\F00{\ifnum\count1=\count13\fi}
\expandafter\def\F00{2x}
```

It is accepted by Tralics. As a consequence, the special `\relax` token inserted by  $\text{\TeX}$  always behaves like `\relax`.

When Tralics sees a `\else` in a true condition, it reads everything at high speed, until finding the matching `\fi`; the example below shows the trace in such a case.

```
1 [1] \iftrue \else \ifcat 11\ifx ab \else \fi \ifnum1=2 \else \fi \fi\fi
2 +\iftrue1
3 +\iftest1 true
4 +\else1
5 +\ifcat1(+1)
6 +\ifx1(+2)
7 +\else1(+2)
8 +\fi1(+2)
9 +\ifnum1(+2)
10 +\else1(+2)
11 +\fi1(+2)
12 +\fi1(+1)
13 +\fi1
```

## 6.7 Pictures and friends

We explain here the translation of some commands related to the picture environment. The syntax is unusual. In some cases, a pair of integers or a pair of real numbers are read. These numbers are multiplied by the value of the current unit of length, and the XML file contains these values, in pt, without the unit. The default value of `\unitlength` is 1pt. For instance

```
\setlength{\unitlength}{3pt}
\def\ten{10}
\put(\ten,\ten.2){x}
```

translates as `<pic-put xpos='30' ypos='30.59999'>x</pic-put>`. As the example shows, arithmetic on scaled integers is exact, but ‘10.2’ cannot be represented exactly. In some cases, arguments are converted to attributes, and errors can be signaled. In the case ‘`\put(1.2,3,4) {}`’, you will see *Missing unit (replaced by pt) {Character . of catcode 12}*, followed by three other errors. In the case of ‘`\makebox(1,2)[$\alpha$]{x}`’ the error is *unexpected element formula*. Without the dollar signs, an error is signaled, the math formula is discarded, a second error is signaled with *unexpected element error*. If you invoke Tralics with ‘-noxmlerror’, the first error produces no `<error>` element, so that there is only one error.

- `\begin{picture}(A,B)(C,D) ... \end{picture}`. The first two arguments are required, the other ones are optional. Normally, there should be no text in the environment, the mode is neither horizontal nor vertical, `\par` commands are forbidden. Example:

```
\begin{picture}(1,2)
\begin{picture}(3,4)(5,6)
  1=1
\end{picture}
\end{picture}
```

This translates as

```
<picture width='1' height='2'>
  <picture xpos='5' ypos='6' width='3' height='4'>
    1=1
  </picture>
</picture>
```

- `\fboxrule`, `\fboxsep`. These two dimensions are defined by L<sup>A</sup>T<sub>E</sub>X, unused by Tralics. Default values: 0pt and 3pt.
- `\makebox(A,B)[x]{y}`. This translates the value  $y$ , inside a group (as is the case for all ‘box’ commands that follow). The result is `<pic-framebox>` element, with attributes `width='A'`, `height='B'`, and `position='x'`.
- `\framebox(A,B)[x]{y}`. Same as `\makebox` above, but the attribute `framed` is set to true. As an example, the translation of

```
\makebox(1,2)[l]{\it x}
\makebox(1,2){\it x}
\framebox(1,2)[l]{\it x}
\framebox(1,2){\it x}
```

is

```
<pic-framebox width='1' height='2' position='l'>
  <hi rend='it'>x</hi></pic-framebox>
<pic-framebox width='1' height='2'>
  <hi rend='it'>x</hi></pic-framebox>
<pic-framebox width='1' height='2' position='l' framed='true'>
  <hi rend='it'>x</hi></pic-framebox>
<pic-framebox width='1' height='2' framed='true'>
  <hi rend='it'>x</hi></pic-framebox>
```

- `\makebox[w][x]{y}`. See alternate syntax above. The first argument must be a dimension, it is the width of the box; its value is currently ignored. The second argument must be one character of ‘lrcs’. A paragraph is started if necessary. The result is a `<mbox>` element, containing the translation of  $y$ , unless either the box contains only a `<figure>`, or if it contains only text (characters, and font changes), case where the result is  $y$ .
- `\mbox{y}`. This is exactly like `\makebox`, with a simple syntax. For instance:

```
\makebox{\it x} \mbox{\it x}
\makebox[2cm]{y\it x}
\makebox[2cm][l]{\it x}
\makebox{$$} \mbox{$$}
\makebox{\xbox{foo}{bar}} \mbox{\xbox{foo}{bar}}
\makebox{\includegraphics{x}} \mbox{\includegraphics{x}}
```



translates as

```
<hi rend='it'>x</hi> <hi rend='it'>x</hi>
<mbox>y<hi rend='it'>x</hi></mbox>
<mbox position='l'><hi rend='it'>x</hi></mbox>
<mbox><formula type='inline'><simplemath>x</simplemath></formula></mbox>
<mbox><formula type='inline'><simplemath>x</simplemath></formula></mbox>
<mbox><foo>bar</foo></mbox> <mbox><foo>bar</foo></mbox>
<figure rend='inline' file='x' /><figure rend='inline' file='x' />
```

- `\framebox[w][x]{y}`. This is like `\makebox`, but the result is a `<fbox>`, instead of `<mbox>`, with attribute `rend='boxed'`. In the case where the content is a `<figure>`, then no box is created but the attribute `framed` is added to the figure.
- `\fbox{y}`. Like `\framebox`, with a simple syntax. For instance

```
\framebox{\it x} \fbox{\it x}
\framebox[2cm]{y\it x}
\framebox[2pt][l]{\it x}
\framebox{$x$} \fbox{$x$}
\framebox{\xbox{foo}{bar}} \fbox{\xbox{foo}{bar}}
\framebox{\includegraphics{x}} \fbox{\includegraphics{x}}
```

translates as

```
<fbox rend='boxed'><hi rend='it'>x</hi></fbox>
<fbox rend='boxed'><hi rend='it'>x</hi></fbox>
<fbox width='56.9055pt' rend='boxed'>y<hi rend='it'>x</hi></fbox>
<fbox width='2.0pt' position='l' rend='boxed'><hi rend='it'>x</hi></fbox>
<fbox rend='boxed'><formula type='inline'><simplemath>x</simplemath>
</formula></fbox>
<fbox rend='boxed'><formula type='inline'><simplemath>x</simplemath>
</formula></fbox>
<fbox rend='boxed'><foo>bar</foo></fbox>
<fbox rend='boxed'><foo>bar</foo></fbox>
<figure framed='true' rend='inline' file='x' />
<figure framed='true' rend='inline' file='x' />
```

- `\scalebox{x}{y}`. The translation is a `<scalebox>` containing `y`, with a `scale` attribute whose value is `x`.
- `\rotatebox{x}{y}`. The translation is a `<pic-rotatebox>` containing `y`, with a `angle` attribute whose value is `x`.
- `\dashbox{A}(B,C)[d]{y}`. The translation is a `<pic-dashbox>` containing `y`, with a `position` attribute whose value is `d`. The three values `A`, `B` and `C` are numbers, affected by the unit length. They produce attributes `dashdim`, `width` and `height`. Example:

```
\setlength\unitlength{2pt}
\scalebox{3}{\it x}
\rotatebox{90}{\it x}
\dashbox{1.2}(2,3)[l]{\it x}
\dashbox{1.2}(2,3){\it x}
```

The translation is

```
<scalebox scale='3'><hi rend='it'>x</hi></scalebox>
<pic-rotatebox angle='90'><hi rend='it'>x</hi></pic-rotatebox>
<pic-dashbox dashdim='2.4' width='4' height='6' position='l'>
```

```

<hi rend='it'>x</hi></pic-dashbox>
<pic-dashbox dashdim='2.4' width='4' height='6'>
<hi rend='it'>x</hi></pic-dashbox>

```

- `\begin{minipage}[x][y][z][t]{dim} ... \end{minipage}`. Optional arguments  $y$  and  $t$  are ignored. Arguments  $x$  and  $z$  should define vertical position. The last argument is a glue. Note: the translation starts in vertical mode. The command `\nocentering` is called.

- `\parbox[x][y][z]{dim}{y}`. This should be the non-environment version of ‘minipage’, but it is much more primitive. It behaves like `\hbox` and `\xbox` for argument parsing. Example

```

\begin{center}
a
\begin{minipage}{2cm} a \end{minipage}
\begin{minipage}[c][l][b][r]{2cm plus 3pt} a \end{minipage}
\end{center}
\parbox[foo][bar][gee]{2cmPlus3mm}{some \it box content}.

```

The translation is

```

<p rend='center'>a
<minipage width='56.9055pt'><p>a </p></minipage>
<minipage inner-pos='b' pos='c' width='56.9055pt'><p>a </p></minipage></p>
some <hi rend='it'>box content</hi><p>.</p>

```

- `\bezier{N}(A,B)(C,D)(E,F)`. The first argument must be an integer, others are real numbers, affected by `\unitlength`. The result is a `<pic-bezier>` element.
- `\qbezier[N](A,B)(C,D)(E,F)`. As above, but the first argument is optional. A Bezier curve is defined by three points, you must give the coordinates. In the first implementation, you had to specify the number of intermediate points used for plotting. For instance

```

\bezier{10}(1,2)(3,4)(5,6)
\qbezier(1,2)(3,4)(5,6)
\qbezier[10](1,2)(3,4)(5,6)

```

translates as

```

<pic-bezier a1='1' a2='2' b1='3' b2='4' c1='5' c2='6' repeat='10' />
<pic-bezier a1='1' a2='2' b1='3' b2='4' c1='5' c2='6' />
<pic-bezier a1='1' a2='2' b1='3' b2='4' c1='5' c2='6' repeat='10' />

```

- `\put(A,B){x}`. The translation is a `<pic-put>` element containing the translation of  $x$ , with attributes `xpos` and `ypos` that come from  $A$  and  $B$ .
- `\xscale`, `\yscale`, `\xscaley`, `\yscalex`. These are user defined commands, that should contain real numbers. The default values are 1, 1, 0, and 0.
- `\scaleput(A,B){x}`. The translation is a `<pic-scaleput>`. It is like `\put`, but current scale values are put in the element. For instance the translation of

```

\setlength\unitlength{2pt}
\put(1,2){x}
\def\xscale{2.0}\def\yscale{3.0}\def\xscaley{4}\def\yscalex{0.5}
\scaleput(1,2){x}

```

is

```

<pic-put xpos='2' ypos='4'>x</pic-put>
<pic-scaleput xpos='2' ypos='4'
xscale='2.0' yscale='3.0' xscaley='4' yscalex='0.5'>x</pic-scaleput>

```

- `\multiput(A,B)(a,b){n}{x}`. The translation is a `<pic-multiput>` element, with all the arguments as attributes. If the command is followed by a star, then  $n$  calls of `\put` are evaluated, with the same argument  $x$ , but at positions  $(A + ka, B + kb)$  instead of  $(A, B)$ . See example of the little car on figure 6.1; the ruler was constructed with `\multiput`, a star is used for the figures.
- `\frame{x}`. This translates as `<pic-frame>` with the value of  $x$ .
- `\oval(A,B)[c]`. This translates as `<pic-oval/>` with some attributes, `xpos`, `ypos`, `spec`.
- `\line(A,B){c}`. This translates as `<pic-line/>`. The first two arguments have to be integers. Only small values are recognized by L<sup>A</sup>T<sub>E</sub>X. The last argument is a real number. Attributes are `xdir`, `ydir`, `width`.
- `\vector(A,B){c}`. This translates as `<pic-vector/>`. Same comments as above. Example.

```

\setlength\unitlength{2pt}
\oval(2,3) \oval(2,3)[4]
\frame{x}
\line(2,3){12.2} \vector(2,3){12.2}

```

The translation is

```

<pic-oval xpos='4' ypos='6' />
<pic-oval xpos='4' ypos='6' specs='4' />
<pic-frame>x</pic-frame>
<pic-line xdir='2' ydir='3' width='24.4' />
<pic-vector xdir='2' ydir='3' width='24.4' />

```

- `\thicklines`. The translation is `<pic-thicklines/>`.
- `\thinlines`. The translation is `<pic-thinlines/>`.
- `\linethickness{x}`. The translation is `<pic-linethickness>` with attribute `size`.
- `\arc[n](A,B){c}`. The translation is an empty `<pic-arc>` with attributes `nbsymb`, `angle`, `xpos`, `ypos` (and also `unit-length`).
- `\bigcircle[n]{c}`. The translation is an empty `<pic-bigcircle>` element with attributes `nbsymb`, `size`, (and also `unit-length`).
- `\circle{c}`. The translation is an empty `<pic-circle>`. For instance

```

\setlength\unitlength{2pt}
\arc[17](2,3){40} \arc(2,3){40}
\bigcircle[17]{40} \bigcircle{40}
\circle{40}

```

translates to

```

<pic-arc angle='40' xpos='4' ypos='6' unit-length='2' nbsymb='17' />
<pic-arc angle='40' xpos='4' ypos='6' unit-length='2' />
<pic-bigcircle size='40' unit-length='2' nbsymb='17' />
<pic-bigcircle size='40' unit-length='2' />
<pic-circle size='80' />

```

- `\curve[n](A1,B1)(A2,B2)(A3,B3),..., (An,Bn)`.
- `\closecurve[n](A1,B1)(A2,B2)(A3,B3),..., (An,Bn)`.
- `\tagcurve[n](A1,B1)(A2,B2)(A3,B3),..., (An,Bn)`. These produces curves.

- `\dashline[A]{B}[C](A1,B1)(A2,B2)(A3,B3),\dots,(An,Bn)`.
- `\dottedline[A]{B}[C](A1,B1)(A2,B2)(A3,B3),\dots,(An,Bn)`.
- `\drawline[A][C](A1,B1)(A2,B2)(A3,B3),\dots,(An,Bn)`. These produce dashes. Example:

```

\setlength\unitlength{2pt}
\curve[12](1,2)(3,4)(4,5)(7,8)
\tagcurve[12](1,2)(3,4)(4,5)(7,8)
\closecurve[12](1,2)(3,4)(4,5)(7,8)
\curve(1,2)(3,4)(4,5)(7,8)
\tagcurve(1,2)(3,4)(4,5)(7,8)
\closecurve(1,2)(3,4)(4,5)(7,8)
\dashline[12]{13}[14](1,2)(3,4)(4,5)(7,8)
\dottedline[12]{13}[14](1,2)(3,4)(4,5)(7,8)
\drawline[12][14](1,2)(3,4)(4,5)(7,8)

```

The translation is

```

<pic-curve unit-length='2' nbsymb='12'>1,2</pic-curve><p>(3,4)(4,5)(7,8)
<pic-tagcurve unit-length='2' nbsymb='12'>1,2</pic-tagcurve>(3,4)(4,5)(7,8)
<pic-closecurve unit-length='2' nbsymb='12'>1,2</pic-closecurve>(3,4)(4,5)(7,8)
<pic-curve unit-length='2'>1,2</pic-curve>(3,4)(4,5)(7,8)
<pic-tagcurve unit-length='2'>1,2</pic-tagcurve>(3,4)(4,5)(7,8)
<pic-closecurve unit-length='2'>1,2</pic-closecurve>(3,4)(4,5)(7,8)
<dashline arg3='14' arg2='13' arg1='12'>
  <point xpos='2' ypos='4'><point xpos='6' ypos='8'>
  <point xpos='8' ypos='10'><point xpos='14' ypos='16'></dashline>
<dottedline arg3='14' arg2='13' arg1='12'><point xpos='2' ypos='4'>
  <point xpos='6' ypos='8'><point xpos='8' ypos='10'>
  <point xpos='14' ypos='16'></dottedline>
<drawline arg3='14' arg1='12'><point xpos='2' ypos='4'>
  <point xpos='6' ypos='8'><point xpos='8' ypos='10'>
  <point xpos='14' ypos='16'></drawline>

```

## 6.8 The title page

Consider the following document fragment. Note the spellings of the commands `\keyword` and `\motcle`, this is not the same as for the environments of the Raweb.

```

1 \documentclass[a4paper]{report}
2 \usepackage{RR}
3
4 \providecommand\Tralics{\xbox{Tralics}{}}
5 \def\XML{XML}
6
7 \RRtitle{Tralics, a \LaTeX\ to XML translator\Partie I}
8 \RRetitle{Tralics, a \LaTeX\ to XML translator\Part I}
9 \RRauthor{José Grimm\thanks{Email: Jose.Grimm@sophia.inria.fr}}
10
11 \RRprojet{Apics}
12 \RRtheme{\THNum}
13
14 \RRresume{
15 Dans cet article\par nous décrivons le logiciel \Tralics,\par...}
16 \RRabstract{
17 In this paper we describe \Tralics, a \LaTeX\ to \XML\ translator.}

```

```

18
19 \RRdate{Aout 2005}
20 \URSophia
21 \motcle{Latex, XML, HTML, MathML, Perl, PostScript, Pdf}
22 \keyword{Latex, XML, HTML, MathML, Perl, PostScript, Pdf}
23
24 \begin{document}
25 \makeRR
26 text
    This is translated by Tralics as follows.
    <?xml version='1.0' encoding='iso-8859-1'?>
    <!DOCTYPE std SYSTEM 'classes.dtd'>
    <!-- Translated from latex by tralics 2.9, date: 2006/10/03-->
    <std chapters='true'>
    <ftitle>Tralics, a <LaTeX/> to XML translator Partie I</ftitle>
    <title>Tralics, a <LaTeX/> to XML translator Part I</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><resume><p>Dans cet article</p>
    <p>nous décrivons le logiciel <Tralics/>,</p>
    <p>...</p></resume>
    <abstract><p>In this paper we describe <Tralics/>, a <LaTeX/> to XML
    translator.</p></abstract>
    <date>Aout 2005</date><location>Sophia Antipolis</location>
    <motcle>Latex, XML, HTML, MathML, Perl, PostScript,
    Pdf</motcle><keyword>Latex, XML, HTML, MathML, Perl, PostScript,
    Pdf</keyword><p>text</p>
    </std>

```

The document you are reading here is a technical report (it has `\makeRT` instead of `\makeRR`, but it uses the same commands starting with ‘RR’, they are defined in the file `RR.sty`, and `Tralics` (since version 2.9) uses some equivalents from the file `RR.pt`. This was translated to XML then HTML (may be you are reading the HTML version).

Since year 2006, Inria’s research reports are to be put on HAL<sup>9</sup>; before that, they were stored on Inria’s Web server. The meta-data were generated automatically: the author sends to the “gescap” mailing list the beginning of the document, up to the magic command `\makeRR`; this is translated by `Tralics` (that does not care about missing `\end{document}`); a post-processor extracts the `<RRstart>` element from the XML result, and converts it to HTML.

Let’s compile the same file as above with the command `tralics tptest.tex -type RR`.

```

<?xml version='1.0' encoding='iso-8859-1'?>
<!DOCTYPE rr SYSTEM 'raweb.dtd'>
<!-- Translated from latex by tralics 2.9, date: 2006/10/03-->
<rr type='RR' chapters='true'>
<RRstart><UR>
<URSophia/>
</UR>
<title>Tralics, a <LaTeX/> to XML translator Partie I</title>
<etitle>Tralics, a <LaTeX/> to XML translator Part I</etitle>
<projet>Apics</projet>
<motcle>Latex, XML, HTML, MathML, Perl, PostScript, Pdf</motcle>
<keyword>Latex, XML, HTML, MathML, Perl, PostScript, Pdf</keyword>
<resume><p>

```

<sup>9</sup>HAL-INRIA is an environment for self-archiving of scientific publications and providing free access to them, like arXiv

```

Dans cet article</p>
<p>nous décrivons le logiciel <Tralics/>,</p>
<p>...</p></resume>
<abstract><p>
In this paper we describe <Tralics/>, a <LaTeX/> to XML translator.</p></abstract>
<author><auth>José Grimm<note id='uid1' place='foot'>
  Email: Jose.Grimm@sophia.inria.fr</note></auth>
</author>
<date>Aout 2005</date>
<RRnumber>????</RRnumber>
<Theme>
<THNum/>
</Theme>
</RRstart>
<p>text</p>
</rr>

```

We have shown above the content of the files RR.plt and RR.tcf. One file ends with `\let \RRstyisuseful \relax` and the other starts with `\ifx \RRstyisuseful \relax \endinput \fi`. As a result, if you load the tcf file, the content of the plt file is discarded.

The syntax in the TitlePage part of a configuration file is the following: each line has some fields, that can be of type A (the word ‘alias’, or ‘action’ or ‘execute’) or type C (a command, a backslash followed by some letters), or E (an element name, delimited by less-than and greater-than) or S (a string, delimited by a double quote). Before each field, you can put one or two modifiers. Only the second and the third fields can have a modifier. More details can be found on the web page. The following combinations are recognized.

**CESS** as in `\makeRR <RRstart> "" "type = 'RR'".` This declaration has to be the first in the list. It can be given only once. No modifiers are allowed. It defines a command `\makeRR`, that can be used only once in the document, after `\begin{document}`.

The effect is to insert the `<RRstart>` element into the XML tree, after some checks (that may produce an error). In what follows, we shall call it the TPA element. This element is formed of other elements defined by the titlepage info, the names of these elements are statically defined, their content is dynamic (i.e., the names depends on the configuration file, the content on the  $\TeX$  document). The first string is a list of attributes added to the TPA element and the second string is a list of attributes added to the document element. In our example, the first string is empty.

In the case where one of the attributes of the second string has the value ‘only title page’, then `\endinput` is evaluated just after the Titlepage command. This means that everything after the titlepage command is ignored. This is useful if you want to extract the titlepage information from a document, without converting the whole document.

**AESS** as in `alias "" "type = 'RT'".` This declaration is valid only after a CESS declaration (or after another AESS declaration). It defines a command `\makeRT` that can be used instead of `\makeRR` (only one of these commands can be used). The result is the same; however it can use different attributes. (Same remark as above for special attribute values in the second string). In what follows, the `\TPA` command means one of the commands defined by this rule or the preceding one.

**CEES** as in `\RRauthor + <author> <auth> "Pas d'auteurs".` Note that the plus sign is required before the `<author>` element. This declaration has as side effect that the TPA element will contain a `<author>` element, formed of a number of `<auth>` elements. Initially there is only one, initialized with ‘Pas d’auteurs’.

The declaration has another effect, it defines a command `\RRauthor`, that has to be used before the `\TPA` command. It takes one argument, and creates a `<auth>` element whose content is the translation of the argument. This element is added to the end of the `<author>` element. The command can be used more than once, in the case there are multiple authors. Note that the default value is removed in case at least one value is given.

**CCS** as in `\myself \RRauthor "JG"`. The effect is the same as `\def\myself{\RRauthor{JG}}`. However, the string argument is not translated, it is taken verbatim.

**AC** as in `alias \URRocq`. This makes `\URrocq` an alias for the command defined on the previous line. Aliasing is achieved via `\let`.

**E** as in `<UR> -`. The dash after the element is required. Another example can be `<sUR fr='unité de recherche' en='research unit'> -`. In this second example, we have an element named `<sUR>`, that has two attributes. The effect is to put, in the XML result, this element (with its attributes), and its content is a list of items declared in the configuration file (the list can be empty).

**CE** as in `\URSophia ?+<UR>` or `\URFuturs ?+<UR d='true'>`. This has as effect to define a command, here `\URSophia` or `\URFuturs`, that takes no argument, whose effect is to insert, to the element `<UR>` (that must be defined by a previous rule), an empty element, whose name is `<URSophia>`, and that has the attributes of `<UR>`.

**CEE** as in `\Paris ?<UR> <Rocquencourt>`. The effect is to define a command `\Paris`, that behaves like `\URSophia`, but the element created is `<Rocquencourt>` instead of one named `<Paris>`.

**S** A character string is inserted verbatim. Note that less-than signs are not converted to entities like `&lt;`;

**AC** as in `execute \foo` or `action \foo`. The expression `\xbox{}{\foo}` is translated. The resulting box is added to the XML tree.

**CES** as `\RRtitle <title> "pas de titre"`. This is the generic command. The element can have the modifiers `p`, `q`, `e` or `E`, and the value can have the modifiers `+ABC`. The effect is to define a command `\RRtitle` (or an environment `'RRtitle'` if the `E` modifier has been given), that can be used only before the `\TPA` command. The argument of the command (or the content of the environment) is translated, put in a `<title>` element, and added to the TPA element.

If no modifier is given for the element, paragraphs are forbidden in the argument. If you want to use paragraphs (either `\par` or `\`) you must use the `P` modifier (lower-case letter). In the same fashion, a lower case `E` means environment without paragraphs, an upper case `E` means environment with paragraphs. If the `q` modifier is given, paragraphs are forbidden, but you can use `\`, which is ignored. (In fact, the command reads an optional star, an optional argument, and the result is replaced by a space). Note that, in this document, there is a `\` in the title, that appears as a space in the page headings. This is done by redefining `\` to `\space`, so that optional arguments are not taken into account. There is a dirty hack in Tralics.

If no modifier is given for the value, then `<title>pas de titre</title>` is added to the TPA element in case the command is never used.

Near the end of the titlepage example, we define `\cmdp`, `\cmdA`, `\cmdB`, and `\cmdC` in a similar fashion, but add a modifier before the value. None of these commands is used in the T<sub>E</sub>X file; if you uncomment them, you can observe the following facts.

- Tralics complains with: *Error signaled at line 25 of file tptest.tex: No value given for command \cmdp.* In fact, when Tralics sees the `\makeRR` command, it notices that `\cmdp` has not been called and complains, because the plus-sign modifier means: this value is required.
- In the case where the modifier is one of A, B or C, then the default value is a  $\LaTeX$  command (in fact, a list of characters, that may be interpreted as a set of  $\LaTeX$  commands, and must be translated). Tralics removes the double quotes, and inserts the characters in one of its buffers, or converts the string into a token list that is appended to the input token list. The non trivial point is: when is the command evaluated?
- In the case of ‘A’ modifier, the command is evaluated just before the `\documentclass` command. There may be other lines that are not in the  $\TeX$  source file that are evaluated there.
- In the case of ‘B’ modifier, the command is evaluated at begin-document. More precisely `\cmdb{\cmdBval}` is tokenised, and the result is added to the token list maintained by `\AtBeginDocument`.
- In the case of ‘C’ modifier, the command is evaluated when the TPA command is seen. The important point is that category codes in effect at this moment will be used to convert the string into a sequence of tokens.

There is special trick for the case where the name of the element associated to the command is empty. Assume that the configuration file contains `\RRtheme <> +"pas de theme"`. In the case where the user does not use `\RRtheme`, an error will be signaled, and the text will appear in the resulting XML. If the user says `\RRtheme{foo}`, then Tralics remembers the use and issues no complain. Moreover, it reads the argument, and pushes `foo\par` in the input stream (the reason why `\par` is executed is to make sure that Tralics remains in vertical mode).

## 6.9 Array and Tables

We describe here the implementation of the arrays in Tralics. One has to distinguish between ‘table’ which is an environment in which you can put some objects (in general tables) with a caption; like the ‘figure’ environment, this generates a floating object. On the other hand, the ‘array’ and ‘tabular’ environments can be used to create a table: the first one is designed for math only, the second for non-math material. Math tables are described in the chapter about mathematics. There is currently no difference between ‘figure’, ‘figure\*’ and ‘wrapfigure’.

Example

```
\begin{table}
\begin{tabular}{c} x \y \end{tabular}
\caption{My caption}
\label{t1}
\end{table}
\begin{tabular}{c} \ref{t1} \end{tabular}
```

The translation is as follows. As you can see, both objects have the same name. If the table contains a tabular, only one XML object is created.

```
<table id='uid1'>
  <head>My caption</head>
  <row><cell align='center'>x</cell></row>
  <row><cell align='center'>y</cell></row>
</table>
```



Year	Word Population
8000BC	5,000,000
50AD	200,000,000
1650AD	500,000,000
1850AD	1,000,000,000
1945AD	2,300,000,000
1980AD	4,400,000,000

Table 6.1: A T<sub>E</sub>X table

```
<p><table rend='inline'><row><cell halign='center'><ref target='uid1' /></cell>
</row></table></p>
```

## The tabular environment

You can say `\begin{tabular} [pos] {cols} ... \end{tabular}` or `\begin{tabular*}{width} [pos] {cols} ... \end{tabular*}`. In both cases, the result is a `<table>` element. This element has a `vpos` attribute whose value is `t`, `b` or `c`, provided that the optional `[pos]` argument is one of `[t]`, `[b]` or `[c]`. The element has a `width` attribute with value `xx`, provided that the `'tabular*'` environment has been used and the first argument evaluates to `xx` as a dimension. The resulting element consists of some `<row>` elements, each of which contains some `<cell>` elements. A more complicated example:

```
\begin{tabular*}{10pc}[b]{lrc}
\hline
a&b&c\ \ [2pt]
\multicolumn{1}{l}{A}&B&C\ \ \hline
\end{tabular*}
```

The translation here shows that the name of elements and attributes can be changed.

```
<Table VPos='b' TableWidth='120.0pt' rend='inline'>
  <Row SpaceAfter='2.0pt' TopBorder='true'>
    <Cell Align='Cleft'>a</Cell>
    <Cell Align='Cright'>b</Cell>
    <Cell Align='Ccenter'>c</Cell>
  </Row>
  <Row BottomBorder='true'>
    <Cell Align='Cleft' Cols='1'>A</Cell>
    <Cell Align='Cright'>B</Cell>
    <Cell Align='Ccenter'>C</Cell>
  </Row>
</Table>
```

## Interpreting the preamble

The preamble of the array is the quantity marked `'{cols}'` in the description above. This is a specification for columns. It specifies how the columns should be formatted. In standard L<sup>A</sup>T<sub>E</sub>X, you cannot use more columns than specified; in Tralics, this is not relevant. The T<sub>E</sub>X primitive is called `\halign`, and L<sup>A</sup>T<sub>E</sub>X has to construct a preamble that matches the requirements of T<sub>E</sub>X; it is very difficult to implement the T<sub>E</sub>X algorithm, so that we make no attempt to implement the commands. This is an example

```

\ vbox{\ offinterlineskip
\ hrule
\ halign{\ vrule#&
\ strut \quad\ hfil#\quad\ cr
height2pt &\ omit&&\ omit&\ cr
&Year\ hfil&&Word Population&\ cr
height2pt &\ omit&&\ omit&\ cr
\ noalign{\ hrule}
\ noalign{\ vskip 2pt}
\ noalign{\ hrule}
height2pt &\ omit&&\ omit&\ cr
&8000BC&&5,000,000&\ cr
&50AD&&200,000,000&\ cr
&1650AD&&500,000,000&\ cr
&1850AD&&1,000,000,000&\ cr
&1945AD&&2,300,000,000&\ cr
&1980AD&&4,400,000,000&\ cr
height2pt &\ omit&&\ omit&\ cr}\ hrule}

```

The table has 5 columns, of the form ABABA, because the preamble has the form `&A&B\cr`, the first `&` marks repetition. Both templates A and B are formed of a `<u>` part, then `#`, then a `<v>` part. In the table, the `#` means: “stick the text of each column entry in this place”. In the case of A, this is almost always empty. In some cases, it is ‘height2pt’, case where the values of B are `\omit`. Here `\omit` says that `<u>` and `<v>` should be omitted; the important point is that the `\strut` be omitted. This gives additional vertical space, of exactly 2pt; other rows have (at least) the vertical size of a `\strut`. Very often row are too narrow;  $\LaTeX$  has a command `\arraystretch` that controls this. Note that A is `\vrule#` and `\vrule` is a command that accepts an optional argument. Note how horizontal rules are inserted in the table.

As the previous example shows, there are three standard column types: `c`, `l` and `r` (centered, left-aligned, right-aligned). A  $\TeX$  preamble like `\quad\hfil#\quad` corresponds to ‘`r`’ (instead of `\quad`,  $\LaTeX$  uses some default intercolumn space that can be modified). You can also say `p{dim}`. This should typeset the column in a `\parbox[t]{dim}`. This feature is not implemented: the argument is ignored, and `p` is replaced by `c`. Note: `\parbox` currently ignores its argument in *Tralics*.

The `array.sty` package adds two options that take a dimension as argument: ‘`m`’ and ‘`b`’. The ‘`b`’ option is like the ‘`p`’ option, but bottom-aligned. The ‘`m`’ option should be used only in math mode (i.e. for the `array` environment, and not `tabular`). In *Tralics*, there is no difference between ‘`b`’, ‘`m`’ and ‘`p`’.

There is a `@{text}` option. It inserts ‘`text`’ in every row, where ‘`text`’ is processed in math mode in the ‘`array`’ environment and in LR mode in the ‘`tabular`’ and ‘`tabular*`’ environments. Such an expression suppresses the space that  $\LaTeX$  normally inserts between columns. For instance, an array specification like `{l@{\hspace{1cm}}l}` says that the two columns of text should be separated by exactly one centimeter. A specification like `{@{}c@{}}` says that no additional space should be added neither of the left nor the right of the column. An `\extracolsep{wd}` command can be used inside such an expression. It causes an extra space to appear to the left of all subsequent columns. Note that `\extracolsep` expands to `\tabskip`; this  $\TeX$  primitive is not implemented in *Tralics*. In fact, *Tralics* ignores an ‘`@`’ and its argument.

You can use a `|` for specifying a vertical rule. However, in *Tralics* you cannot use double or triple rules. Sorry. There is also a `!{...}` options that is not implemented.

Every specification (‘`l`’, ‘`r`’, ‘`c`’, ‘`p`’, ‘`b`’, ‘`m`’) can be preceded by a `>{xx}` declaration, and followed by a `<{yy}` declaration. In case of multiple declarations, the last will be executed first.

Said otherwise, `>{3}>{b}c<{a}<{z}` is the same as `>{b3}c<{za}`. The effect is to insert ‘b3’ before the cell in the current position, and ‘za’ after the cell. See the last tabular in table 6.2. This corresponds to  $\langle u \rangle$  and  $\langle v \rangle$  parts of a T<sub>E</sub>X array. Note that the cell is finished when a token is sensed that indicates either a new cell, a new row or the end of the array. Technically, this means a `&`, a `\`, or an `\end` (the end of the environment). A special marker is pushed back after the ‘za’. This is a special endtemplate token in the case of a cell, and a `\cr` in the case of `\`. You should not use `\cr` or `\crrc` outside an array defined by `\halign` (this is not yet implemented). You must be careful that the ‘za’ (more generally, the  $\langle v \rangle$  part) does not contain something that reads the special end marker. For instance `\def\x#1{\halign{#\x&#\cr 1&2\cr}}` is an error. Finally, `*{N}{text}` can be used instead of N occurrences of ‘text’.

Note. At the end of Chapter 22 of the T<sub>E</sub>Xbook, Knuth gives an example of a table where the preamble is `\centerline{#}`. Such a construction cannot be done in Tralics, since a specification of the form `>{\centerline}c<{}` would transform into `\centerline???` and question marks cannot be replaced by braces; you could try `>{\expandafter\centerline?}` and replace the question mark by something that expands to an open brace but contains as many open braces as closing ones, for instance `\expandafter {\iffalse}\fi`. However, it is not possible to put in the `<{?}` part something that the parser considers as a closing brace followed by some other text (otherwise, this closing brace would terminate parsing of the `<{?}` part).

Knuth says that an entry of the form `a}b{c` is legitimate, with respect to this template. This cannot be the case in Tralics, but it would be valid for a template like `>{\bgroup\bf}c<{\egroup}`. This justifies that a table has to be terminated by `\cr` or `\crrc`. In the case of Tralics, this is not needed.

## New column types

You can add new column types to the list of existing one, using `\newcolumnntype`, with as argument a letter. For instance:

```
\newcolumnntype{C}{>{\$}c<{\$}}
\newcolumnntype{L}{>{\$}l<{\$}}
\newcolumnntype{R}{>{\$}r<{\$}}
\newcolumnntype{d}[1]{>{\rightdots{#1}}r<{\endrightdots}}
\newcolumnntype{X}{CLR}
\begin{tabular}{*{3}{|c|}d{23}X}
\end{tabular}
```

In this case, the transcript file will contains (line breaks added before ‘r’)

```
{Push tabular 2}
array preamble at start: |c||c||c|d{23}X
array preamble after X: |c||c||c|d{23}CLR
array preamble after d: |c||c||c|>{\rightdots {23}}
                        r<{\endrightdots }CLR
array preamble after C: |c||c||c|>{\rightdots {23}}
                        r<{\endrightdots }>{\$}c<{\$}LR
array preamble after L: |c||c||c|>{\rightdots {23}}
                        r<{\endrightdots }>{\$}c<{\$}>{\$}l<{\$}R
array preamble after R: |c||c||c|>{\rightdots {23}}
                        r<{\endrightdots }>{\$}c<{\$}>{\$}l<{\$}>{\$}r<{\$}
array preamble parse: | c | | c | | c | >>{}
                        r <<{} >>{} c <<{} >>{} l <<{} >>{} r <<{}
```

Whenever a tabular is seen, optional arguments are read, and then the first argument is handled. In a first pass, `*` is evaluated. This gives the lines marked ‘at start’. After that, the preamble

contains, at toplevel (outside braces) two characters ‘d’ and ‘X’ that are defined to be new column types. These are evaluated one after the other (the order is irrelevant, here alphabetic order is used so that X is expanded first). Since the expansion was non trivial, a second try is made. Note that only a finite numbers of tries are executed. In case of recursion, strange things can happen. Note how you can use commands with arguments (here ‘d’ takes one argument, it is ‘23’).

The table is empty, on purpose, because there are two undefined macros, moreover, because, in the current version of `Tralics`, dollar signs have to be explicit, and not hidden in a `>{ }...<{ }` construction.

## Another example

We consider here the following new column types. As you can see, one of them is the character `+`, another is the character `_`. The fact that these characters have special catcodes is irrelevant (they cannot be of catcode 1 and 2, because this would interfere with brace matching, and they cannot be of catcode 10, because space characters should be ignored in the preamble).

```
\newcolumnntype{L} >{\large\bfseries 2}l <{y}|}
\newcolumnntype{+} >{B}l <{D}|}
\newcolumnntype_{_}{r|c<{x}}
\newcolumnntype{x}>{b}c<{a}}
```

Consider the four following tables

```
\begin{tabular*}{339pt}[b]{*{4}{_c|}}
a1&a2&a3&a4 & b1&b2&b3&b4 & c1&c2&c3&c4& d1&d2&d3&d4\\
Wa1&Wa2&Wa3&Wa4 & Wb1&Wb2&Wb3&Wb4 & Wc1&Wc2&Wc3&Wc4& Wd1&Wd2&Wd3&Wd4\\
\end{tabular*}
```

```
\begin{tabular}{|l|rr|cc|}
\hline a&b&c&d&e&f\\
aaa&bbb&ccc&ddd&eee&fff\\
\hline
A&\multicolumn{3}{+}{C}&E&F\\
\multicolumn{2}{|l}{ab}&c&d&e&f\\
\cline{1-3}\cline{6-6}
aaa&bbb&ccc&ddd &eee&fff\\
\end{tabular}
```

```
\begin{tabular} { | >{\large 1}c <{x}| L > {\large\itshape 3}c <{z}|}
\hline A&B&C\\
\hline 100&10 &1\\
\end{tabular}
```

```
\begin{tabular} { | >{\large 1}c <{x}| L > {\large\itshape 3}x <{z}|}
\hline A&B&C\\
\hline 100&10 &1
\end{tabular}
```

You can see the  $\text{\LaTeX}$  result on table 6.2. We had to change the `\tabcolsep` of the first table to 0, otherwise, it is two wide. Specifying 0pt as width gives the following warning : *Overfull \hbox (339.38422pt too wide) in alignment*, from this we deduced that 339pt could be a good value, specifying 340 gives an underful box. The XML translation can be found on the Web page. In order to explain what happens, we consider an example:

```
\begin{tabular*}{10pc}[b]{1>{x}r<{y}c}
\hline
a&b&c\\
\end{tabular}
```

```
a1a2 a3x a4 | b1b2 b3x b4 | c1c2 c3x c4 | d1d2 d3x d4 |
Wa1Wa2Wa3xWa4|Wb1Wb2Wb3xWb4|Wc1Wc2Wc3xWc4|Wd1Wd2Wd3xWd4|
```

a	b	c	d	e	f
aaa	bbb	ccc	ddd	eee	fff
A	BCD			E	F
ab		c	d	e	f
aaa	bbb	ccc	ddd	eee	fff

1Ax	<b>2By</b>	3Cz
1100x	<b>210y</b>	31z

1Ax	<b>2By</b>	b3Cza
1100x	<b>210y</b>	b31za

Table 6.2: Some L<sup>A</sup>T<sub>E</sub>X tables

```
a&\omit b&c\\
\multicolumn{1}{1}{A}&B&C\\ \hline
\end{tabular*}
```

We explain now the translation. Line numbers refer to the transcript file given below. We do not show the start of the job (initialization). The command `\par` is redefined to do nothing. It is restored on line 107.

- A first procedure is used to start a row. See example, lines 1, 35, 63, 101. It can take an argument, that explains that some vertical spacing should be added between rows, this is the case for line 35. Tokens are read and expanded. Spaces are ignored. If the command that comes next is `\hline`, for instance lines 1 and 101, we add a bottom-border attribute to the previous row (if we are about to start the first row, we mark this as top-border for the current row). If the command is `\cline`, see syntax above, we try to add a bottom-border attribute to some cells (this is not shown on the example). This could fail, because cells can span more than one column; in this case an row of empty cells may be added. If the next token is `\end`, we do nothing (line 101). Otherwise, we create a new `\row` element (lines 3, 36, and 63) and start a cell.
- When a cell is started, the equivalent of `\begingroup` is evaluated, and a `\cell` element is started. The example table has 9 cells, started on lines 4, 12, 22, 37, 45, 53, 64, 82, 92. The ‘Push’ line indicates that a `<cell>` element is created, the line that follows show the `\begingroup` command. We consider the first non space token after expansion. This token may be `\omit`, as on line 47. The cell is marked omitted; otherwise the `<u>` part of the current template is added. The current token could be `\multicolumn`. This is a command that takes 3 arguments: an integer  $n$ , a specification  $c$ , a value  $v$ . The example on lines 66 to 69 shows how  $n$  is read; this might change, L<sup>A</sup>T<sub>E</sub>X uses the equivalent of `\setcounter` for parsing the number. The specification is handled in the same fashion as an array preamble (lines 70-71), it should provide only one cell specification. The integer  $n$  specifies the span, it will be added as an attribute to the cell, but also used to find the position of the next cell in the row.

The cell is marked ‘omit’; in fact, we push in the current input stream the value of the `<u>` part, `<v>` part and content (last argument) of `\multicolumn`. Braces are added for some obscure reason (see lines 72 and 73).

- Translation proceeds as usual, three events may change it. Assume that we see `&`, as on lines 7, 15, 40, 77, 85. This is only allowed in a table. A final space in the XML element will be removed. A `\endtemplate` token is pushed to be read again. It will be read again on lines 9, 18, 42, 79, and 88. If the cell on the stack is not marked ‘omit’ then the `<v>` part of the template is inserted in the input stream (see lines 17 and 87).
- The translation of `\endtemplate` is trivial: it is an error if not in a cell. Otherwise, the current cell is finished, the equivalent of `\endgroup` is executed, and a new cell is started as above. On line 10 you can see for instance that the cell and the row are in ‘array’ mode, the tabular in horizontal mode, the paragraph and the document in vertical mode.
- Assume that we see `\\`, for instance lines 25, 56, 95. We have explained the meaning of this command outside a cell. In any case, an optional argument is read (between the command and the optional argument can be a space or a newline character, so that source line 10 is read; see line 58). The behavior is like `&`, except that `\cr` is pushed instead of `\endtemplate`. (in the case of an optional argument, the code is a bit hacked, see line 30).
- The translation of `\cr` is trivial. We have two versions: one that reads a dimension, and one that does not. On lines 27-29, you can see how ‘scanglue’ reads to optional argument to `\\`, on line 31, you see that ‘scanint’ reads 1703, this is the address in a table of the string that represents the glue. The command finishes the cell, finishes the row, evaluates `\endgroup` and starts a new row.
- The `\end` command, with argument ‘tabular’ or ‘tabular\*’ behaves in a strange manner: If inside a cell, it behaves like `\\`, except that it does not insert `\endtemplate` but itself with the argument, then a `\cr`. We have seen that `\cr` starts a new row. We have seen that no row is started if the token is `\end`. The magic is that the top-stack contains the ‘begin’, and a normal ‘end’ can be used, instead of a hacked one. In our example, the array is terminated by `\\hline`, so that the end occurs when we are about to start a new row. In order to show this, we have commented out this line, and re-run the example. The result can be seen on line 111, a `\cr` is inserted and evaluated on line 115, the second `\end` is evaluated on line 19.
- The commands `\cr`, `\crrc`, `\span`, `\halign`, `\valign`, `\noalign` are not implemented as described in the `TEXbook`.

This is the transcript file.

```

1 [7] \hline
2 [8] a&b&c\\[2pt]
3 {Push row 3}
4 {Push cell 4}
5 +stack: level + 3 for cell
6 Character sequence: a.
7 {alignment tab character &}
8 {Text:a}
9 {\endtemplate}
10 {Pop 4: document_v p_v tabular*_h row_a cell_a}
11 +stack: level - 3 for cell
12 {Push cell 4}
13 +stack: level + 3 for cell
14 Character sequence: xb.
15 {alignment tab character &}
16 {Text:xb}
17 Character sequence: y.
18 {\endtemplate}
19 {Text:y}

```

```

20 {Pop 4: document_v p_v tabular*_h row_a cell_a}
21 +stack: level - 3 for cell
22 {Push cell 4}
23 +stack: level + 3 for cell
24 Character sequence: c.
25 {\}
26 {Text:c}
27 +scanint for \->2
28 +scandimen for \->2.0pt
29 {scanglue 2.0pt}
30 {\cr withargs}
31 +scanint for \cr withargs->1703
32 {Pop 4: document_v p_v tabular*_h row_a cell_a}
33 +stack: level - 3 for cell
34 {Pop 3: document_v p_v tabular*_h row_a}
35 [9] a&\omit b&c\
36 {Push row 3}
37 {Push cell 4}
38 +stack: level + 3 for cell
39 Character sequence: a.
40 {alignment tab character &}
41 {Text:a}
42 {\endtemplate}
43 {Pop 4: document_v p_v tabular*_h row_a cell_a}
44 +stack: level - 3 for cell
45 {Push cell 4}
46 +stack: level + 3 for cell
47 Character sequence: b.
48 {alignment tab character &}
49 {Text:b}
50 {\endtemplate}
51 {Pop 4: document_v p_v tabular*_h row_a cell_a}
52 +stack: level - 3 for cell
53 {Push cell 4}
54 +stack: level + 3 for cell
55 Character sequence: c.
56 {\}
57 {Text:c}
58 [10] \multicolumn{1}{1}{A}&B&C\\\hline
59 {\cr}
60 {Pop 4: document_v p_v tabular*_h row_a cell_a}
61 +stack: level - 3 for cell
62 {Pop 3: document_v p_v tabular*_h row_a}
63 {Push row 3}
64 {Push cell 4}
65 +stack: level + 3 for cell
66 {Push argument 5}
67 Character sequence: 1.
68 {Text:1}
69 {Pop 5: document_v p_v tabular*_h row_a cell_a argument_a}
70 array preamble at start: 1
71 array preamble parse: 1
72 {begin-group character {}}
73 +stack: level + 4 for brace
74 Character sequence: A.
75 {end-group character {}}
```

```

76 +stack: level - 4 for brace
77 {alignment tab character &}
78 {Text:A}
79 {\endtemplate}
80 {Pop 4: document_v p_v tabular*_h row_a cell_a}
81 +stack: level - 3 for cell
82 {Push cell 4}
83 +stack: level + 3 for cell
84 Character sequence: xB.
85 {alignment tab character &}
86 {Text:xB}
87 Character sequence: y.
88 {\endtemplate}
89 {Text:y}
90 {Pop 4: document_v p_v tabular*_h row_a cell_a}
91 +stack: level - 3 for cell
92 {Push cell 4}
93 +stack: level + 3 for cell
94 Character sequence: C.
95 {\}
96 {Text:C}
97 {\cr}
98 {Pop 4: document_v p_v tabular*_h row_a cell_a}
99 +stack: level - 3 for cell
100 {Pop 3: document_v p_v tabular*_h row_a}
101 [11] \end{tabular*}
102 {\end}
103 {\end tabular*}
104 {\endtabular*}
105 {Pop 2: document_v p_v tabular*_h}
106 {\endgroup (for env)}
107 +stack: restoring \par=\par
108 +stack: ending environment tabular*; resuming document.
109 +stack: level - 2 for environment
110 Character sequence: .

    Alternate version, where the final \\\hline is commented out
111 [11] \end{tabular*}
112 {\end}
113 {Text:C}
114 {\end tabular*}
115 {\cr}
116 {Pop 4: document_v p_v tabular*_h row_a cell_a}
117 +stack: level - 3 for cell
118 {Pop 3: document_v p_v tabular*_h row_a}
119 {\end}
120 {\end tabular*}
121 {\endtabular*}

```

## 6.10 Actions declared in the configuration file

An action is defined by a name, an equals sign, and a value. Optional spaces can be used. The syntax of DocType and DocAttrib is special. In all other cases, double quotes must delimit the value. All names contain only letters, digits, and underscores. If the name has the form `att_foo`, this changes the value of attribute 'foo'. If the name has the form `xml_foo`, it changes the value



of element ‘foo’. In a previous version you had to give the full name, `xml_foo_name`. Names of elements and attributes can be dynamically changed: when you say

```
\ChangeElementName{item}{Item}
\ChangeElementName*{rend}{rendering}
\ChangeElementName*{quote}{quotation}
```

this changes the name of element ‘item’ and attributes ‘rend’ and ‘quote’.

In the examples that follow, we shall assume that the file defined in section 6.3.6 is loaded.

- `makefo`, `makehtml`, `checkxml`, `makepdf`, `makedvi`, `dvitops`, `generatedvi`, `generateps`. These specify actions for the Raweb. For details see the Web page.
- `theme_vals` (Raweb only). This defines the list of all Inria Themes.
- `section_vals` (Raweb only). This defines the list of all valid sections in the Raweb. If the first character is a ‘+’ sign, the remaining of the line is appended to the previous value. You must use a slash as separator.
- `ur_vals` (Raweb only). This defines the list of all valid UR (name and value) in the Raweb. If the first character is a ‘+’ sign, the remaining of the line is appended to the previous value. You must use a slash as separator. After each name, you must give a value (if empty, the name will be used). Only letters can be used in the name.
- `affiliation_vals`, `profession_vals` (Raweb only). Same syntax as above. These lists specify which values are valid for argument 3 and 4 of the `\pers` command. The value ‘Other’ is always valid.
- `catperso_vals`, (Raweb only). Same syntax as above. These lists specify which values are valid for argument of the `\catperso` command. If no such declaration is given, any value is accepted.
- `Language`. The main element contains an attribute pair of the form `language=‘english’`. Specifying ‘Language’ in the configuration file modifies the name of the attribute.
- `lang_fr`, `lang_en`. See above. These two commands can parametrize the value of the attribute. The attribute pair is added to the main element: just after translation of the preamble in the Raweb case, by the `titlepage` command, or in the at-begin-document hook. The language chosen is the “default language”.
- `url_font`. If you specify a value ‘\foo’, then `\def\urlfont{\foo}` will be added to the document-hook. The command is empty by default. It is the font used for urls.
- `distinguish_refer_in_rabib` (Raweb only). This overrides the command line argument of the same name.
- `entity_names`. This overrides the command line option `entnames`.
- `bibtex_fields`. This specifies a list of additional BibT<sub>E</sub>X fields that Tralics should read and translate.
- `bibtex_extensions`. This specifies a list of additional BibT<sub>E</sub>X entry types that Tralics should read and translate.
- `everyjob`. If the value is ‘\foo’, then `\everyjob={\foo}` will be executed. In fact, this is the last line of the bootstrap code. After these lines are translated, the value of the token list `\everyjob` is inserted in the input stream, before the first line of the input file.

- `no_footnote_hack`. If the value is ‘true’, then no hack is applied to footnotes. The translation of

```
a\footnote{B}\footnote{C\par D}
```

is

```
<p>a<note id='uid1' place='foot'>B</note>
<note id='uid2' place='foot'><p>C</p> <p>D</p>
</note></p>
```

As you can see, if the footnote holds only a single `<p>` element, it will be removed. If the switch is true, then all footnotes translate the same.

- `xml_footnote_name`, `att_place`, `att_foot_position`. These assignments explain how to change the name of the `<note>` element, the attribute name and the attribute value in ‘place=’foot’. Thus, the translation of the example above can be:

```
<p>a<Note id='uid12' Place='Inline'><p>B</p>
</Note><Note id='uid13' Place='Inline'><p>C</p>
<p>D</p>
</Note>
```

- `use_font_elt`. The default translation of `{\tiny \textit{A}B}` is

```
<hi rend='small'><hi rend='it'>A</hi></hi><hi rend='small'>B</hi>
```

If you say ‘`use_font_elt=true`’, this changes the translation as follows

```
<small><it>A</it></small><small>B</small>
```

We shall see below how to replace ‘small’ by something else. the method is the same whether ‘small’ is the name of a an element or an attribute value.

- `use_all_sizes`. By default, Tralics knows only three font sizes, normal, smaller, larger. If the switch is true, the previous example translates to

```
<small4><it>A</it></small4><small4>B</small4>
```

or

```
<hi rend='small4'><hi rend='it'>A</hi></hi><hi rend='small4'>B</hi>
```

- `xml_font_small`, `xml_font_large`, `xml_font_normalsize`. These elements specify the names of small, large and normal size. The translation of `{\tiny a}{\normalsize b}{\large c}` can be changed to

```
<Small>a</Small>b<Large>c</Large>
```

Currently, no tag is inserted in the case of `\normalsize`. In some cases, this is wrong.

- `xml_font_small1`, `xml_font_small2`, `xml_font_small3`, `xml_font_small4`, `xml_font_large1`, `xml_font_large2`, `xml_font_large3`, `xml_font_large4`, `xml_font_large5`. These specify the names of all font sizes for the case where all sizes are used. For instance, the translation of

```
{\Huge a}{\huge b}{\LARGE c}{\Large d}{\large e}{\normalsize f}{\small
g}{\footnotesize h}{\scriptsize i}{\tiny j}
```

is

```
<font-large5>a</font-large5><font-large4>b</font-large4>
<font-large3>c</font-large3><font-large2>d</font-large2>
<font-large1>e</font-large1>f<font-small1>g</font-small1>
<font-small2>h</font-small2><font-small3>i</font-small3>
<font-small4>j</font-small4>
```

It can be changed to

```
<hi rend='font-large5'>a</hi><hi rend='font-large4'>b</hi>
<hi rend='font-large3'>c</hi><hi rend='font-large2'>d</hi>
<hi rend='font-large1'>e</hi>f<hi rend='font-small1'>g</hi>
<hi rend='font-small2'>h</hi><hi rend='font-small3'>i</hi>
<hi rend='font-small4'>j</hi>
```

- `xml_font_upright`, `xml_font_it`, `xml_font_slanted`, `xml_font_sc`. These four parameters specify the shape. The first one is currently unused. The default translation of

```
{\upshape a}{\itshape b}{\slshape c}{\scshape d}
```

is

```
a<hi rend='it'>b</hi><hi rend='slanted'>c</hi><hi rend='sc'>d</hi>
```

or (using elements):

```
a<it>b</it><slanted>c</slanted><sc>d</sc>
```

You can change it to:

```
a<font-italic-shape>b</font-italic-shape>
<font-slanted-shape>c</font-slanted-shape>
<font-small-caps-shape>d</font-small-caps-shape>
```

- `xml_font_medium`, `xml_font_bold`. These two parameters specify the series. The first is currently unused.
- `xml_font_roman`, `xml_font_tt`, `xml_font_sansserif`. These three parameters specify the family. The first is currently unused. The translation of

```
{\mdseries e}{\bfseries f}{\ttfamily h}{\sffamily h}{\rmfamily i}
```

is

```
e<hi rend='bold'>f</hi><hi rend='tt'>h</hi><hi rend='sansserif'>h</hi>i
```

but you can change it to

```
e<bold>f</bold><tt>h</tt><sansserif>h</sansserif>i
```

or

```
e<font-bold-series>f</font-bold-series>
<font-typewriter-family>h</font-typewriter-family>
<font-sansserif-family>h</font-sansserif-family>i
```

- `xml_sup_name`, `xml_sub_name`. These two parameters control the name of superscript and subscripts in text mode.
- `xml_oldstyle_name`, `xml_overline_name`, `xml_underline_name`. These parameters control the names used for oldstyle numbers, overline and underline.
- `xml_caps_name`, `xml_ul_name`, `xml_hl_name`, `xml_so_name`, `xml_st_name`. These control element names and attributes defined by the soul package. The translation of

```
\ul{a}\caps{b}\hl{c}\so{d}\st{e}
\textsuperscript{f}\textsubscript{g}
\oldstylenums{h}\overline{i}\underline{j}
```

is by default

```

<hi rend='ul'>a</hi><hi rend='caps'>b</hi><hi rend='hl'>c</hi>
<hi rend='so'>d</hi><hi rend='st'>e</hi>
<hi rend='sup'>f</hi><hi rend='sub'>g</hi>
<hi rend='oldstyle'>h</hi><hi rend='overline'>i</hi>
<hi rend='underline'>j</hi>

```

but can be changed to

```

<font-ul>a</font-ul><font-caps>b</font-caps><font-hl>c</font-hl>
<font-so>d</font-so><font-st>e</font-st>
<font-super>f</font-super><font-sub>g</font-sub>
<font-oldstyle>h</font-oldstyle><font-overline>i</font-overline>
<font-underline>j</font-underline>

```

- `alternate_item`. If true or false, it redefines `\item` to `\@item` or `\@@item`. The translation of

```

\begin{itemize}
\makeatletter
\@item [A]b
\@@item [A]b
\end{itemize}

```

is

```

<List type='simple'>
<Item id='uid14' Label='A'><p Noindent='true'><b></p>
</Item><Label>A</Label>
<Item id='uid15'><p Noindent='true'><b></p>
</Item></List>

```

- `xml_labelitem_name`. This specifies the name of the element containing the optional argument of an `\item`. Default value is 'label'. If alternate items are used, it specifies the names of an attribute.
- `xml_item_name`. This specifies the name of the element containing an `\item`. Default value is 'item'.
- `xml_list_name`. This is the name of the element generated by list environments like `itemize`, `enumerate` and `description`.
- `att_user_list`. List defined by the 'list' environment are by default of type 'description'.

Example

```

\begin{list}{}{}
\item[item label] item value
\begin{description}
\item[first item] okok
\item[second item]
\begin{itemize}
\item First
\item Second
\begin{enumerate}
\item some item
\item another one
\end{enumerate}
\end{itemize}
\end{description}
\end{list}

```

This translates by default to

```
<list type='description'><label>item label</label>
<item id='uid3'><p noindent='true'>item value</p>
<list type='description'><label>first item</label>
<item id='uid4'><p noindent='true'>okok</p>
</item><label>second item</label>
<item id='uid5'>
<list type='simple'>
<item id='uid6'><p noindent='true'>First</p>
</item>
<item id='uid7'><p noindent='true'>Second</p>
<list type='ordered'>
<item id='uid8'><p noindent='true'>some item</p>
</item>
<item id='uid9'><p noindent='true'>another one</p>
</item></list>
</item></list>
</item></list>
</item></list>
</item></list>
```

It can be changed to:

```
<List type='MyList'>
<Item id='uid16' Label='item label'><p Noindent='true'>item value</p>
<List type='description'>
<Item id='uid17' Label='first item'><p Noindent='true'>okok</p>
</Item>
<Item id='uid18' Label='second item'>
<List type='simple'>
<Item id='uid19'><p Noindent='true'>First</p>
</Item>
<Item id='uid20'><p Noindent='true'>Second</p>
<List type='ordered'>
<Item id='uid21'><p Noindent='true'>some item</p>
</Item>
<Item id='uid22'><p Noindent='true'>another one</p>
</Item></List>
</Item></List>
</Item></List>
</Item></List>
```

- `xml_gloitem_name`. This specifies the name of the element containing the first argument of `\glo`. Default value is 'label'.
- `xml_glo_name`. This controls the name of the glossary.
- `att_gloss_type`. In a glossary, a `<list>` element is created with attribute `type='gloss'`. This variable can be used to change the value of the attribute. The translation of

```
\begin{glossaire}
\glo{aa}{bb}
\glo{cc}{dd}
\end{glossaire}
```

can be

```

<List type='Gloss'>
  <Head>The famous glossary</Head>
  <Glolabel>aa</Glolabel><Item><p>bb</p></Item>
  <Glolabel>cc</Glolabel><Item><p>dd</p></Item>
</List>

```

- `xml_head_name`. This is the name of the `<head>` element in a title.
- `att_nonnumber`. This is the value of the 'rend' attribute for unnumbered sections.
- `xml_frontmatter_name`, `xml_mainmatter_name`, `xml_backmatter_name`. These are the name of elements associated to frontmatter, mainmatter and backmatter.
- `xml_div0_name`, `xml_div1_name`, `xml_div2_name`, `xml_div3_name`, `xml_div4_name`, `xml_div5_name`, `xml_div6_name`. This specifies the name of a section. The default translation of

```

\frontmatter
\chapter{Chapter one} OK
\mainmatter
\part{First Part of document}
Some text here
\part{Second Part}
\chapter{Chapter two}
\section{Section one}
\subsection{Subsection one}
\subsubsection{Subsubsection one}
\paragraph{First paragraph}
\subparagraph{First sub paragraph}
Normal text\label{a}
\backmatter
We are now in the backmatter.\ref{a}\pageref{a}

```

is

```

<frontmatter>
<div1 id='uid1' rend='nonnumber'><head>Chapter one</head>
<p>OK</p>
</div1></frontmatter>
<mainmatter>
<div0 id='uid2'><head>First Part of document</head>
<p>Some text here</p>
</div0>
<div0 id='uid3'><head>Second Part</head>
<div1 id='uid4'><head>Chapter two</head>
<div2 id='uid5'><head>Section one</head>
<div3 id='uid6'><head>Subsection one</head>
<div4 id='uid7'><head>Subsubsection one</head>
<div5 id='uid8'><head>First paragraph</head>
<div6 id='uid9'><head>First sub paragraph</head>
<p>Normal text</p>
</div6></div5></div4></div3></div2></div1></div0></mainmatter>
<backmatter><p>We are now in the backmatter.
<ref target='uid9'><ref target='uid9' rend='page' /></p>
</backmatter>

```

but you can use these commands to get:

```
<Frontmatter>
<Xdiv1 id='uid1' Rend='NoNumber'><Head>Chapter one</Head>
<p>OK</p>
</Xdiv1></Frontmatter>
<Mainmatter>
<Xdiv0 id='uid2'><Head>First Part of document</Head>
<p>Some text here</p>
</Xdiv0>
<Xdiv0 id='uid3'><Head>Second Part</Head>
<Xdiv1 id='uid4'><Head>Chapter two</Head>
<Xdiv2 id='uid5'><Head>Section one</Head>
<Xdiv3 id='uid6'><Head>Subsection one</Head>
<Xdiv4 id='uid7'><Head>Subsubsection one</Head>
<Xdiv5 id='uid8'><Head>First paragraph</Head>
<Xdiv6 id='uid9'><Head>First sub paragraph</Head>
<p>Normal text</p>
</Xdiv6></Xdiv5></Xdiv4></Xdiv3></Xdiv2></Xdiv1></Xdiv0></Mainmatter>
<Backmatter><p>We are now in the backmatter.
<Ref target='uid9'><Ref target='uid9' Rend='Page' /></p>
</Backmatter>
```

- `xml_fbox_name`, `xml_scalebox_name`, `xml_box_name`. These control the names of various box commands.
- `att_boxed`. This is the name of the attributed added by `\fbox`, unless it is a `includegraphics`.
- `att_framed`. If a figure is in a `\fbox`, it will have `framed='true'`. This changes the name of the attribute.
- `att_box_width`. Name of width attribute for commands like `\framebox`.
- `att_box_scale`. This controls the name scale attribute, unless in a figure. Example

`\fbox{foo}\scalebox{0.3}{foo}\framebox(10,10){foo}\framebox[10pt][11]{foo}`  
translates as

```
<fbox rend='boxed'>foo</fbox>
<scalebox scale='0.3'>foo</scalebox>
<pic-framebox width='10' height='10' framed='true'>foo</pic-framebox>
<fbox width='10.0pt' rend='boxed'>foo</fbox>
```

You can change it to

```
<Framebox Rendering='Boxed'>foo</Framebox>
<Scalebox BoxScale='0.3'>foo</Scalebox>
<Xbox Width='10' Height='10' Framed='true'>foo</Xbox>
<Framebox BoxWidth='10.0pt' Rendering='Boxed'>foo</Framebox>
```

- `xml_keywords_name`. This is the name of the element generated by `'motscle'` environment. The default value is `'keywords'`.
- `xml_term_name`. This is the name of the element used for each keyword. The default value is `'term'`. For instance,

```
\begin{motscle} first, second, {(max,+)}\end{motscle}
```

translates to

```

<keywords>
<term>first</term>
<term>second</term>
<term>(max,+)</term>
</keywords>

```

and can be changed to

```

<Keywords>
<Term>first</Term>
<Term>second</Term>
<Term>(max,+)</Term>
</Keywords>

```

- `xml_scaption_name`. Special caption (outside figure or table).
- `xml_mbox_name`. This is the name of the element used by the `\mbox`, unless the result is trivial.
- `xml_rotatebox_name`. This is the name of the element generated by `\rotatebox`.
- `att_rotate_angle`. This controls the name of the `angle` attribute in the case of `\rotatebox`.
- `xml_subfigure_name`. This is the name of the element generated by `\subfigure`. The default value is 'subfigure'.
- `xml_texte_name`. This is the name of the element generated by `\subfigure`, containing the required argument.
- `xml_leg_name`. This is the name of the element generated by `\subfigure`, containing the optional argument.

For instance, the translation of

```

\noindent \mbox{\xbox{a}{b}} \rotatebox{30}{x}
\subfigure[a]{b}

```

is

```

<p noindent='true'>
  <mbox><a>b</a></mbox>
  <pic-rotatebox angle='30'>x</pic-rotatebox>
  <subfigure><leg>a</leg><texte>b</texte></subfigure>
</p>

```

but can be changed to

```

<p Noindent='true'>
  <Xmbox><a>b</a></Xmbox>
  <Rotatebox Rangle='30'>x</Rotatebox>
  <Xsubfigure><Leg>a</Leg><Texte>b</Texte></Xsubfigure>
</p>

```

- `xml_topic_name`, `xml_topic_title`, `att_topic_num`. These are used for topics (Raweb only). The default values for the elements are 'topic' and 't\_titre', for the attribute it is 'num'.
- `xml_caption_name`, `xml_graphics_name`, `xml_figure_env_name`, `xml_table_env_name`, `xml_Table_name`. These names are used by some commands and environments. For instance, consider the following code. Normally, you put a tabular in a table, an image in a figure (as in B or D), this gives smaller XML objects.



```

\begin{tabular}{c}a\end{tabular}
\caption{A}
%
\begin{table}[htbp]
\begin{tabular}{c}a\end{tabular}
\caption{B}
\end{table}
%
\begin{table}[htbp]
\includegraphics{a}
\caption{C}
\end{table}
%
\begin{figure}[htbp]
\includegraphics{a}
\caption{D}
\end{figure}

```

This is the default translation:

```

<table rend='inline'><row><cell halign='center'>a</cell></row></table>
<p><caption>A</caption></p>
<table rend='display' id='uid10'><head>B</head>
  <row><cell halign='center'>a</cell></row></table>
<table rend='display' id='uid11'><head>C</head>
  <figure rend='inline' file='a' /></table>
<figure file='a' id='uid12'><head>D</head></figure>

```

This shows how all names can be parametrized.

```

<Table Rend='inline'><Row><Cell Halign='Center'>a</Cell></Row></Table>

<p><SCaption><p>A</p></SCaption></p>

<FloatTable Rend='display' id='uid14'>
  <Caption><p>B</p></Caption>
  <Row><Cell Halign='Center'>a</Cell></Row>
</FloatTable>

<FloatTable Rend='display' id='uid15'>
  <Caption><p>C</p></Caption>
  <Figure Rend='inline' File='a' />
</FloatTable>

<FloatFigure File='a' id='uid16'>
  <Caption><p>D</p></Caption>
</FloatFigure>

```

- `xml_xref_name`. This is the name of the element associated to `\url`.
- `xml_project_name`. This controls the name of the `<projet>` element.
- `xml_accueil_name`. This controls the name of the `<accueil>` element.
- `xml_composition_ra_name`. This controls the name of the `<composition>` element. In fact, in Raweb mode, when a environment 'RAsection' is closed, and the argument of the environment is the value of this variable, then modules inside the element are replaced by their content.

- `xml_xtheorem_name`. This controls the theorem names. This is an example of theorems

```

\theorembodyfont{\sl}
\theoremstyle{break}
\newtheorem{Cor}{Corollary}
\theoremstyle{plain}
%\newcounter{section}
\setcounter{section}{17}
\newtheorem{Exa}{Example}[section]
{\theorembodyfont{\rmfamily}\newtheorem{Rem}{Remark}}
\theoremstyle{marginbreak}
\newtheorem{Lem}[Cor]{lemma}
\theoremstyle{change}
\theorembodyfont{\small\itshape} \newtheorem{Def}[Cor]{Definition}
\theoremheaderfont{\scshape}
\def\Lenv#1{\texttt{#1}}

\begin{Cor}
  This is a sentence typeset in the theorem environment \Lenv{Cor}.
\end{Cor}
\begin{Exa}
  This is a sentence typeset in the theorem environment \Lenv{Exa}.
\end{Exa}
\begin{Rem}
  This is a sentence typeset in the theorem environment \Lenv{Rem}.
\end{Rem}
\begin{Lem}[Ben User]
  This is a sentence typeset in the theorem environment \Lenv{Lem}.
\end{Lem}
\begin{Def}[Very Impressive definition]
  This is a sentence typeset in the theorem environment \Lenv{Def}.
\end{Def}

```

This is the translation:

```

<p><hi rend='sc'>Corollary 1 </hi><hi rend='slanted'>This
is a sentence typeset in the theorem environment </hi>
<hi rend='slanted'><hi rend='tt'>Cor</hi></hi><hi rend='slanted'>.</hi></p>

<p><hi rend='sc'>Example 17.1 </hi><hi rend='slanted'>This
is a sentence typeset in the theorem environment </hi>
<hi rend='slanted'><hi rend='tt'>Exa</hi></hi><hi rend='slanted'>.</hi></p>

<p><hi rend='sc'>Remark 1 </hi>This is a sentence typeset in
the theorem environment <hi rend='tt'>Rem</hi>.</p>

<p><hi rend='sc'>lemma 2 (Ben User) </hi><hi rend='slanted'>
This is a sentence typeset in the theorem environment </hi><hi
rend='slanted'><hi rend='tt'>Lem</hi></hi><hi rend='slanted'>.</hi></p>

<p><hi rend='sc'>Definition 3 (Very Impressive definition) </hi><hi
rend='small' /><hi rend='small'><hi rend='it'>
This is a sentence typeset in the theorem environment </hi></hi>

```

```
<hi rend='small'><hi rend='it'><hi rend='tt'>Def</hi></hi></hi>
<hi rend='small'><hi rend='it'>.</hi></hi></p>
```

This may be changed to:

```
<BTheorem id='uid29'><p><SC>Corollary 1 </SC>
<SL>This is a sentence typeset in the theorem environment </SL>
<SL><TT>Cor</TT></SL><SL>.</SL></p>
</BTheorem>
<BTheorem id='uid30'><p><SC>Example 17.1 </SC>
<SL>This is a sentence typeset in the theorem environment </SL>
<SL><TT>Exa</TT></SL><SL>.</SL></p>
</BTheorem>
<BTheorem id='uid31'><p><SC>Remark 1 </SC>
This is a sentence typeset in the theorem environment <TT>Rem</TT>.</p>
</BTheorem>
<BTheorem id='uid32'><p><SC>lemma 2 (Ben User) </SC><SL>
This is a sentence typeset in the theorem environment
</SL><SL><TT>Lem</TT></SL><SL>.</SL></p>
</BTheorem>
<BTheorem id='uid33'><p><SC>Definition 3 (Very Impressive definition) </SC>
<Small/><Small><IT>
This is a sentence typeset in the theorem environment
</IT></Small><Small><IT><TT>Def</TT></IT></Small><Small><IT>.</IT></Small></p>
</BTheorem>
```

- `xml_theorem_head`. This controls the element that contains the optional argument of the theorem, if the alternate syntax is used (default is `alt_head`, shown as `AltHead` below).
- `xml_theorem_name`. This is used for theorems. The value of the variable is unused. Redefining the command changes the meaning of `\@begintheorem` to `\@xbegintheorem`. As a consequence the translation of the previous theorem is:

```
<BTheorem style='break' type='Cor' id='uid29'>
<Head>Corollary</Head>
<p>This is a sentence typeset in the theorem environment <TT>Cor</TT>.</p>
</BTheorem>
<BTheorem style='plain' type='Exa' id='uid30'>
<Head>Example</Head>
<p>This is a sentence typeset in the theorem environment <TT>Exa</TT>.</p>
</BTheorem>
<BTheorem style='plain' type='Rem' id='uid31'>
<Head>Remark</Head>
<p>This is a sentence typeset in the theorem environment <TT>Rem</TT>.</p>
</BTheorem>
<BTheorem style='marginbreak' type='Lem' id='uid32'>
<Head>lemma</Head>
<ThHead>Ben User</ThHead>
<p>This is a sentence typeset in the theorem environment <TT>Lem</TT>.</p>
</BTheorem>
<BTheorem style='change' type='Def' id='uid33'>
<Head>Definition</Head>
<ThHead>Very Impressive definition</ThHead>
<p>This is a sentence typeset in the theorem environment <TT>Def</TT>.</p>
</BTheorem>
```

- `mfenced_separator_val`. If the value of this quantity is 'X', then Tralics will add `separators = 'X'` to each `<mfenced>` that are created (except for fences for arrays, i.e., matrices and the like). The default value is the empty string. If the value is 'NONE', no attribute will be added. If the value is 'mrow', no attribute is added, but the `<mfence>` elements has a single child, a `<mrow>` element being inserted, if needed. Example

$$\$\left[ a + b \right] \left(x\right) \$$$

This is the translation if the separator is `&ThinSpace`;

```
<formula type='inline'>
  <math xmlns='http://www.w3.org/1998/Math/MathML'>
    <mrow>
      <mfenced separators='&ThinSpace;' open='[' close=']'>
        <mi>a</mi><mo>+</mo><mi>b</mi>
      </mfenced>
      <mfenced separators='&ThinSpace;' open='(' close=')'>
        <mi>x</mi>
      </mfenced>
    </mrow>
  </math>
</formula>
```

Translation if the value the separator is the empty string (this is the default behavior). No separator added in case where the `<mfenced>` element has a single child.

```
<formula type='inline'>
  <math xmlns='http://www.w3.org/1998/Math/MathML'>
    <mrow>
      <mfenced separators='' open='[' close=']'>
        <mi>a</mi><mo>+</mo><mi>b</mi>
      </mfenced>
      <mfenced open='(' close=')'><mi>x</mi></mfenced>
    </mrow>
  </math>
</formula>
```

Translation if the value the separator is 'mrow'. Here a `<mrow>` element is added.

```
<formula type='inline'>
  <math xmlns='http://www.w3.org/1998/Math/MathML'>
    <mrow>
      <mfenced open='[' close=']'>
        <mrow><mi>a</mi><mo>+</mo><mi>b</mi></mrow>
      </mfenced>
      <mfenced open='(' close=')'><mi>x</mi></mfenced>
    </mrow>
  </math>
</formula>
```

Translation if the value the separator is 'NONE'. This is the behaviour of Tralics before 2.9.4.

```
<formula type='inline'>
  <math xmlns='http://www.w3.org/1998/Math/MathML'>
    <mrow>
      <mfenced open='[' close=']'><mi>a</mi><mo>+</mo><mi>b</mi></mfenced>
      <mfenced open='(' close=')'><mi>x</mi></mfenced>
    </mrow>
```

```

    </math>
  </formula>

```

- `xml_biblio`. This controls the name of the element that contains the bibliography.
- `att_box_pos`. This controls the name of the `pos` attribute in a box.
- `att_full`. If a star follows `\circle`, an attribute pair `full='true'` is added. This controls the attribute name.
- `att_table_width`. Name of width attribute for a table.
- `att_pos`. Used by environments like 'minipage', for the horizontal position.
- `att_vpos`. Used by 'minipage', for the vertical position.
- `att_inner_pos`. Used by 'minipage', for the third optional argument.
- `att_minipage_width`. Used by 'minipage', for the mandatory argument.
- `xml_figure_name`. This controls the name of the element produced by `\includegraphics`.
- `att_file`, `att_angle`, `att_scale`, `att_clip`, `att_width`, `att_height`. These control the attribute names set by `\includegraphics`.

The translation of

```

\fbbox{\includegraphics[clip=, angle=90, scale=3, %
width=10cm, height=3m]{A}}

```

is

```

<figure framed='true' rend='inline' height='3m' width='10cm'
scale='3' angle='90' clip='true' file='A'/>

```

You can change it to

```

<Figure Framed='true' Rend='inline' Height='3m' Width='10cm' Scale='3'
Angle='90' Clip='true' File='A'/>

```

- `att_rend`, `att_fbox_rend`. These control the name of the `rend` attribute (why is 'fbox' a special case?)
- `att_noindent`. This is the name of the `noindent` attribute.
- `att_space_before`. This is the name of the attribute that indicates space between the current paragraph and the element before it.
- `att_flush_right`, `att_flush_left`, `att_quotation`, `att_quote`, `att_centering`. The five environments `center`, `quote`, `quotation`, `flushleft`, and `flushright` modify (locally) an integer that asks each `<p>` to get an attribute `rend` with value 'flushed-left', 'flushed-right', 'center' or 'quote'. These environment execute the equivalent of `\par` just after the `\begin` (before the parameter is changed), and just after the `\end`. The `\centering` command modifies the integer and the attribute list of the current `<p>`. These assignments show how to control the value of the attribute. Example:

```

a\[[2pt]b
\begin{center}A\end{center}
\begin{quote}B\end{quote}
\begin{quotation}C\end{quotation}
\begin{flushleft}D\end{flushleft}
\begin{flushright}E\end{flushright}

```

The default translation is

```
<p>a</p>
<p noindent='true' spacebefore='2.0pt'>b</p>
<p rend='center'>A</p>
<p rend='quoted'>B</p>
<p rend='quoted'>C</p>
<p rend='flushed-left'>D</p>
<p rend='flushed-right'>E</p>
```

but you can change it to

```
p>a</p>
<p Noindent='true' Spacebefore='2.0pt'>b</p>
<p Rend='Center'>A</p>
<p Rend='Quote'>B</p>
<p Rend='Quotation'>C</p>
<p Rend='FlushLeft'>D</p>
<p Rend='FlushRight'>E</p>
```

- `att_cell_bottomborder`, `att_cell_leftborder`, `att_cell_rightborder`, `att_cell_topborder`. Specifying a `\hline` in a table, or vertical rules may add attributes to cells. The name is `left-border` etc. The name of the attribute can be changed via these commands.
- `att_row_spaceafter`, `att_row_spacebefore`. Inside a table, an optional argument to `\` asks for a space before or after the current row. This is done by adding a `spacebefore` and `spaceafter` attribute. The name of the attribute can be changed via this variable. Note: There is no `spacebefore` in the current version; is this a bug?
- `att_cols`, `att_halign`, `att_cell_left`, `att_cell_right`, `att_cell_center`. The translation of a command of the form `\multicolumn{1}{c}{...}` is a cell, with attributes `halign='center'` and `cols='1'`. These parameters allow you to change the name of the attribute name or value.
- `xml_table_name`, `xml_row_name`, `xml_cell_name`. These commands allow you to change the names of the elements `<table>`, `<row>` and `<cell>`.

The default translation of

```
\begin{tabular}{|r|l|c}
\hline a&b&\multicolumn{1}{c}{x}\hline\ \ [2pt]
\end{tabular}
 $\begin{array}{|r|l|}
a&b&\multicolumn{1}{c}{x}\ \ [2pt]
\end{array}$ 
```

is

```
<table rend='inline'>
  <row spaceafter='2.0pt' top-border='true'>
    <cell right-border='true' halign='right' left-border='true'>a</cell>
    <cell right-border='true' halign='left'>b</cell>
    <cell right-border='true' halign='center' cols='1'>x</cell>
  </row>
</table>
<formula type='inline'>
  <math xmlns='http://www.w3.org/1998/Math/MathML'>
    <mtable>
      <mtr>
```

```

      <mtd columnalign='right'><mi>a</mi></mtd>
      <mtd columnalign='left'><mi>b</mi></mtd>
      <mtd colspan='1'><mi>x</mi></mtd>
    </mtr>
  </mtable>
</math>
</formula>

```

You can parametrize the `<table>`, using all parameters shown above. Of course, it has no influence on the `<mtable>`.

```

<Table Rend='Inline'>
  <Row Spaceafter='2.0pt' Topborder='true'>
    <Cell Rightborder='true' Halign='Right' Leftborder='true'>a</Cell>
    <Cell Rightborder='true' Halign='Left'>b</Cell>
    <Cell Rightborder='true' Halign='Center' Cols='1'>x</Cell>
  </Row>
</Table>

```

- `att_prenom`, `att_full_first`, `att_nom`, `att_particule`, `att_junior`, `xml_bpers_name`. These control typesetting of `\bpers` in a bibliography. The default translation of

```
\bpers[E]{A}{B}{C}{D}
```

is

```
<bpers prenom='A' part='B' nom='C' junior='D' prenomcomplet='E'/>
```

- `xml_pers_name`. This controls the element name of the `\pers` command.
- `att_affiliation`, `att_profession`, `att_HDR`. These control the attributes of the fields of the `\pers` command added in 2006 (first name and last name use `att_nom` and `att_prenom`).
- `xml_citation_name`. This is the name of the element produced by the `\citation` command. Since this command is assumed to appear only in a bibliography, it cannot be used directly in the text, thus the calls to `\@setmode` here. Example

```

\makeatletter {
\leavevmode\@setmode 4
\citation{A}{cite:B}{bid2}{D}{E}
\bauthors{\bpers {a}{c}{d}}
\beditors{\bpers[AA] {a}{b}{c}{d}}
\endcitation
\@setmode 1}\par

```

Translation

```

<p>
<Citation from='D' key='A' id='bid2' userid='cite:B' type='E'>
<bauteurs><Bpers Firstname='a' Lastname='c' Junior='d'/></bauteurs>
<bediteur>
  <Bpers Firstname='a'P article='b' Lastname='c' Junior='d' FullFirst='AA'/>
</bediteur>
</Citation></p>

```

- `xml_cit_name`, `xml_ref_name`. Translation of `\cite` is a `<ref>` in a `<cit>`; these switches control the element names.
- `xml_prenote_name`, `xml_citetype_name`. These control elements defined for the `natbib` extensions to `\cite`. For instance,

```

\def\cite@prenote{cf}
\def\cite@type{fx}
\cite[p 30]{toto}

```

translates as

```
<Cit Prenote='cf' Citetype='fx'><Ref target='bid0'>p 30</Ref></Cit>
```

- `xml_leaders_name`, `xml_cleaders_name`, `xml_xleaders_name`. For leaders. Example

```

\setbox0\hfil\copy0\hfil\cleaders\copy0\hfil\xleaders\copy0\hfil

```

Translation is

```

<Leaders><foo>bar</foo></Leaders><hfil/>
<Cleaders><foo>bar</foo></Cleaders><hfil/>
<Xleaders><foo>bar</foo></Xleaders><hfil/>

```

- `att_page`. Translation of `\ref` is a ref element, whose name can be changed as above, translation of `\pageref` is the same, with a attribute `rend`, with value 'page', that can be changed via this flag; for instance we may get

```
<Ref target='uid9' /><Ref target='uid9' Rend='Page' />
```

- `xml_tableofcontents_name`. Name of the TOC element.
- `xml_texmath_name`. Name of the element produced by the trivial math mechanism.
- `att_pre_name`. For verbatim environments.
- `xml_anchor_name`. Name of the anchor element.
- `xml_index_name`, `xml_theindex_name`. These variables indicated the name of the element produced by the `\index` command, and the name of the index (produced by `\makeindex`).
- `att_level`. An index has a level attribute (that can be 1, 2 or 3). This controls the name of the attribute.
- `att_encap`. Controls page encapsulation for the `\index` command. In the following example we have three anchors, because some `\index` commands are followed by a space (shown as newline in the XML document).

```

\noindent Word\index{foo}
\index{foo!a@bar|gee}
\index{foo!b@bara}%
\index{foo}

```

Translation.

```

<p Noindent='true'>Word<Anchor id='uid27' />
<Anchor id='uid28' />
<Anchor id='uid29' /></p>
...
<TheIndex>
  <Index target='uid27 uid29' Level='1'>foo</Index>
  <Index target='uid28' Level='2' Encap='gee'>bar</Index>
  <Index target='uid29' Level='2'>bara</Index>
</TheIndex>

```

- `xml_picture_name`. This can be used to change the name of the element created by the 'picture' environment (it is 'picture').



- `xml_bezier_name`. This specifies the name of the element produced by `\bezier`.
- `xml_put_name`. This specifies the name of the element produced by `\put`.
- `xml_arc_name`. This specifies the name of the element produced by `\arc`.
- `xml_scaleput_name`. This specifies the name of the element produced by `\scaleput`.
- `xml_multiput_name`. This specifies the name of the element produced by `\multiput`.
- `xml_line_name`. This specifies the name of the element produced by `\line`.
- `xml_lineC_name`. This specifies the name of the element produced by `\centerline` and friends.
- `xml_vector_name`. This specifies the name of the element produced by `\vector`.
- `xml_oval_name`. This specifies the name of the element produced by `\oval`.
- `xml_dashline_name`. This specifies the name of the element produced by `\dashline`.
- `xml_drawline_name`. This specifies the name of the element produced by `\drawline`.
- `xml_dottedline_name`. This specifies the name of the element produced by `\dottedline`.
- `xml_circle_name`. This specifies the name of the element produced by `\circle`.
- `xml_bigcircle_name`. This specifies the name of the element produced by `\bigcircle`.
- `xml_curve_name`. This specifies the name of the element produced by `\curve`.
- `xml_closecurve_name`. This specifies the name of the element produced by `\closecurve`.
- `xml_tagcurve_name`. This specifies the name of the element produced by `\tagcurve`.
- `xml_thicklines_name`. This specifies the name of the element produced by `\thicklines`.
- `xml_thinlines_name`. This specifies the name of the element produced by `\thinlines`.
- `xml_linethickness_name`. This specifies the name produced by `\linethickness`.
- `att_xpos`, `att_ypos`, `att_xdir`, `att_ydir`, `att_size`, `att_dx`, `att_dy`, `att_repeat`. These attributes are used by some picture commands.
- `att_xscale`, `att_xscaley`, `att_yscale`, `att_yscalex`. Commands `\xscale`, `\xscaley`, `\yscale`, and `\yscalex` read values and store them somewhere. Whenever needed, they are put as attribute on the elements. The default name of the argument is the name of the command, but this can be changed.
- `att_curve_nbpts`. Commands like `\arc`, `\bigcircle`, `\closecurve`, `\tagcurve`, `\curve` take an optional argument, the number of points. It will be stored in an attribute, named ‘nbsymb’. This parameter can change the name of the attribute.
- `att_unit_length`. Inside a picture environment, the value of `\unitlength` is stored in an attribute named ‘unit-length’. It can be changed using this command.

The example of a picture environment is

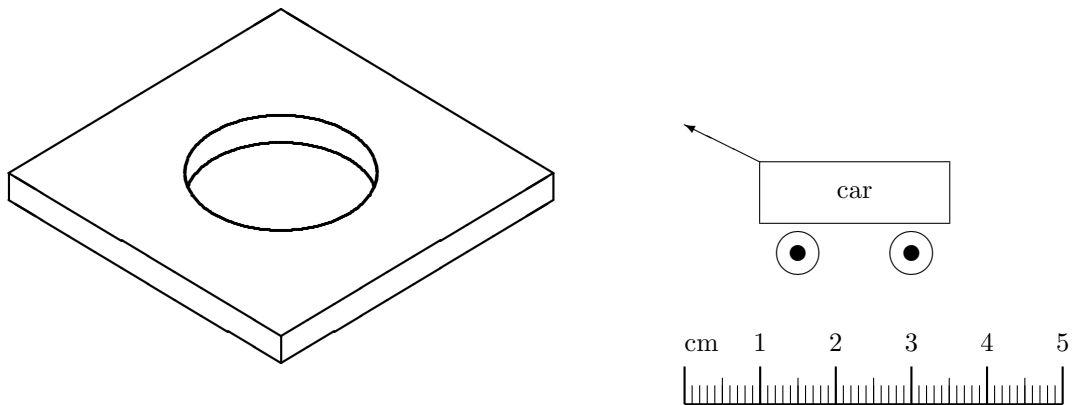


Figure 6.1: Two examples of a picture environment from [2]

```

\setlength{\unitlength}{1.8mm}
\begin{picture}(40,30)
  \thicklines
  \multiput(20,5)(20,12){2} {\line(0,-1){2}\line(-5,3){20}}
  \multiput(20,5)(-20,12){2} {\line(5,3){20}}
  \put(20,3){\line(5,3){20}}
  \put(20,3){\line(-5,3){20}}
  \put(0,15){\line(0,1){2}}
  \linethickness{1pt}
  \put(20,5) {
    \renewcommand{\xscale}{1}
    \renewcommand{\yscale}{-1}
    \renewcommand{\yscalex}{0.6}
    \renewcommand{\yscaley}{0.6}
    \scaleput(10,10){\bigcircle{10}}
    \put(0,-2){%
      \scaleput(10,10){\arc(5,0){121}}
      \scaleput(10,10){\arc(5,0){-31}}}}
\end{picture}

```

it is translated as

```

<picture width='204.85962' height='153.64471'>
  <pic-thicklines/>
  <pic-multiput xpos='102.42981' ypos='25.60745'
    repeat='2' dx='102.42981' dy='61.45789'>
    <pic-line xdir='0' ydir='-1' width='10.24298' />
    <pic-line xdir='-5' ydir='3' width='102.42981' />
  </pic-multiput>
  <pic-multiput xpos='102.42981' ypos='25.60745'
    repeat='2' dx='-102.42981' dy='61.45789'>
    <pic-line xdir='5' ydir='3' width='102.42981' />
  </pic-multiput>
  <pic-put xpos='102.42981' ypos='15.36447'>
    <pic-line xdir='5' ydir='3' width='102.42981' /></pic-put>
  <pic-put xpos='102.42981' ypos='15.36447'>
    <pic-line xdir='-5' ydir='3' width='102.42981' /></pic-put>
  <pic-put xpos='0' ypos='76.82236'>

```

```

    <pic-line xdir='0' ydir='1' width='10.24298' /></pic-put>
  <pic-linethickness size='1pt' />
  <pic-put xpos='102.42981' ypos='25.60745'>
    <pic-scaleput xpos='51.2149' ypos='51.2149'
      xscale='1' yscale='0.6' xscaley='-1' yscalex='0.6'>
      <pic-bigcircle size='10' unit-length='5.12149' />
    </pic-scaleput>
  <pic-put xpos='0' ypos='-10.24298'>
    <pic-scaleput xpos='51.2149' ypos='51.2149' xscale='1'
      yscale='0.6' xscaley='-1' yscalex='0.6'>
      <pic-arc angle='121' xpos='25.60745' ypos='0' unit-length='5.12149' />
    </pic-scaleput>
  <pic-scaleput xpos='51.2149' ypos='51.2149' xscale='1'
    yscale='0.6' xscaley='-1' yscalex='0.6'>
    <pic-arc angle='-31' xpos='25.60745' ypos='0' unit-length='5.12149' />
  </pic-scaleput>
</pic-put>
</pic-put>
</picture>

```

You can parametrize it so as to obtain this

```

<Xpicture Width='204.85962' Height='153.64471'>
  <Thicklines/>
  <Multipt Xpos='102.42981' Ypos='25.60745'
    Repeat='2' Dx='102.42981' Dy='61.45789'>
    <Xline XDir='0' YDir='-1' Width='10.24298' />
    <Xline XDir='-5' YDir='3' Width='102.42981' />
  </Multipt>
  <Multipt Xpos='102.42981' Ypos='25.60745'
    Repeat='2' Dx='-102.42981' Dy='61.45789'>
    <Xline XDir='5' YDir='3' Width='102.42981' />
  </Multipt>
  <Xput Xpos='102.42981' Ypos='15.36447'>
    <Xline XDir='5' YDir='3' Width='102.42981' /></Xput>
  <Xput Xpos='102.42981' Ypos='15.36447'>
    <Xline XDir='-5' YDir='3' Width='102.42981' /></Xput>
  <Xput Xpos='0' Ypos='76.82236'>
    <Xline XDir='0' YDir='1' Width='10.24298' /></Xput>
  <Linethickness Size='1pt' />
  <Xput Xpos='102.42981' Ypos='25.60745'>
    <Scaleput Xpos='51.2149' Ypos='51.2149'
      Xscale='1' Yscale='0.6' XscaleY='-1' YscaleX='0.6'>
      <Bigcircle Size='10' Unitlength='5.12149' />
    </Scaleput>
  <Xput Xpos='0' Ypos='-10.24298'>
    <Scaleput Xpos='51.2149' Ypos='51.2149'
      Xscale='1' Yscale='0.6' XscaleY='-1' YscaleX='0.6'>
      <Arc Angle='121' Xpos='25.60745' Ypos='0' Unitlength='5.12149' />
    </Scaleput>
    <Scaleput Xpos='51.2149' Ypos='51.2149'
      Xscale='1' Yscale='0.6' XscaleY='-1' YscaleX='0.6'>
      <Arc Angle='-31' Xpos='25.60745' Ypos='0' Unitlength='5.12149' />
    </Scaleput>
  </Xput>
</Xput>
</Xpicture>

```

A second example:

```

\newcounter{cms}
\setlength{\unitlength}{1mm}
\begin{picture}(50,39)
\put(0,7){\makebox(0,0)[bl]{cm}}
\multiput*(10,7)(10,0){5}{% remove the * for latex
  \addtocounter{cms}{1}\makebox(0,0)[b]{\arabic{cms}}}
\put(15,20){\circle{6}}
\put(30,20){\circle{6}}
\put(15,20){\circle*{2}}
\put(30,20){\circle*{2}}
\put(10,24){\framebox(25,8){car}}
\put(10,32){\vector(-2,1){10}}
\multiput(1,0)(1,0){49}{\line(0,1){2.5}}
\multiput(5,0)(10,0){5}{\line(0,1){3.5}}
\thicklines
\put(0,0){\line(1,0){50}}
\multiput(0,0)(10,0){6}{\line(0,1){5}}
\end{picture}

```

The translation is

```

<Xpicture Width='142.26303' Height='110.96516'>
  <Xput Xpos='0' Ypos='19.91682'>
    <Xbox Width='0' Height='0' BPos='bl'>cm</Xbox></Xput>
  <Xput Xpos='28.4526' Ypos='19.91682'>
    <Xbox Width='0' Height='0' BPos='b'>1</Xbox></Xput>
  <Xput Xpos='56.90521' Ypos='19.91682'>
    <Xbox Width='0' Height='0' BPos='b'>2</Xbox></Xput>
  <Xput Xpos='85.35782' Ypos='19.91682'>
    <Xbox Width='0' Height='0' BPos='b'>3</Xbox></Xput>
  <Xput Xpos='113.81042' Ypos='19.91682'>
    <Xbox Width='0' Height='0' BPos='b'>4</Xbox></Xput>
  <Xput Xpos='142.26303' Ypos='19.91682'>
    <Xbox Width='0' Height='0' BPos='b'>5</Xbox></Xput>
  <Xput Xpos='42.67891' Ypos='56.90521'>
    <Xcircle Size='17.07156' /></Xput>
  <Xput Xpos='85.35782' Ypos='56.90521'>
    <Xcircle Size='17.07156' /></Xput>
  <Xput Xpos='42.67891' Ypos='56.90521'>
    <Xcircle Full='true' Size='5.69052' /></Xput>
  <Xput Xpos='85.35782' Ypos='56.90521'>
    <Xcircle Full='true' Size='5.69052' /></Xput>
  <Xput Xpos='28.4526' Ypos='68.28625'>
    <Xbox Width='71.13152' Height='22.76208' Framed='true'>car</Xbox></Xput>
  <Xput Xpos='28.4526' Ypos='91.04834'>
    <Xvector XDir='-2' YDir='1' Width='28.4526' /></Xput>
  <Multiput Xpos='2.84526' Ypos='0' Repeat='49' Dx='2.84526' Dy='0'>
    <Xline XDir='0' YDir='1' Width='7.11314' /></Multiput>
  <Multiput Xpos='14.2263' Ypos='0' Repeat='5' Dx='28.4526' Dy='0'>
    <Xline XDir='0' YDir='1' Width='9.9584' /></Multiput>
  <Thicklines/>
  <Xput Xpos='0' Ypos='0'>
    <Xline XDir='1' YDir='0' Width='142.26303' /></Xput>
  <Multiput Xpos='0' Ypos='0' Repeat='6' Dx='28.4526' Dy='0'>

```

```

      <Xline XDir='0' YDir='1' Width='14.2263' /></Multitup>
    </Xpicture>

```

## 6.11 Trace of titlepage

This is a part of the transcript file for the titlepage command.

```

1 Defining \makeRR as \TitlePageCmd 0
2   main <RRstart -- type = 'RR' />
3 Defining \makeRT as \TitlePageCmd 1
4   main <RRstart -- type = 'RT' />
5 Defining \UR as \TitlePageCmd 2
6   ur_list <UR />
7 Defining \URSophia as \TitlePageCmd 3
8   ur <URSophia />
9 Defining \URRocquencourt as \TitlePageCmd 4
10  ur <URRocquencourt />
11 Defining \URRocq as alias to \URRocquencourt
12 Defining \Paris as \TitlePageCmd 6
13   ur <Rocquencourt />
14 Defining \URRhôneAlpes as \TitlePageCmd 7
15   ur <URRhôneAlpes />
16 Defining \URRennes as \TitlePageCmd 8
17   ur <URRennes />
18 Defining \URLorraine as \TitlePageCmd 9
19   ur <URLorraine />
20 Defining \URFuturs as \TitlePageCmd 10
21   ur <URFuturs d='true' />
22 Defining \RRtitle as \TitlePageCmd 11
23   usual <title /> (flags -par)
24 Defining \RRetitle as \TitlePageCmd 12
25   usual <etitle /> (flags -par)
26 Defining \RRprojet as \TitlePageCmd 13
27   usual <projet />
28 Defining \motcle as \TitlePageCmd 14
29   usual <motcle />
30 Defining \keyword as \TitlePageCmd 15
31   usual <keyword />
32 Defining \RRresume as \TitlePageCmd 16
33   usual <resume /> (flags +par)
34 Defining \RRabstract as \TitlePageCmd 17
35   usual <abstract /> (flags +par)
36 Defining \RRauthor as \TitlePageCmd 18
37   list <author /> and <auth />
38 Defining \RRdate as \TitlePageCmd 19
39   usual <date />
40 Defining \RRno as \TitlePageCmd 20
41   usual <RRnumber />
42 Defining \RRtheme as \TitlePageCmd 21
43   usual </> (flags +list)
44 Defining \Theme as \TitlePageCmd 22
45   ur_list <Theme />
46 Defining \THNum as \TitlePageCmd 23
47   ur <THNum />
48 Defining \THCog as \TitlePageCmd 24
49   ur <THCog />

```

RT n° 309

```

50 Defining \THCom as \TitlePageCmd 25
51   ur <THCom/>
52 Defining \THBio as \TitlePageCmd 26
53   ur <THBio/>
54 Defining \THSym as \TitlePageCmd 27
55   ur <THSym/>
56 Defining \myself as \TitlePageCmd 28
57   list? <grimm/> and <auth/>
58 Defining \cmdp as \TitlePageCmd 29
59   usual <cmdp/> (flags +list)
60 Defining \cmda as \TitlePageCmd 30
61   usual <cmdA/> (flags +A)
62 Defining \cmb as \TitlePageCmd 31
63   usual <cmdB/> (flags +B)
64 [1] \cmb{\cmdBval}
65 ++ End of virtual file.
66 Defining \cmdc as \TitlePageCmd 32
67   usual <cmdC/> (flags +C)
68 [1] \documentclass[a4paper]{report}
69 ...
70 [1] \cmda{\cmdAval}
71 {(Unknown)}
72 {\titlepage 30}
73 {\titlepage 30=\cmda}
74 {Push cmdA 1}
75 Error signaled at line 1 of file tptest.tex:
76 Undefined command \cmdAval.
77 ...
78 [8] \RRetitle{Tralics, a \LaTeX\ to XML translator\Part I}
79 {(Unknown)}
80 {\titlepage 12}
81 {\titlepage 12=\RRetitle}
82 {Push etitle 1}
83 ...
84 [12] \RRtheme{\THNum}
85 {(Unknown)}
86 {\titlepage 21}
87 {\titlepage 21=\RRtheme}
88 {(Unknown)}
89 {\titlepage 23}
90 {\titlepage 23=\THNum}
91 {\par}
92 [13]
93 ...
94 [24] \begin{document}
95 {\begin}
96 {\begin document}
97 +stack: level + 2 for environment
98 {\document}
99 +stack: ending environment document; resuming document.
100 +stack: level - 2 for environment
101 +stack: level set to 1
102 [1] \let\do\noexpand\ignorespaces
103 ++ End of virtual file.
104 atbegindocumenthook= \cmb {\cmdBval } \let \AtBeginDocument \@notprerr \let \do
105 \noexpand \ignorespaces

```

```

106 {(Unknown)}
107 {\titlepage 31}
108 {\titlepage 31=\cmdb}
109 {Push cmdB 1}
110 Error signaled at line 24 of file tptest.tex:
111 Undefined command \cmdBval.
112 {Pop 1: document_v cmdB_t}
113 {\let}
114 {\let \AtBeginDocument \notprerr}
115 {\let}
116 {\let \do \noexpand}
117 {\ignorespaces}
118 [25] \makeRR
119 {(Unknown)}
120 {\titlepage 0}
121 {\titlepage 0=\makeRR}
122 Error signaled at line 25 of file tptest.tex:
123 No value given for command \cmdp.
124 [1] \cmdc{\cmdCval}
125 ++ End of virtual file.
126 {(Unknown)}
127 {\titlepage 32}
128 {\titlepage 32=\cmdc}
129 {Push cmdC 1}
130 Error signaled at line 25 of file tptest.tex:
131 Undefined command \cmdCval.
132 {Pop 1: document_v cmdC_t}
133 {Push p 1}
134 [26] text
135 ...
136 Output written on tptest.xml (1059 bytes).
137 There were 4 errors.
138 (For more information, see transcript file tptest.log)

```

## 6.12 Extensions

We describe here the extensions defined by  $\epsilon$ -T<sub>E</sub>X, and how they are implemented in Tralics.

**Tracing and Diagnostics.** When `\tracingcommands` has a value of 3 or more, the commands following a prefix (like `\global`) are shown by  $\epsilon$ -T<sub>E</sub>X; this is the standard behaviour of Tralics. When `\tracinglostchars` has a value of two or more, missing characters are displayed on the terminal; no character is lost by Tralics. When `\tracingassigns` has a value of 1 and more, all assignments subject to T<sub>E</sub>X's grouping mechanism are traced. This is set to one by `\tracingall`. When `\tracingifs` has a value of one or more, all conditionals are traced, together with the starting line and nesting level; not implemented in Tralics, but it is easy to find the `\if` associated to a `\fi` because each of them has a serial number. When `\tracinggroups` has a value of 1 or more, the start and end of each save group is traced, together with the starting line and grouping level. Not implemented in Tralics, but since version 2.9, you will see line numbers when a group is started (for instance *+stack: level + 2 for brace entered on line 9*) or terminated (as in *+stack: level - 2 for brace from line 9*). When `\tracingnesting` has a value of 1 or more, unclosed conditionals are printed in the transcript file; not implemented in Tralics. When `\tracingscantokens` has a value of one or more, the opening and closing of pseudo files is recorded as for any another file.

Example. Given the following input

```

1 \global\count66=12
2 {\count66=12 \count66=13}
3 \catcode200=12
4 \parindent=3pt\parindent=1\parindent\global\parindent=2\parindent
5 \parskip=\parindent plus 2pt\parskip=\parskip
6 \setbox0\null
7 \setbox0\box{foo}{bar}
8 \everyhbox{abc}\everyhbox{abc}\everyhbox{c}
9 \let\foo\relax
10 \newcommand*\foo{\relax}\renewcommand\foo{\relax}
11 \let\bar\foo

```

We get the following lines in the transcript file.

```

1 [8] \global\count66=12
2 {\global}
3 {\global\count}
4 +scanint for \count->66
5 +scanint for \count->12
6 {globally changing \count66=0 into \count66=12}
7 [9] {\count66=12 \count66=13}
8 {begin-group character}
9 +stack: level + 2 for brace entered on line 9
10 {\count}
11 +scanint for \count->66
12 +scanint for \count->12
13 {reassigning \count66=12}
14 {\count}
15 +scanint for \count->66
16 +scanint for \count->13
17 {changing \count66=12 into \count66=13}
18 {end-group character}
19 +stack: restoring integer value 12 for \count66
20 +stack: level - 2 for brace from line 9
21 [10] \catcode200=12
22 {\catcode}
23 +scanint for \catcode->200
24 +scanint for \catcode->12
25 {reassigning \catcode200=12}
26 [11] \parindent=3pt\parindent=1\parindent\global\parindent=2\parindent
27 {\parindent}
28 +scanint for \parindent->3
29 +scandimen for \parindent->3.0pt
30 {changing \parindent=0.0pt into \parindent=3.0pt}
31 {\parindent}
32 +scanint for \parindent->1
33 +scandimen for \parindent->3.0pt
34 {reassigning \parindent=3.0pt}
35 {\global}
36 {\global\parindent}
37 +scanint for \parindent->2
38 +scandimen for \parindent->6.0pt
39 {globally changing \parindent=3.0pt into \parindent=6.0pt}
40 [12] \parskip=\parindent plus 2pt\parskip=\parskip
41 {\parskip}
42 +scanint for \parskip->2
43 +scandimen for \parskip->2.0pt
44 {scanglue 6.0pt plus 2.0pt}

```



```

45 {changing \parskip=0.0pt into \parskip=6.0pt plus 2.0pt}
46 {\parskip}
47 {reassigning \parskip=6.0pt plus 2.0pt}

```

Transcript for boxes and token lists:

```

1 [13] \setbox0\null
2 {\setbox}
3 \null ->\hbox {}
4 +scanint for \setbox->0
5 {Constructing an anonymous box}
6 +stack: level + 2 for brace entered on line 13
7 {Push hbox 1}
8 {end-group character}
9 +stack: finish a box of type 0
10 {Pop 1: document_v hbox_v}
11 +stack: level - 2 for brace from line 13
12 {changing \box0=</> into \box0=}
13 [14] \setbox0\xbox{foo}{bar}
14 {\setbox}
15 +scanint for \setbox->0
16 {Push argument 1}
17 Character sequence: foo.
18 {Text:foo}
19 {Pop 1: document_v argument_v}
20 {Constructing a box named foo}
21 +stack: level + 2 for brace entered on line 14
22 {Push hbox 1}
23 Character sequence: bar.
24 {end-group character}
25 +stack: finish a box of type 0
26 {Text:bar}
27 {Pop 1: document_v hbox_v}
28 +stack: level - 2 for brace from line 14
29 {changing \box0= into \box0=<foo>bar</foo>}
30 [15] \everyhbox{abc}\everyhbox{abc}\everyhbox{c}
31 {\everyhbox}
32 {changing \everyhbox= into \everyhbox=abc}
33 {\everyhbox}
34 {reassigning \everyhbox=abc}
35 {\everyhbox}
36 {changing \everyhbox=abc into \everyhbox=c}

```

This is the transcript file for the case of `\def` and friends. Here, two lines are printed by `\tracingassigns`.

```

1 [16] \let\foo\relax
2 {\let}
3 {\let \foo \relax}
4 {changing \foo=undefined}
5 {into \foo = \relax}
6 [17] \newcommand*\foo{\relax}\renewcommand\foo{\relax}
7 {\newcommand}
8 {\newcommand* \foo}
9 {changing \foo=\relax}
10 {into \foo= macro:->\relax }
11 {\renewcommand}
12 {\newcommand \foo}
13 {changing \foo=macro:->\relax }

```

```

14 {into \foo=\long macro:->\relax }
15 [18] \let\bar\foo
16 {\let}
17 {\let \bar \foo}
18 {changing \bar=macro:->\mathaccent "7016\relax }
19 {into \bar = \long macro:->\relax }

```

In order to debug conditionals, the variables `\currentiflevel`, `\currentifbranch` and also `\currentiftyp` can be consulted. The level is the number of currently active conditionals, the branch is 1 if the ‘then branch’ is taken, -1 if the ‘else branch’ is taken, 0 otherwise (condition not yet evaluated, or out of condition). The type is given in the following table (if the case of `\unless`, the opposite of this number is returned).

1	<code>\if</code>	8	<code>\ifmmode</code>	15	<code>\iftrue</code>
2	<code>\ifcat</code>	9	<code>\ifinner</code>	16	<code>\iffalse</code>
3	<code>\ifnum</code>	10	<code>\ifvoid</code>	17	<code>\ifcase</code>
4	<code>\ifdim</code>	11	<code>\ifhbox</code>	18	<code>\ifdefined</code>
5	<code>\ifodd</code>	12	<code>\ifvbox</code>	19	<code>\ifcsname</code>
6	<code>\ifvmode</code>	13	<code>\ifx</code>	20	<code>\iffontchar</code>
7	<code>\ifhmode</code>	14	<code>\ifeof</code>		

The `\unless` command is an extension of  $\epsilon$ -TeX; the behavior of `\unless\iftrue` is the same as `\iffalse`. This means the following: This command is expandable; it reads a token; this token must be a conditional, but not `\ifcase`. The conditional computes a truth value, which is then negated. Expansion of the command is the same as that of the conditional, said otherwise, the next token, if the test is true, otherwise what follows the `\else` or `\fi`.

Example.

```

\def\showif{%
\typeout{type \the\currentiftyp,
level \the\currentiflevel,
branch \the\currentifbranch.}}
\count3=0

\unless\iffalse
\showif
\iffalse\showif\else\showif
\ifnum\count3=\currentifbranch
\showif \fi\fi\fi

```

The following is printed on the terminal.

```

type -16, level 1, branch 1.
type 16, level 2, branch -1.
type 3, level 3, branch 1.

```

The command `\ifdefined` is a conditional; it reads a token and its truth value is true if this token is defined. The command `\ifcsname` reads and expands all tokens as `\csname`, until finding `\endcsname`. The condition is true if the token exists and is defined. If the token does not exist, it will not be created. In L<sup>A</sup>T<sub>E</sub>X, `\@ifundefined` call `\csname`, but has as side effect that the resulting token is never undefined.

The command `\iffontchar` is a conditional; it reads a font identifier, and a character position, and evaluates to true in the case where this character is defined in the font. Since Tralics does not read font metric files, nothing special happens, we pretend that the character exists, unless the font is the null font. Thus

```

\ifdefined \undefined \bad\fi
\ifdefined \par\else\bad \fi

```

```

\ifcsname foo bar\endcsname\bad\fi
\ifcsname bar\endcsname\else\bad\fi
\csname foo bar\endcsname\ifcsname foo bar\endcsname\else\bad\fi
\unless\iffontchar\font 32 \bad\fi
\iffontchar\nullfont 32 \bad\fi

```

The command `\protected` is a prefix, like `\long`, that applies to a macro definition. A protected macro is not expanded when building an expanded token list (for instance in `\edef`).

The command `\scantokens` absorbs a list of unexpanded tokens, converts it into a character string, that is treated as if it were an external file and starts to read from this pseudo file. Every newline character<sup>10</sup> is interpreted as the start of a new line.

The command `\showgroups` shows the current grouping structure. The read-only integer `\currentgrouplevel` returns the current save group level, and `\currentgroupstype` returns a number representing the type of the current group. This gives a number between 0 and 16, see the  $\epsilon$ -TeX documentation. The values used by Tralics are: 0 is bottom level (no group), 1 is simple group, 9 is math group, 14 is semi simple group, 18 is environment, 19 is a cell in a table, 20 is a local group (corresponds to 4 and 5 in  $\epsilon$ -TeX), 21 is a title-page group, 17 is impossible.

Example. If we have a file with

```

\begin{foo}
\begingroup
{
\begin{tabular}{c}
\showgroups!
\end{tabular}
\the\currentgrouplevel
\the\currentgroupstype
}
\endgroup
\end{foo}

```

The following is printed by  $\epsilon$ -TeX

```

### simple group (level 10) entered at line 13 ({)
### align group (level 9) entered at line 12 (align entry)
### align group (level 8) entered at line 12 (\halign{)
### vcenter group (level 7) entered at line 12 (\vcenter{)
### math shift group (level 6) entered at line 12 ($)
### hbox group (level 5) entered at line 12 (\hbox{)
### semi simple group (level 4) entered at line 12 (\begingroup)
### simple group (level 3) entered at line 11 ({)
### semi simple group (level 2) entered at line 10 (\begingroup)
### semi simple group (level 1) entered at line 9 (\begingroup)
### bottom level

```

or by Tralics

```

### cell group (level 5) entered at line 13
### environment group (level 4) entered at line 12
### brace group (level 3) entered at line 11
### \begingroup group (level 2) entered at line 10
### environment group (level 1) entered at line 9
### bottom level

```

---

<sup>10</sup>The  $\epsilon$ -TeX document says: current newline character

You may wonder why  $\epsilon$ -TeX uses twice as many stack levels as Tralics. This is because tables are implemented in a different way. For instance, using math mode for tables is very strange; one might wonder what the current mode is, when `\tracinggroupd` is seen; the answer is restricted horizontal mode, but why? If we insert a `\showlist` command, we see

```
### restricted horizontal mode entered at line 13 []
spacefactor 3000
### restricted horizontal mode entered at line 13 []
spacefactor 0
### internal vertical mode entered at line 12
prevdepth ignored
### internal vertical mode entered at line 12
prevdepth ignored
### math mode entered at line 12
### restricted horizontal mode entered at line 12
spacefactor 1000
### horizontal mode entered at line 12 []
spacefactor 1000
### vertical mode entered at line 0
### current page: []
```

In the same situation, you will see the following lines in the Tralics transcript file. Here, the level is an index in the XML stack, and you can see that the current mode is ‘a’ (for array).

```
level 0 entered at line 0, type document, mode_v:
<std>
<table rend='inline'><row><cell/></row></table></std>
level 1 entered at line 12, type tabular, mode_v:
<table rend='inline'><row><cell/></row></table>
level 2 entered at line 13, type row, mode_a:
<row><cell/></row>
level 3 entered at line 13, type cell, mode_a:
<cell/>
```

If you translate the following line in verbose mode

```
{\the\currentgrouplevel}
```

you will see the following in the transcript file. The current level outside the group is one, so that you see it increase to 2; but  $\epsilon$ -TeX shows it as zero, so that the `\the` inside the group expands to one. The reason for this strange behaviour is that a quantity defined at level zero is never defined; the level is never zero, so that the `\the` never returns a negative value.

```
[9] {\the\currentgrouplevel}
{begin-group character}
+stack: level + 2 for brace entered on line 9
{\the}
{\the \currentgrouplevel}
\the->1.
{end-group character}
+stack: level - 2 for brace from line 9
```

The command `\eTeXversion` expands to a token list containing the current  $\epsilon$ -TeX revision. The counter `\eTeXversion` returns  $\epsilon$ -TeX’s major version number. Thus

```
\message{\number\eTeXversion\eTeXrevision}
```

prints something like ‘2.0’.

The commands `\gluestretchorder`, `\glueshrinkorder`, `\gluestretch`, `\glueshrink` can be used when some internal quantity is scanned, for instance after `\the`. They read some glue and return one part of the glue, it can be the stretch order or the shrink order (an integer between 0 and 3), or the stretch or shrink value (as a dimension). The commands `\gluetomu`, `\mutoglu` read and return some glue. The  $\epsilon$ -T<sub>E</sub>X manual says: glue is converted into muglue and vice versa by simply equating 1pt with 1mu. Example: we define here a command, whose value will be used later.

```
\muskip0 = 18mu plus 36mu minus 1 fill
\skip0 = 10pt plus 20pt minus 1 fil
\edef\uselater{%
\the\muskip0,%
\the\mutoglu\muskip0,%
\the\skip0,%
\the\gluetomu\skip0,%
\the\mutoglu\gluetomu\skip0,%
\the\glueshrink\skip0,%
\the\gluestretch\skip0,%
\the\glueshrinkorder\skip0,%
\the\gluestretchorder\skip0}
```

The commands `\detokenize` and `\unexpanded` read a token list. The second command returns the list unchanged, the first one detokenizes it; said otherwise the token list is converted into a character list, of category code 12 (except for space). These commands behave like `\the`, in that the resulting token list is not expanded, even in a `\edef` or `\write`. This example shows how `\unexpanded` works.

```
\def\foo{12}\def\equals{<}
\edef\A{\unexpanded{\foo\equals}\foo\relax}
\def\equals{=}
\ifnum\A\else\bad\fi
\def\foo{11}
\ifnum\A\bad\fi
\ifnum\unexpanded{\foo\equals}\foo\relax\else\bad\fi
```

The command `\uselater`, defined above, should compare equal to `\xoo` defined here<sup>11</sup>

```
{\let\GDEF\gdef\let\XDEF\xdef\def\S{ }
\catcode'm=12 \catcode'u=12 \catcode'p=12 \catcode'f=12
\catcode'i=12 \catcode'l=12 \catcode'n=12 \catcode'i=12 \catcode's=12
\catcode't=12
\GDEF\MU{mu}\GDEF\PT{pt}\GDEF\FIL{fil}\GDEF\FILL{fill}%
\GDEF\PLUS{plus}\GDEF\MINUS{minus}
\XDEF\xoo{18.0\MU\S \PLUS\S 36.0\MU\S \MINUS\S 1.0\FILL,%
18.0\PT\S \PLUS\S 36.0\PT\S \MINUS\S 1.0\FILL,%
10.0\PT\S \PLUS\S 20.0\PT\S \MINUS\S 1.0\FIL,%
10.0\MU\S \PLUS\S 20.0\MU\S \MINUS\S 1.0\FIL,%
10.0\PT\S \PLUS\S 20.0\PT\S \MINUS\S 1.0\FIL,%
1.0\PT,20.0\PT,1,0}}
```

Using `\detokenize` makes life easier. The test should be true here.

```
\edef\yoo{\detokenize{18.0mu plus 36.0mu minus 1.0fill,%
18.0pt plus 36.0pt minus 1.0fill,%
10.0pt plus 20.0pt minus 1.0fil,%
10.0mu plus 20.0mu minus 1.0fil,%
```

<sup>11</sup>Guess why the macro is not called `\foo`

```

10.0pt plus 20.0pt minus 1.0fil,%
1.0pt,20.0pt,1,0}}
\ifx\yoo\xoo\else\bad\fi

```

As in the case of  $\epsilon$ -TeX, Tralics provides the notion of expressions of type number, dimen, glue or muglue, that can be used whenever a quantity of that type is needed. Such an expression is read by the scanning mechanism; basically `scanint` and friends are used to read a quantity, and `\multiply` and friends are used to perform operations. The four commands that can be used are `\numexpr`, `\dimexpr`, `\glueexpr` and `\muexpr`. They determine a type  $t$ , the type of the result, and read an expression, that is followed by an optional `\relax` (that will be read). When scanning for an operator or the end of an expression, spaces are discarded. An expression consists of one or more terms of type  $t$ , that are added or subtracted. A term of type  $t$  consists of an initial factor of type  $t$ , multiplied or divided by a numeric (integer) factor. Finally, a factor is either a quantity of type  $t$ , or a parenthesized expression. Example.

```

\ifdim \dimexpr(2pt-5pt) *\numexpr 3-3*13/5\relax + 34pt/2=32pt
\else\bad\fi

```

Here the `\relax` terminates the `\numexpr`. This is the trace. You will see *expr so far* when a term is converted into an expression (prefix '='), or after an addition or subtraction (prefix '+' or '-'). You will see *term so far* after a multiplication or division (prefix '\*' or '/') or a scaling (prefix backslash). In the case of  $a * b/c$ , a 64bit intermediate product is computed.

```

[8] \ifdim \dimexpr(2pt-5pt) *\numexpr 3-3*13/5\relax + 34pt/2=32pt
+\ifdim1
+scanint for \dimexpr->2
+scandimen for \dimexpr->2.0pt
+expr so far for \dimexpr= 2.0pt
+scanint for \dimexpr->5
+scandimen for \dimexpr->5.0pt
+expr so far for \dimexpr- -3.0pt
+scanint for \numexpr->3
+expr so far for \numexpr= 3
+scanint for \numexpr->3
+scanint for \numexpr->13
+scanint for \numexpr->5
+term so far for \numexpr\ 8
+expr so far for \numexpr- -5
+scan for \numexpr= -5
+scanint for \dimexpr->-5
+term so far for \dimexpr* 15.0pt
+expr so far for \dimexpr= 15.0pt
+scanint for \dimexpr->34
+scandimen for \dimexpr->34.0pt
+scanint for \dimexpr->2
+term so far for \dimexpr/ 17.0pt
+expr so far for \dimexpr+ 32.0pt
+scan for \dimexpr= 32.0pt
+scandimen for \ifdim->32.0pt
+scanint for \ifdim->32
+scandimen for \ifdim->32.0pt
+tiftest1 true

```

Note that  $3*13/5$  is  $8-1/5$ , and this is rounded to 8. In the case of `\divide`, the result is truncated. All intermediate expressions are checked for overflow<sup>12</sup>, which is  $2^{31}$  for an integer, and  $2^{30}$  otherwise (in magnitude). This means that dimensions and components of glue must be less than  $2^{14}$  in units of `pt`, `mu` or `fil`.

One important point is that these operations do no side effects, hence can be used inside an `\edef`. If used out of context, you can see error messages like *You can't use '\numexpr' in horizontal mode*, (the messages depends on the current mode), in Tralics, the error is *Read only variable \numexpr*, because these operations are implemented as the value of a read only variable.

Example

```
\def\foo#1#2{\number#1
  \ifnum#1<#2, %
  \expandafter\foo
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi}

\edef\Bar{\foo{7}{13}}
\def\xBar{7, 8, 9, 10, 11, 12, 13}
\ifx\Bar\xBar\else \bad\fi
```

The integer `\lastnodetype` returns a number indicating the type of the last node, if any, on the current list. This is not implemented in Tralics, and the value is always zero.

The `\interactionmode` command allows you to get or set the current interaction mode, an integer between 0 and 3. Setting it is no-op in Tralics (no error signaled), the value is always zero (this is batchmode in  $\TeX$ , which is more or less the only mode of interaction of Tralics).

The commands `\fontcharwd`, `\fontcharht`, `\fontchardp`, `\fontcharic` can be used to get some information about characters; do not use them to set a value. The command reads a font identifier, and a character position; if the character does not exist, the value is zero, otherwise the width, height, depth or italic correction. In the following example, Tralics shows 0 for the interaction mode, and 0.0pt for the other values; in  $\epsilon\text{-}\TeX$ , only the italics correction is zero.

```
{
  \showthe\interactionmode
  \interactionmode=2
  \showthe\fontcharwd\font'q
  \showthe\fontcharht\font'q
  \showthe\fontchardp\font'q
  \showthe\fontcharic\font'q
}
\showthe\interactionmode
```

The commands `\parshapelength`, `\parshapeindent`, read an integer  $n$ ; they return the length or indentation of the line  $n$  in the current parshape; the command `\parshapedimen` reads  $2n$  or  $2n + 1$ , and returns one of these quantities, depending on the parity of the argument. Example: in the following code, the `\bad` macro is not called.

```
\parshape 3 1pt 2pt 3pt 4pt 5pt 6pt
\ifnum\parshape = 3 \else\bad\fi
\ifdim\parshapelength 1 = 2.0pt\else\bad\fi
\ifdim\parshapeindent 2 = 3.0pt\else\bad\fi
\ifdim\parshapedimen 4 = 4.0pt\else\bad\fi
```

<sup>12</sup>Since version 2.12, overflow is checked for all operations

```

\ifdim\parshapedimen 5 = 5.0pt\else\bad\fi
\ifdim\parshapedimen 6 = 6.0pt\else\bad\fi
\ifdim\parshapedimen 7 = 5.0pt\else\bad\fi
\ifdim\parshapedimen 0 = 0.0pt\else\bad\fi
\parshape 0
\ifdim\parshapedimen 1 = 0.0pt\else\bad\fi

```

The four commands `\interlinepenalties`, `\clubpenalties`, `\widowpenalties`, as well as `\displaywidowpenalties` can be used to get or set penalties. The values are read, but not used by `Tralics`. The syntax is the following. In a set context, an optional equals sign is read, followed by an integer  $n$ . If the integer is positive, then  $n$  integer values are read and stored, otherwise the table is cleared. In a get context, an integer  $n$  is read, and the result is an integer; if  $n$  is negative, this is zero, if  $n$  is zero it is the length of the table, if  $n$  is positive it is the value found in the table (or the last value if  $n$  is too big). Example: in the following code, the `\bad` macro is not called.

```

\interlinepenalties=3 1 2 3
\clubpenalties=3 11 12 13
\widowpenalties=3 101 102 103
\displaywidowpenalties=3 1001 1002 1003
\widowpenalties=-1
\edef\foo{%
\the\interlinepenalties 1
\the\clubpenalties\interlinepenalties2
\the\displaywidowpenalties -1
\the\displaywidowpenalties 0
\the\displaywidowpenalties 4
\the\widowpenalties 0}
\def\xfoo{1120310030}
\ifx\foo\xfoo\else\bad\fi

```

The command `\showtokens` reads a token list, and prints it on the terminal and transcript file. As the example below shows, the start of the token list is obtained by expanding tokens and ignoring `\relax` until a brace (implicit or explicit) is found.

```

\showtokens \expandafter{\jobname}
\showtokens \expandafter{\tralicsversion }

```

The command `\readline` has the same syntax as `\read`, it is followed by a channel number, a `to` keyword, and a definable command. It reads a line from a file, and puts it in the command. The difference is that all characters are assumed of category code 12, except space that has its standard category code; only one line is read, since the result is always properly nested.

The command `\everyeof` holds a token list, like `\everypar`, that is inserted at the end of every file, or virtual file.

The counter `\lastlinefit` contains an integer, that is used by  $\epsilon$ -`TEX` to set the glue in the last line of a paragraph. For each line, the actual space used by a glue item of the form 10pt plus 4pt minus 3pt is  $10+4f$  or  $10-3f$ , depending on whether the natural width is too big or too small. The last line of a paragraph is generally terminated by an infinite stretchable glue, so that the glue factor  $f$  for normal glue is zero. In  $\epsilon$ -`TEX`, you can use the same factor as the previous line, or an interpolation ( $l/1000$  times the factor of the previous line), where  $l$  is the value of `\lastlinefit`, or 0 if negative, or 1000 if greater than 1000. Not used in `Tralics`.

The integer quantity `\savingshyphcodes`, when positive, tells  $\epsilon$ -`TEX` to store the current `\lccode` table together with the hyphenation table of the current language. See  $\epsilon$ -`TEX` documentation for why it is useful to store such a table; not used in `Tralics`.

When `TEX`'s page builder transfers material from the 'recent contribution' to the 'page so far', it discards discardable items preceding the first box or rule on the page. When  $\epsilon$ -`TEX`'s



parameter `\savingdiscards` is positive, these discarded items are stored in a special list; the command `\pagediscards` reinserts these items (and clears the list). The same holds for `\vsplit`, the command is then `\splitdiscards`.

The `\middle` command is not implemented in Tralics. This is a math-only command that reads a delimiter, and should be placed between `\left` and `\right`, more than one such command can be used. The height of the delimiter of `\left`, `\right` and `\middle` is the height of the formula, from `\left` to `\right`.

The six commands `\marks`, `\firstmarks`, `\topmarks`, `\botmarks`, `\splitfirstmarks`, and `\splitbotmarks` generalise commands like `\mark`, they read an integer  $N$ , and set a mark at position  $N$ , or get the mark at position  $N$ . No error is signalled if  $N$  is out of range.

There is a possibility to type text from, left to right or right to left in  $\epsilon$ -T<sub>E</sub>X. Not implemented in Tralics. The use of these features is controlled by the integer `\TeXXeTstate`.

The integer `\predisplaydirection` contains the text direction preceding a display. The commands `\beginL`, `\beginR`, `\endL`, `\endR` mark the start and end of a left-to-right or right-to-left region.

## 6.13 Bootstrap code

The transcript file of Tralics contains a line for each use of `\dimendef` or friends. Here is the list of all standard definitions (for Tralics version 2.12). Note that `\count@` is counter 255, while `\dimen@` is dimension 0. The names (and values) of the commands `\z@`, `\@ne`, `\tw@`, `\thr@@`, `\sixt@@n`, `\@cclv`, `\@cclvi`, `\@m`, `\@M`, `\@Mi`, `\@Mii`, `\@Miii`, `\@Miv`, `\@MM`, were chosen by Knuth (maybe Lamport). The values are 0, 1, 2, 3, 16, 255, 256, -1, 1000, 10000, 10001, 10002, 10003, 10004, and 20000. These numbers were typeset via `\number\@MM`.

Remember that the internal name of the L<sup>A</sup>T<sub>E</sub>X counter ‘enumi’ is `\c@enumi`.

```
\trivialmath=1
{\countdef \count@=\count255}
{\countdef \c@page=\count0}
{\dimendef \dimen@=\dimen0}
{\dimendef \dimen@i=\dimen1}
{\dimendef \dimen@ii=\dimen2}
{\dimendef \epsfxsize=\dimen11}
{\dimendef \epsfysize=\dimen12}
{\chardef \@ne=\char1}
{\chardef \tw@=\char2}
{\chardef \thr@@=\char3}
{\chardef \sixt@@n=\char16}
{\chardef \@cclv=\char255}
{\mathchardef \@cclvi=\mathchar256}
{\mathchardef \@m=\mathchar1000}
{\mathchardef \@M=\mathchar10000}
{\mathchardef \@Mi=\mathchar10001}
{\mathchardef \@Mii=\mathchar10002}
{\mathchardef \@Miii=\mathchar10003}
{\mathchardef \@Miv=\mathchar10004}
{\mathchardef \@MM=\mathchar20000}
{\chardef \active=\char13}
{\tokesdef \toks@=\toks0}
{\skipdef \skip@=\skip0}
{\dimendef \z@=\dimen13}
{\dimendef \p@=\dimen14}
```

```

{\dimendef \oddsidemargin=\dimen15}
{\dimendef \evensidemargin=\dimen16}
{\dimendef \leftmargin=\dimen17}
{\dimendef \rightmargin=\dimen18}
{\dimendef \leftmargini=\dimen19}
{\dimendef \leftmarginii=\dimen20}
{\dimendef \leftmarginiii=\dimen21}
{\dimendef \leftmarginiv=\dimen22}
{\dimendef \leftmarginv=\dimen23}
{\dimendef \leftmarginvi=\dimen24}
{\dimendef \itemindent=\dimen25}
{\dimendef \labelwidth=\dimen26}
{\dimendef \fboxsep=\dimen27}
{\dimendef \fboxrule=\dimen28}
{\dimendef \arraycolsep=\dimen29}
{\dimendef \tabcolsep=\dimen30}
{\dimendef \arrayrulewidth=\dimen31}
{\dimendef \doublerulesep=\dimen32}
{\dimendef \@tempdima=\dimen33}
{\dimendef \@tempdimb=\dimen34}
{\dimendef \@tempdimc=\dimen35}
{\dimendef \footnotesep=\dimen36}
{\dimendef \topmargin=\dimen37}
{\dimendef \headheight=\dimen38}
{\dimendef \headsep=\dimen39}
{\dimendef \footskip=\dimen40}
{\dimendef \columnsep=\dimen41}
{\dimendef \columnseprule=\dimen42}
{\dimendef \marginparwidth=\dimen43}
{\dimendef \marginparsep=\dimen44}
{\dimendef \marginparpush=\dimen45}
{\dimendef \maxdimen=\dimen46}
{\dimendef \normallineskiplimit=\dimen47}
{\dimendef \jot=\dimen48}
{\dimendef \paperheight=\dimen49}
{\dimendef \paperwidth=\dimen50}
{\skipdef \topsep=\skip11}
{\skipdef \partopsep=\skip12}
{\skipdef \itemsep=\skip13}
{\skipdef \labelsep=\skip14}
{\skipdef \parsep=\skip15}
{\skipdef \fill=\skip16}
{\skipdef \@tempskipa=\skip17}
{\skipdef \@tempskipb=\skip18}
{\skipdef \@flushglue=\skip19}
{\skipdef \listparindent=\skip20}
{\skipdef \hideskip=\skip21}
{\skipdef \z@skip=\skip22}
{\skipdef \normalbaselineskip=\skip23}
{\skipdef \normallineskip=\skip24}
{\skipdef \smallskipamount=\skip25}
{\skipdef \medskipamount=\skip26}
{\skipdef \bigskipamount=\skip27}
{\skipdef \floatsep=\skip28}
{\skipdef \textfloatsep=\skip29}
{\skipdef \intextsep=\skip30}

```

```

{\skipdef \dblfloatsep=\skip31}
{\skipdef \dbltextfloatsep=\skip32}
{\countdef \m@ne=\count20}
{\countdef \c@FancyVerbLine=\count21}
{\countdef \c@enumi=\count22}
{\countdef \c@enumii=\count23}
{\countdef \c@enumiii=\count24}
{\countdef \c@enumiv=\count25}
{\countdef \c@footnote=\count26}
{\countdef \c@part=\count27}
{\countdef \c@chapter=\count28}
{\countdef \c@section=\count29}
{\countdef \c@subsection=\count30}
{\countdef \c@subsubsection=\count31}
{\countdef \c@paragraph=\count32}
{\countdef \c@subparagraph=\count33}
{\countdef \c@mpfootnote=\count34}
{\countdef \c@bottomnumber=\count35}
{\countdef \c@topnumber=\count36}
{\countdef \@tempcnta=\count37}
{\countdef \@tempcntb=\count38}
{\countdef \c@totalnumber=\count39}
{\countdef \c@dbltopnumber=\count40}
{\countdef \interfootnotelinepenalty=\count41}
{\countdef \interdisplaylinepenalty=\count42}
{\toksdef \@temptokena=\toks11}
{\chardef \@tempboxa=\char11}
{\chardef \voidb@x=\char12}

```

At the start of the run, some commands are created; we start with commands that take no argument, whose expansion is formed of characters only.

- `\lq`, `\rq` (left and right quotes). Expansion is ‘‘ and ’’.
- `\lbrack`, `\rbrack` (left and right brackets) expansion is ‘[’ and ‘]’.
- `\xscale`, `\yscale`, `\xscaley`, `\yscalex`. Expansion is one, one, zero, zero (in fact ‘1.0’ and ‘0.0’).
- `\textfraction`, `\floatpagefraction`, `\dblfloatpagefraction`, `\bottomfraction`, `\dbltopfraction`, `\topfraction`. Parameters for float placement. Expansion is 20%, 50%, 50%, 30%, 70%, and 70%, in the form of ‘.7’ for instance.
- `\@vpt`, `\@vipt`, `\@viipt`, `\@viiipt`, `\@ixpt`, `\@xpt`, `\@xipt`, `\@xiipt`, `\@xivpt`, `\@xvipt`, `\@xxpt`, `\@xxvpt`. Font sizes. Expansion is 5, 6, 7, 8, 9, 10, 10.95, 12, 14.4, 17.28, 20.74 and 24.88.
- `\@height`, `\@depth`, `\@width`, `\@plus`, `\@minus`. Expansion is the name of the command without the at-sign. In the L<sup>A</sup>T<sub>E</sub>X kernel, you can see ‘`\vskip \z@ \@plus .2\p@`’ instead of the more obvious ‘`\vskip 0pt plus 0.2pt`’.
- `\encodingdefault`, `\familydefault`, `\seriesdefault`, `\shapedefault`: This default the current font as T1+cmr/m/n.
- `\I`, `\J`. Uppercase version of `\i` and `\j`. Expansion is I or J.
- `\baselinestretch`, `\arraystretch`: expansion is 1.

- `\space`, `\thinspace`, `\@headercr\headercr"@headercr`: Expansion is a single space character.
- `\refname` expands to ‘Bibliography’.
- `\footcitesep`, `\citepunct`. Expansion is comma space for both commands.
- `\fmtname` expands to ‘Tralics’.

We continue with more complicated commands:

- `\@spaces`. Expansion is four `\space`.
- Space, sharp, underscore, newline, as active characters, are defined to be equivalent, respectively, to `\space`, `\#`, `\_` and `\par`.
- An active `&` character produces `&`.
- An active form-feed character is an outer `\par`.
- The tilde and no-breakspace (character 160) expand to `\nobreakspace`.
- `\obeyspaces` makes the space character active.
- `\@vobeyspaces` makes the space character active, and defines it to be `\nobreakspace`.
- `\obeylines` makes newline character active (and defines it to be `\par`).
- The `sloppypar` environment expands to `\par` in the begin and end part.
- The command `\hb@xt@` is a short hand for `\hbox to`.
- The `\null` command produces an empty box. This is invisible, but some rules will be applied differently (for instance it can inhibit ligatures).
- `\labelitemi`, `\labelitemii`, `\labelitemiii`, and `\labelitemiv` produce a bullet, endash, asterisk centered, and period centered, via `\textbullet`, etc.
- The `guillemets` environment produces `\guillemotleft` and `\guillemotright`
- `\@thirdofthree` reads three arguments, and expands to the last.
- `\@carcube` reads three arguments and returns them. Moreover, it reads and discards all tokens up to `\@nil`.
- The `\date` command takes an argument that L<sup>A</sup>T<sub>E</sub>X puts in `\@date` which is filled with `\today`. The `\today` command is defined in some classes using English month names, and some packages (using local names). In Tralics, the default is a numeric list like ‘2006/10/12 19:14:59’ (this is the same string that appears in the transcript file). The `\date` command produces a `<date>` element with the arguments.
- The commands `\markright` and `\markboth` take one or two arguments and call `\mark`; since `\mark` does nothing in Tralics, the implementation is oversimplified; it modifies the command `\@themark`.
- `\mod`, `\bmod`, `\pmod`, `\pod` are commands that take one argument, they are different implementation of the modulo operator.

- The six following commands `\chaptermark`, `\sectionmark`, `\subsectionmark`, `\subsubsectionmark`, `\paragraphmark`, and `\subparagraphmark` are defined by compatibility with L<sup>A</sup>T<sub>E</sub>X to take an argument and ignore it. These commands are redefined by standard classes<sup>13</sup> and typically call `\markboth` or `\markright`.
- The command `\@addnl` inserts a newline in the XML tree.
- `\addto@hook` takes two arguments A and B. First argument is assumed to be a reference to a token list (for instance `\everypar`). The tokens in the the list B are put at the end of A.
- `\g@addto@macro` takes two arguments A and B. First argument is assumed to be a macro that takes no argument. The tokens in the the list B are put at the end of A. Assignment is global.
- The command `\@setmode` scans an integer. Values outside the range 0-6 are replaced by 0. Other values correspond to argument mode (0), or horizontal mode (1), or vertical mode(2), no mode (3), bibliographic mode (4), array mode (5), mathmode (6). Do not use this command if you do not understand the consequences. In a case like `1\@setmode 2 \space3 4 5\par`, there is no space between 1 and 3, because we are vertical mode, horizontal mode is entered when the the digit 3 is seen. In a case like `1\@setmode 4 \space3 4 5\par` there is no space at all since they are ignored in bib mode. The `\par` command is ignored as well, so you had better to add `\@setmode1` or enclose everything in a group.
- `\x@tag` and `\y@tag` are internal macros for equation tags.

We show here the meaning of `\@sanitize`, `\@nnil` and `\do`.

```
\@sanitize=macro: ->\@makeoother \ \@makeoother \\ \@makeoother \$ \@makeoother
  & \@makeoother # \@makeoother ^ \@makeoother _ \@makeoother \% \@makeoother ~.
\dospecials=macro: ->\do \ \do \\ \do \$ \do & \do # \do ^ \do _ \do \% \do
  ~ \do { \do \}.
\@nnil=macro: ->\@nil .
```

The page counter is `\count0`. We first define `\c@page`, via `\countdef`, then `\cl@page` to be empty, `\c@page` to be one, and `\thepage` to use arabic numbers; the `\pagenumbering` command redefines the `\thepage`. The command `\p@page` is not defined because `\p@foo` is used only for printing the label associated to the counter (if you want the page number of reference ‘foo’, use `\pageref`). Note that in Tralics, the page counter is never modified, `\pageref` is the same as `\ref`. Consider a reference to the page containing the start of the verbatim environment we are commenting: 283. In L<sup>A</sup>T<sub>E</sub>X, the value of `\@currentlabel` is considered, in Tralics, the current anchor is used instead; this works well for `\ref`. The current label is defined here:6.13, at the start of the section. For the HTML version of the document, we have cheated a bit. We have added an `\index` command, and this inserts an anchor after the word ‘bootstrap’; the label follows the `\index`. The `\pageref` command uses this label. Note that the `\anchor` command adds an anchor, but this is not defined by L<sup>A</sup>T<sub>E</sub>X. The commands defined here produce a `<pagestyle>` element.

```
\def\pagenumbering#1{\xmlelt{pagestyle}{\XMLaddatt{numbering}{#1}}}
\def\pagestyle#1{\xmlelt{pagestyle}{\XMLaddatt{style}{#1}}}
\def\thispagestyle#1{\xmlelt{pagestyle}{\XMLaddatt{this-style}{#1}}}
```

We show now the remaining of the bootstrap code. We start with filling some registers.

```
[1] %% Begin bootstrap commands for latex
[2] \@flushglue = 0pt plus 1fil
[3] \hideskip =-1000pt plus 1fill
[4] \smallskipamount=3pt plus 1pt minus 1pt
```

<sup>13</sup>maybe also indirectly via `\pagestyle`

```
[5] \medskipamount=6pt plus 2pt minus 2pt
[6] \bigskipamount=12pt plus 4pt minus 4pt
[7] \z@=0pt\p@=1pt\m@ne=-1 \fboxsep = 3pt %
[8] \c@page=1 \fill = 0pt plus 1fill
[9] \paperheight=297mm\paperwidth=210mm
[10] \jot=3pt\maxdimen=16383.99999pt
```

The two commands defined here take as argument either a character or a one character command.

```
[33] \def\@makeoother#1{\catcode'#1=12\relax}
[34] \def\@makeactive#1{\catcode'#1=13\relax}
```

Other commands

```
[11] \def\newfont#1#2{\font#1=#2\relax}
[12] \def\symbol#1{\char #1\relax}
[16] \newenvironment{cases}{\left\{\begin{array}{l}}{\end{array}\right.}%
[19] \def\stretch#1{\z@ \@plus #1fill\relax}
[20] \theoremstyle{plain}\theoremheaderfont{\bfseries}
[21] \def\@namedef#1{\expandafter\def\csname #1\endcsname}
[22] \def\@nameuse#1{\csname #1\endcsname}
[23] \def\@arabic#1{\number #1}
[24] \def\@roman#1{\romannumeral#1}
[25] \def\@Roman#1{\Romannumeral#1}
[30] \def\LaTeXe{\LaTeX2$\epsilon}
[32] \def\enspace{\kern.5em}
[35] \def\root#1\of{\@root{#1}}
[40] \def\eqref#1{(\ref{#1})}
[43] \def\on@line{ on input line \the\inputlineno}
[47] %% End bootstrap commands for latex
```

## 6.14 Standard packages

Version 1 of this document described the status of standard packages for Tralics2.9. It has been withdrawn. The list of all packages, with documentation, is be found on the web.

## 6.15 Images

We give here some examples of the `\includegraphics` command. We consider a file with the following content. Notice that the clip attribute is set to true if 'clip' appears in the list, whether or not a value has been given. A colon and an underscore in a file name is never interpreted. The extension is always removed:

```
\let\IC=\includegraphics
\def\FILE{Logo-INRIA-couleur}
\IC[angle=0,width=3cm,clip]{\FILE}
\IC[angle=20,width=.5\textwidth,height=.3\textheight]{\FILE.ducon}
\IC[angle=0,width=\columnwidth,height=\textheight]{\FILE.foo_bar}
{\language=1 a:c
\IC[angle=0, =foo,,width=3cm,scale=1,scale=2,clip]{.././a_b:c}
}
\framebox{\includegraphics{x_.ps}}
```

We continue with an example of `\epsfbox`.

```
{
  \setlength\epsfxsize{50pt}
  \setlength\epsfysize{60pt}
  \epsfbox{x.ps}
  \setlength\epsfysize{70pt}
  \epsfbox{x.eps}
  \epsfbox{x.epsf}
}
```

Tralics pretends that there are 4 different images. The translation is:

```
<figure rend='inline' clip='true' width='3cm' file='Logo-INRIA-couleur'/>
<figure rend='inline' height='6.cm' width='7.5cm'
  angle='20' file='Logo-INRIA-couleur'/>
<figure rend='inline' height='20.cm' width='15.cm'
  file='Logo-INRIA-couleur'/>
a :c
<figure rend='inline' clip='true' scale='2' width='3cm' file='.././a_b:c'/>
<figure framed='true' rend='inline' file='x_'/></p>
<figure height='60.0pt' width='50.0pt' rend='inline' file='x'/>
<figure height='70.0pt' rend='inline' file='x'/>
<figure rend='inline' file='x'/>
```

The file ‘\jobname.img’ contains the following. The last number is the number of times the image was included. The second number explains in which format the file has been found. Whether or not the image file is found is irrelevant. The information given in the file is for information only.

```
# images info, 1=ps, 2=eps, 4=epsi, 8=epsf, 16=pdf, 32=png, 64=gif
see_image("Logo-INRIA-couleur",1+16,3);
see_image(".././a_b:c",0,1);
see_image("x_",0,1);
see_image("x",0,3);
```

## 6.16 The puzzle

The only requirements for the xii file is that ~ is an active character, \ has category code 0, % is a comment character, the end of line character is as usual. The file modifies the category code of 7, F, j and P, in such a way that ‘jdefjx71F71P’ is the same as ‘\def\x#1{#1}’. This is one way of making the code incomprehensible, the other is to use commands so that ‘six’, ‘geese’ and ‘laying’ are replaced by ‘/sx’, ‘Yegse’ and ‘RyalD’. The programs make the letter H active and defines it via ‘AHHFLP’. For those who want to write puzzles like this one: is it possible to avoid doubling the H? Without using all these strange commands, the file could be written as

```
\let~\catcode ~'A13 \defA#1{~'#113\def}
AZZ{}APP{\par}AXX#1{\bigskip On the #1 day of Christmas my true love gave to me}
ABB{PZAZZ{and }a partridge in a pear tree.}
ACC{Ptwo turtle doves}
ADD{Pthree french hens}
AEE{Pfour calling birds}
AFF{Pfive gold rings}
AGG{Psix geese a laying}
AHH{Pseven swans a swimming}
AII{Peight maids a milking}
AJJ{Pnine ladies dancing}
AKK{Pten lords a leaping}
```

```

ALL{Peleven pipers piping}
AMM{Ptwelve drummers drumming}
\def\F#1#2#3{}AVV#1\fi{\fi#1}
ATT#1 #2,#3:{\if.#3.\elseT#3:\fiX{#1}\U#1 #2,#3:}
\def\U#1#2#3#4 #5,#6{\F#1#2#3#5\if:#6\elseV\U#6\fi}
Ttwelfth M,eleventh L,tenth K,ninth J,eighth I,seventh H,sixth G,fifth
F,fourth E,third D,second C,first B,:\bye

```

The size of the file is 698 characters (compare to the 767 of the xii file). Note how the double loop is constructed. The xii file is made obscure by replacing B, C, D etc., by expression that have the same expansion, using instead of \F a command that uses some characters (because ‘nine’ and ‘ninth’ start with the same letters, etc.) An interesting point is that we can write a smaller file, with all loops unrolled, replacing the last 5 lines by the following (this makes a total of 642 characters):

```

X{first}BX{second}CBX{third}DCBX{fourth}EDCBX{fifth}FEDCBX{sixth}GFEDCB
X{seventh}HGFEDCBX{eighth}IHGFEDCBX{ninth}JIHGFEDCBX{tenth}KJIHGFEDCB
X{eleventh}LKJIHGFEDCBX{twelfth}MLKJIHGFEDCB\bye

```



# Index

$\!$ , 55, 90, 146  
 $\"$ , 135  
 $\'$ , 135  
 $($ , 90  
 $\langle$ , 88, 166  
 $\rangle$ , 90  
 $\rangle$ , 88, 166  
 $\,$ , 90, 146  
 $\-$ , 144  
 $\.$ , 90  
 $\.$ , 135  
 $\backslash$ , 191  
 $\:$ , 55, 90  
 $\;$ , 55, 90  
 $<$ , 90  
 $\=$ , 135  
 $>$ , 90  
 $\>$ , 55, 90  
 $\@$ , 191  
 $[$ , 90  
 $\lbrack$ , 88, 166  
 $\#$ , 21, 61  
 $\#$ , 144  
 $\$$ , 7, 87  
 $\%$ , 62, 80  
 $\%$ , 90, 146  
 $\&$ , 90, 146  
 $\^$ , 135  
 $\sim$ , 23, 130  
 $\backslash\backslash$ , 54  
 $\_$ , 90, 146  
 $\$$ , 90, 146  
 $\lvert$ , 92  
 $|$ , 90  
 $\sim$ , 135  
 $\sim$ , 55  
 $\{$ , 90, 146  
 $\}$ , 90, 146  
 $\}$ , 88  
 $]$ , 90  
 $\'$ , 135  
 $\a$ , 84, 135, 166  
 $\AA$ , 84, 144  
 $\aa$ , 144  
 $\abarnodeconnect$ , 194  
 $\above$ , 99  
 $\abovedisplaysshortskip$ , 38  
 $\abovedisplayskip$ , 38  
 $\abovewithdelims$ , 99  
 $\accent$ , 10, 135  
 $\active$ , 7, 29  
 $\acute$ , 92  
 $\addattributestodocument$ , 186  
 $\AddAttToCurrent$ , 36, 186  
 $\AddAttToLast$ , 36, 186  
 $\@addnl$ , 283  
address (bibtex field), 115  
 $\addto@hook$ , 283  
 $\addtocounter$ , 48, 84, 172  
 $\addtolength$ , 49, 84, 171  
 $\@addtoreset$ , 50  
 $\adjdemerits$ , 40  
 $\advance$ , 38, 48, 64  
 $\AE$ , 144  
 $\ae$ , 144  
affiliation\_ vals, 247  
 $\afterassignment$ , 35  
 $\@afterelsefi$ , 76, 84, 167  
 $\@afterfi$ , 76, 84, 167  
 $\aftergroup$ , 80, 82  
 $\aleph$ , 91  
align, 88  
aligned, 88  
all (tralics option), 200  
 $\allowbreak$ , 191  
 $\@Alph$ , 48, 84  
 $\Alph$ , 48, 84  
 $\@alph$ , 48, 84  
 $\alph$ , 48, 84  
 $\alpha$ , 90  
alternate\_item, 250  
 $\amalg$ , 91  
 $\anchor$ , 182, 283  
 $\AND$ , 173  
 $\and$ , 173

`\angle`, 90  
`\anodeconnect`, 194  
`\anodecurve`, 194  
`\aparaitre`, 184  
`\approx`, 91  
`apr` (bibtex field), 114  
`\@arabic`, 48, 83, 167  
`\arabic`, 48, 84  
`\arc`, 233, 263  
`\arccos`, 91  
`\arcsin`, 91  
`\arctan`, 91  
`\arg`, 91  
`array`, 89, 238  
`\arraycolsep`, 43  
`\arrayrulewidth`, 43  
`\arraystretch`, 281  
`article` (bibtex entry type), 114  
`\ast`, 91  
`\asymp`, 91  
`\AtBeginDocument`, 158  
`\AtEndDocument`, 27, 158  
`\AtEndOfClass`, 161  
`\AtEndOfPackage`, 161  
`\atop`, 99  
`\atopwithdelims`, 99  
`att_affiliation`, 261  
`att_angle`, 259  
`att_box_pos`, 259  
`att_box_scale`, 253  
`att_box_width`, 253  
`att_boxed`, 253  
`att_cell_bottomborder`, 260  
`att_cell_center`, 260  
`att_cell_left`, 260  
`att_cell_leftborder`, 260  
`att_cell_right`, 260  
`att_cell_rightborder`, 260  
`att_cell_topborder`, 260  
`att_centering`, 259  
`att_clip`, 259  
`att_cols`, 260  
`att_curve_nbpts`, 263  
`att_dx`, 263  
`att_dy`, 263  
`att_encap`, 262  
`att_fbox_rend`, 259  
`att_file`, 259  
`att_flush_left`, 259  
`att_flush_right`, 259  
`att_foot_position`, 248  
`att_framed`, 253  
`att_full`, 259  
`att_full_first`, 261  
`att_gloss_type`, 251  
`att_halign`, 260  
`att_HDR`, 261  
`att_height`, 259  
`att_inner_pos`, 259  
`att_junior`, 261  
`att_level`, 262  
`att_minipage_width`, 259  
`att_noindent`, 259  
`att_nom`, 261  
`att_nonumber`, 252  
`att_page`, 262  
`att_particule`, 261  
`att_place`, 248  
`att_pos`, 259  
`att_prenom`, 261  
`att_profession`, 261  
`att_quotation`, 259  
`att_quote`, 259  
`att_rend`, 259  
`att_repeat`, 263  
`att_rotate_angle`, 254  
`att_row_spaceafter`, 260  
`att_row_spacebefore`, 260  
`att_scale`, 259  
`att_size`, 263  
`att_space_before`, 259  
`att_table_width`, 259  
`att_topic_num`, 206, 254  
`att_unit_length`, 263  
`att_user_list`, 250  
`att_vpos`, 259  
`att_width`, 259  
`att_xdir`, 263  
`att_xpos`, 263  
`att_xscale`, 263  
`att_xscaley`, 263  
`att_ydir`, 263  
`att_ypos`, 263  
`att_yscale`, 263  
`att_yscalex`, 263  
`aug` (bibtex field), 114  
`author` (bibtex field), 115  
`\b`, 84, 135  
`babel` (package), 158  
`\backmatter`, 183  
`\backslash`, 91  
`\badness`, 41  
`\bar`, 92

---

`\barnodeconnect`, 194  
`\baselineskip`, 38  
`\baselinestretch`, 281  
`\batchmode`, 190  
`\bauthors`, 117, 123  
`\beditors`, 117, 123  
`\begin`, 25  
`\beginngroup`, 26  
`\beginL`, 279  
`\beginR`, 279  
`\@begintheorem`, 173  
`\belowdisplayshortskip`, 38  
`\belowdisplayskip`, 38  
`\beta`, 90  
`\bezier`, 232, 263  
`\bf`, 5, 54, 93  
`\bfseries`, 54, 249  
`\bgroup`, 29, 80  
`\bibitem`, 113  
`\bibliography`, 124  
`\bibliographystyle`, 124  
`bibtex_extensions`, 247  
`bibtex_fields`, 247  
`\Big`, 101  
`\big`, 15, 101  
`\bigbreak`, 58  
`\bigcap`, 91  
`\bigcirc`, 233, 263  
`\bigcup`, 91  
`\Bigg`, 101  
`\bigg`, 101  
`\Biggl`, 101  
`\biggl`, 101  
`\Biggm`, 101  
`\biggm`, 101  
`\Biggr`, 101  
`\biggr`, 101  
`\Bigl`, 101  
`\bigl`, 101  
`\Bigm`, 101  
`\bigm`, 101  
`\bigodot`, 91  
`\bigoplus`, 91  
`\bigotimes`, 91  
`\Bigr`, 101  
`\bigr`, 101  
`\bigskip`, 56  
`\bigskipamount`, 39  
`\bigsqcup`, 91  
`\bigtriangledown`, 91  
`\bigtriangleup`, 91  
`\biguplus`, 91  
`\bigvee`, 91  
`\bigwedge`, 91  
`\bindnasrepma`, 91  
`\binom`, 95, 99  
`\binoppenalty`, 39  
`\bm`, 13  
`Bmatrix`, 89  
`bmatrix`, 89  
`\bmod`, 282  
`\boldmath`, 13  
`book` (bibtex entry type), 114  
`booklet` (bibtex entry type), 114  
`booktitle` (bibtex field), 115  
`\boolean`, 172  
`bootstrap code`, 283  
`\bot`, 90  
`\botmark`, 83, 166  
`\botmarks`, 166, 279  
`\bottom`, 90  
`\bottomfraction`, 281  
`bottomnumber` (counter), 42  
`\bowtie`, 92  
`\Box`, 91  
`\box`, 44  
`\boxmaxdepth`, 42  
`bp` (unit), 45  
`\bpers`, 116, 123, 261  
`\break`, 191  
`\breve`, 92  
`\brokenpenalty`, 39  
`\bullet`, 91  
`\bye`, 204  
  
`\C`, 84, 135  
`\c`, 84, 135  
`\cal`, 93  
`calc`, 49  
`\calc`, 172  
`calc` (package), 158  
`\cap`, 91  
`\caps`, 139  
`\caption`, 181  
`\@car`, 84, 167  
`\@carcube`, 282  
`cases`, 89  
`\catcode`, 7, 44, 132  
`catperso`, 222, 223  
`catperso_vals`, 247  
`cc` (unit), 45  
`\@cclv`, 279  
`\@cclvi`, 279

`\cdot`, 91  
`\cdots`, 92  
`\@cdr`, 84, 167  
`center`, 191, 259  
`\centering`, 181, 191, 259  
`\centerline`, 191, 263  
`\cfoot`, 189  
`\cgloss@gl1`, 195  
`\cgloss@gl11`, 195  
`\chapter`, 182, 252  
`chapter` (bibtex field), 115  
`chapter` (counter), 42  
`\chaptermark`, 283  
`\char`, 10, 11, 130  
`\chardef`, 29  
`\thead`, 189  
`\check`, 92  
`check` (tralics option), 200  
`\CheckCommand`, 25  
`checkxml` (raweb action), 247  
`\chi`, 90  
`\choose`, 99  
`\circ`, 91  
`\circle`, 233, 259, 263  
`\citation`, 26, 116, 123  
`\cite`, 107, 111  
`\cite@type`, 112  
`\cite@one`, 111, 112  
`\cite@prenote`, 112  
`\cite@type`, 112  
`\citepunct`, 111, 282  
`\cititem`, 123  
`\ClassError`, 191  
`\ClassErrorNoLine`, 191  
`\ClassInfo`, 191  
`\ClassWarning`, 191  
`\ClassWarningNoLine`, 191  
`\cleaders`, 184  
`\cleardoublepage`, 191  
`\clearpage`, 191  
`\cline`, 243  
`\closecurve`, 233, 263  
`\closein`, 176  
`\closeout`, 178  
`\clubpenalty`, 39  
`\clubsuit`, 91  
`\cl@xxx`, 48  
`cm` (unit), 45  
`\columnsep`, 43  
`\columnseprule`, 43  
`\columnwidth`, 43  
`comment`, 191  
`comment` (bibtex keyword), 114  
`confdir` (tralics option), 200  
`conference` (bibtex entry type), 114  
`config` (tralics option), 200  
`configfile` (tralics option), 200  
`\cong`, 91  
`\@cons`, 34  
`\coprod`, 91  
`\copy`, 29  
`\copyright`, 146  
`\cos`, 91  
`\cosh`, 91  
`\cot`, 91  
`\coth`, 91  
`\count@`, 279  
`\count`, 17, 44  
`\count@`, 29  
`\countdef`, 29  
`coursenotes` (bibtex entry type), 114  
`\cr`, 241, 244  
`\crr`, 241, 244  
`crossref` (bibtex field), 115  
`\csc`, 91  
`\csname`, 21, 25, 31, 32, 33, 50, 59, 78, 83, 166, 272  
`\cup`, 91  
`\currentgrouplevel`, 41, 273  
`\currentgrouptype`, 41, 273  
`\currentifbranch`, 42, 272  
`\currentiflevel`, 42, 272  
`\currentifttype`, 42, 272  
`\CurrentOption`, 162  
`\curve`, 233, 263  
`\c@xxx`, 48  
`\D`, 84, 135  
`\d`, 84, 135  
`\dag`, 144  
`\dagger`, 91  
`\dashbox`, 231  
`\dashline`, 234, 263  
`\dashv`, 91  
`\date`, 282  
`\day`, 40  
`\dbinom`, 95  
`\dblfloatpagefraction`, 281  
`\dblfloatsep`, 39  
`\dbltextfloatsep`, 39  
`\dbltopfraction`, 281  
`dbltopnumber` (counter), 42  
`dd` (unit), 45  
`\ddag`, 145

---

`\ddagger`, 91  
`\ddddot`, 92  
`\ddot`, 92  
`\dots`, 92  
`\deadcycles`, 44  
`dec` (bibtex field), 114  
`\DeclareMathOperator`, 104  
`\DeclareOption`, 162  
`\DeclareRobustCommand`, 25  
`\declaretopic`, 221, 223  
`\def`, 7  
`defaultclass` (tralics option), 201  
`\default@ds`, 162  
`\defaultshyphenchar`, 40, 52  
`\defaultskewchar`, 40, 52  
`\DefineShortVerb`, 139  
`\DefineVerbatimEnvironment`, 143  
`\deg`, 91  
`\delcode`, 44, 132  
`\delimiter`, 105  
`\delimiterfactor`, 40  
`\delimitershortfall`, 42  
`\Delta`, 90  
`\delta`, 90  
`\dendcsname`, 21  
`\@depth`, 281  
`\depthof`, 172  
`description`, 187  
`\det`, 91  
`\detokenize`, 166, 275  
`\dfrac`, 92, 101  
`\DH`, 144, 145  
`\dh`, 144, 145  
`\Diamond`, 91  
`\diamond`, 91  
`\diamondsuit`, 91  
`\dim`, 91  
`\dimen@`, 279  
`\dimen`, 17, 44  
`\dimen@`, 49  
`\dimen@i`, 49  
`\dimen@ii`, 49  
`\dimendef`, 29  
`\dimexpr`, 42, 276  
`\ding`, 192  
`dir` (tralics option), 201  
`\discretionary`, 190, 197  
`\displayindent`, 43  
`\displaylimits`, 89  
`displaymath`, 88  
`\displaystyle`, 15, 95, 100  
`\displaywidowpenalty`, 39  
`\displaywidth`, 42  
`distinguish_refer_in_rabib` (raweb action), 247  
`distinguish_refer_in_rabib` (tralics option), 201  
`\div`, 91  
`\divide`, 38, 48  
`\DJ`, 144, 145  
`\dj`, 144, 145  
`\do`, 34, 61, 80, 283  
`doc-type` (tralics option), 201  
`document`, 5, 27, 110, 114, 124  
`\documentclass`, 5, 158, 161  
`doi` (bibtex field), 115  
`\dot`, 92  
`\doteq`, 91  
`\dots`, 90, 146  
`\dottedline`, 234, 263  
`\doublehyphendemerits`, 40  
`\doublerulewidth`, 43  
`\Downarrow`, 90, 92  
`\downarrow`, 90, 92  
`\dp`, 44  
`\drawline`, 234, 263  
`\dump`, 8  
`dvitops` (raweb action), 247  
`\eachwordone`, 195  
`\eachwordthree`, 195  
`\eachwordtwo`, 195  
`\edef`, 33, 34, 70  
`edition` (bibtex field), 115  
`editor` (bibtex field), 115  
`\egroup`, 29, 80  
`\eject`, 58  
`\ell`, 90  
`\else`, 64, 66, 83, 167  
`\@elt`, 50  
`\em`, 54  
`em` (unit), 45  
`\emergencystretch`, 43  
`\emph`, 6, 54  
`\empty`, 74  
`\emptyset`, 90  
`\encodingdefault`, 281  
`\@@end`, 181  
`\end`, 26, 82  
`\endcitation`, 26, 123  
`\endcsname`, 25, 31, 32, 33, 50, 78, 83  
`\endgraf`, 18, 26  
`\endgroup`, 26, 27  
`\endguillemets`, 145

`\endinput`, 26, 83, 167, 178, 179  
`\endL`, 279  
`\endline`, 26  
`\endlinechar`, 26, 40, 54, 69, 129  
`\endR`, 279  
`\endsec`, 26, 183, 252  
`\@endtheorem`, 173  
`\ensuremath`, 84, 94, 101  
`entity_names`, 247  
`entnames` (tralics option), 201  
`enumerate`, 187  
`enumi` (counter), 42, 187  
`enumii` (counter), 42, 187  
`enumiii` (counter), 42, 187  
`enumiv` (counter), 42, 187  
`\epsfbox`, 190, 284  
`\epsffile`, 190  
`\epsfig`, 190  
`\epsfxsize`, 43  
`\epsfysize`, 43  
`\epsilon`, 90  
`eqnarray`, 88  
`\eqno`, 86  
`\eqref`, 182  
`\equal`, 71, 172  
`equation`, 88  
`\equiv`, 91  
`\eres`, 54  
`\errhelp`, 39  
`\errmessage`, 181  
`\error`, 191  
`\errorcontextlines`, 41  
`\errorstopmode`, 190  
`\escapechar`, 32, 40  
`\eta`, 90  
`etex` (tralics option), 201  
`\eTeXrevision`, 167  
`\eTeXversion`, 42, 274  
`\evensidemargin`, 43  
`\everycr`, 39  
`\everydisplay`, 39, 88  
`\everyeof`, 39, 278  
`\everyhbox`, 35, 36, 39, 184  
`everyjob`, 247  
`\everyjob`, 39, 247  
`\everymath`, 39, 88  
`\everypar`, 39  
`\everyvbox`, 36, 39, 184  
`\everyxbox`, 36, 39, 184  
`\ex`, 195  
`ex` (unit), 45  
`exe`, 195  
`\ExecuteOptions`, 162  
`\exhyphenpenalty`, 39  
`\exists`, 91  
`\exp`, 91  
`\expandafter`, 25, 30, 62, 64, 65, 82, 83, 166  
`external-prog` (tralics option), 201  
`\f`, 84, 135  
`\fam`, 40, 53, 105  
`\familydefault`, 281  
`\fancyfoot`, 189  
`fancyhdr` (package), 158  
`\fancyhead`, 189  
`\fancyhf`, 189  
`\fancyinternal`, 189  
`\fancyplain`, 189  
`FancyVerbLine` (counter), 42  
`\fbox`, 231, 253  
`\fboxrule`, 43, 230  
`\fboxsep`, 43, 230  
`feb` (bibtex field), 114  
`\fg`, 145  
`\fi`, 32, 64, 66, 83, 167  
`figure`, 238  
`fil`, `fill`, `fill` (unit), 45  
`filecontents`, 175  
`\fill`, 39  
`\fillbreak`, 58  
`\finalhyphendemerits`, 40  
`findwords` (tralics option), 201  
`\firstmark`, 83, 166  
`\firstmarks`, 166, 279  
`\@firstofone`, 84, 166  
`\@firstoftwo`, 84, 166  
`\flat`, 90, 146  
`\floatingpenalty`, 40  
`\floatpagefraction`, 281  
`\floatsep`, 39  
`flushleft`, 191, 259  
`flushright`, 191, 259  
`\fmtname`, 282  
`\@fnsymbol`, 48, 84  
`\fnsymbol`, 48, 84  
`\font`, 10, 34  
`\fontchardp`, 42, 277  
`\fontcharht`, 42, 277  
`\fontcharic`, 42, 277  
`\fontcharwd`, 42, 277  
`\fontdimen`, 34, 52  
`\fontencoding`, 52  
`\fontfamily`, 52  
`\fontname`, 83, 166

---

`\fontseries`, 52  
`\fontshape`, 52  
`\fontsize`, 52  
`foot` (bibtex type), 111, 115  
`\footcite`, 111  
`\footcitepre`, 111  
`\footcitesep`, 111, 282  
`\footnote`, 182, 248  
`footnote` (counter), 42  
`\footnotesize`, 54, 248  
`\footskip`, 43  
`\forall`, 91  
`fp` (package), 158  
`\FPabs`, 150  
`\FPadd`, 150  
`\FParccos`, 154  
`\FParcsin`, 154  
`\FParcsincos`, 154  
`\FPclip`, 150  
`\FPcos`, 153  
`\FPcot`, 153, 154  
`\FPcsolve`, 154  
`\FPdiv`, 151  
`\FPe`, 152  
`\FPeval`, 156  
`\FPexp`, 152  
`\FPifeq`, 152  
`\FPifgt`, 152  
`\FPifint`, 152  
`\FPiflt`, 152  
`\FPifneg`, 152  
`\FPifpos`, 152  
`\FPifzero`, 152  
`\FPindent`, 150  
`\FPln`, 152  
`\FPlsolve`, 154  
`\FPmax`, 152  
`\FPmin`, 152  
`\FPmul`, 150  
`\FPneg`, 150  
`\FPpascal`, 152  
`\FPpow`, 153  
`\FPprint`, 149, 150  
`\FPqqsolve`, 155  
`\FPqsolve`, 154  
`\FPrand`, 152  
`\FProot`, 153  
`\FPround`, 152  
`\FPseed`, 41, 152  
`\FPset`, 149, 150  
`\FPsgn`, 150  
`\FPsin`, 153  
`\FPSincos`, 153  
`\FPsub`, 150  
`\FPtan`, 153, 154  
`\FPtancot`, 153, 154  
`\FPtruncate`, 152  
`\FPupn`, 155  
`\frac`, 92, 95, 101  
`\frame`, 233  
`\framebox`, 230, 231, 253  
`french` (package), 158  
`frenchle` (package), 158  
`\frontmatter`, 183  
`\frown`, 92  
`\fsc`, 190  
`\futurelet`, 30, 73  
`\fvset`, 143  
`\g@addto@macro`, 283  
`\Gamma`, 90  
`\gamma`, 90  
`\gcd`, 91  
`\gdef`, 25, 81  
`\ge`, 91  
`generatedvi` (raweb action), 247  
`generateps` (raweb action), 247  
`\GenericError`, 191  
`\GenericInfo`, 191  
`\GenericWarning`, 191  
`\genfrac`, 92, 95, 101  
`\geq`, 91  
`\geqslant`, 91  
`german` (package), 158  
`\gets`, 91  
`\gg`, 91  
`\gll`, 195  
`\glll`, 195  
`\glo`, 187  
`\global`, 21, 48, 80, 83, 177  
`\globaldefs`, 40  
`glossaire`, 187, 222  
`\glue`, 36  
`\glueexpr`, 42, 276  
`\glueshrink`, 42, 275  
`\glueshrinkorder`, 42, 275  
`\gluestretch`, 42, 275  
`\gluestretchorder`, 42, 275  
`\gluetomu`, 42, 275  
`\@gobble`, 84, 190  
`\@gobblefour`, 84  
`\@gobbletwo`, 84, 190  
`\goodbreak`, 58  
`\grave`, 92

---

<code>\gtrless</code> , 91	<code>\hyphenation</code> , 191, 197
<code>\guillemets</code> , 145	<code>\hyphenchar</code> , 34, 41
<code>\guillemotleft</code> , 19, 146	<code>\hyphenpenalty</code> , 39
<code>\guillemotright</code> , 19, 147	
	<code>\I</code> , 281
<code>\H</code> , 84, 135	<code>\i</code> , 90, 135, 146
<code>\h</code> , 84, 135	<code>\ieme</code> , 54, 145
<code>hacknotitle</code> (tralics option), 201	<code>\iemes</code> , 54, 145
<code>\halign</code> , 244	<code>\ier</code> , 54, 145
<code>\hangafter</code> , 40	<code>\iere</code> , 54, 145
<code>\hangindent</code> , 43	<code>\ieres</code> , 145
<code>\hat</code> , 92	<code>\iers</code> , 54, 145
<code>\hbadness</code> , 40	<code>\if</code> , 67, 83, 167
<code>\hbar</code> , 90	<code>\@ifbempty</code> , 168
<code>\hbox</code> , 25, 95, 100, 113, 184	<code>\ifcase</code> , 67, 83, 167
<code>\hb@xt@</code> , 282	<code>\ifcat</code> , 67, 83, 167
<code>\hdots</code> , 92	<code>\@ifclasslater</code> , 161
<code>\headercr</code> , 282	<code>\@ifclassloaded</code> , 161
<code>\headheight</code> , 43	<code>\@ifclasswith</code> , 161
<code>\headsep</code> , 43	<code>\ifcsname</code> , 66, 70, 167, 272
<code>\heartsuit</code> , 91	<code>\ifdefined</code> , 66, 167, 272
<code>\@height</code> , 281	<code>\ifdim</code> , 68, 83, 167
<code>\heightof</code> , 172	<code>\ifeof</code> , 69, 83, 167, 176
<code>\hexnumber@</code> , 84, 167	<code>\iff</code> , 91
<code>\hfil</code> , 56	<code>\iffalse</code> , 68, 83, 167
<code>\hfill</code> , 56	<code>\IfFileExists</code> , 179
<code>\hfilneg</code> , 56	<code>\iffontchar</code> , 66, 167, 272
<code>\hfuzz</code> , 42	<code>\iffptest</code> , 150
<code>\hideskip</code> , 39	<code>\ifhbox</code> , 70, 83, 167
<code>\hl</code> , 139	<code>\ifhmode</code> , 70, 83, 167
<code>\hline</code> , 243	<code>\ifinner</code> , 70, 83, 167
<code>\hoffset</code> , 43	<code>\ifmmode</code> , 70, 83, 167
<code>\holdinginserts</code> , 41	<code>\@ifnextchar</code> , 190
<code>\hom</code> , 91	<code>\ifnextchar</code> , 72
<code>\hookleftarrow</code> , 92	<code>\ifnum</code> , 64, 68, 83, 167
<code>\hookrightarrow</code> , 91	<code>\ifodd</code> , 83, 167
<code>howpublished</code> (bibtex field), 115	<code>\@ifpackagelater</code> , 161
<code>\Href</code> , 181	<code>\@ifpackageloaded</code> , 161
<code>\href</code> , 181	<code>\@ifpackagewith</code> , 162
<code>\hrule</code> , 192	<code>\@ifstar</code> , 190
<code>\hsize</code> , 42	<code>\@iftempty</code> , 168
<code>\hskip</code> , 55, 56, 95	<code>\ifthenelse</code> , 66, 173
<code>\hspace</code> , 55, 56, 95, 100	<code>\iftrue</code> , 68, 83, 167
<code>\hss</code> , 56	<code>\@ifundefined</code> , 189
<code>\ht</code> , 44	<code>\ifvbox</code> , 70, 83, 167
<code>\htmladdnormallink</code> , 181	<code>\ifvmode</code> , 70, 83, 167
<code>\htmladdnormallinkfoot</code> , 181	<code>\ifvoid</code> , 70, 83, 167
<code>\htmlimage</code> , 191	<code>\ifx</code> , 30, 71, 83, 167
<code>htmlonly</code> , 191	<code>\ignorespaces</code> , 54, 180
<code>\HTMLset</code> , 191	<code>\IJ</code> , 144
<code>\Huge</code> , 54, 248	<code>\ij</code> , 144
<code>\huge</code> , 54, 248	<code>\Im</code> , 90



---

`\imath`, 91  
`\immediate`, 191  
`\in`, 91  
in (unit), 45  
inbook (bibtex entry type), 114  
`\include`, 83, 179  
`\includegraphics`, 190, 259, 284  
incollection (bibtex entry type), 114  
`\indent`, 57  
`\index`, 262  
`\inf`, 91  
`\infty`, 90  
`\injl`, 91  
`\Input`, 179  
`\input`, 26, 83, 167, 179  
input-path (tralics option), 201  
`\input@encoding`, 133  
`\input@encoding@default`, 133  
`\input@encoding@val`, 133  
`\InputClass`, 161  
`\inputencoding`, 134  
`\inputencodingname`, 134  
inputfile (tralics option), 201  
`\InputIfFileExists`, 179  
`\inputlineno`, 41  
`\insert`, 181, 197  
`\insertbibliohere`, 124  
`\insertpenalties`, 40  
institution (bibtex field), 115  
`\int`, 14, 91  
`\interactionmode`, 277  
interactivebib (tralics option), 201  
interactivemath (tralics option), 201  
`\interdisplaylinepenalty`, 41  
`\interfootnotelinepenalty`, 41  
`\interlinepenalties`, 41, 278  
`\interlinepenalty`, 40  
`\intertext`, 14  
`\intertextsep`, 39  
`\iota`, 90  
isbn (bibtex field), 115  
`\isdefined`, 70  
`\isodd`, 172  
`\isproject`, 221  
isrn (bibtex field), 115  
issn (bibtex field), 115  
`\isundefined`, 173  
`\it`, 9, 54, 93  
`\@@item`, 189  
`\@item`, 188  
`\item`, 8, 22, 188  
`\itemindent`, 43  
`itemize`, 187  
`\@itemlabel`, 187, 188  
`\itemsep`, 38  
`\itshape`, 9, 54, 249  
`\@iwhiledim`, 169  
`\@iwhilenum`, 169, 171  
`\@iwhilesw`, 169  
`\@ixpt`, 281  
  
`\J`, 281  
`\j`, 146  
jan (bibtex field), 114  
`\jmath`, 91  
`\jobname`, 83, 166  
`\Join`, 92  
`\jot`, 44  
journal (bibtex field), 115  
jul (bibtex field), 114  
jun (bibtex field), 114  
  
`\k`, 84, 135  
`\kappa`, 90  
`\ker`, 91  
`\kern`, 36, 95  
key  
    (bibtex field), 110, 115, 117  
    cite, 109  
    print, 109, 110  
    sort, 110, 117  
`\keyword`, 234  
keywords, 210, 222  
  
`\L`, 144  
`\l`, 144  
`\label`, 94, 182  
`\labelitemi`, 187, 282  
`\labelitemii`, 187, 282  
`\labelitemiii`, 187, 282  
`\labelitemiv`, 187, 282  
`\labelsep`, 39  
`\labelwidth`, 43  
`\Lambda`, 90  
`\lambda`, 90  
`\land`, 91  
lang\_en, 206, 247  
lang\_fr, 206, 247  
`\langle`, 90, 92  
Language, 206, 247  
`\language`, 41, 158  
`\LARGE`, 54, 248  
`\Large`, 54, 248  
`\large`, 54, 248  
`\lastbox`, 185

---

`\lastkern`, 41, 180  
`\lastlinefit`, 41  
`\lastnodetype`, 41, 277  
`\lastpenalty`, 41, 180  
`\lastskip`, 41, 180  
`\LaTeX`, 145  
`LaTeXonly`, 191  
`latexonly`, 191  
`latin1` (tralics option), 201  
`\lbrace`, 90, 92  
`\lbrack`, 281  
`\lccode`, 44, 132  
`\lceil`, 90, 92  
`\ldots`, 90, 146  
`\le`, 91  
`\leaders`, 184  
`\leavevmode`, 57, 112  
`\left`, 6, 7, 94  
`\Leftarrow`, 91  
`\leftarrow`, 91  
`\leftharpoondown`, 91  
`\leftharpoonup`, 91  
`\lefthyphenmin`, 41  
`\leftline`, 191  
`\leftmargin`, 43  
`\leftmargini`, 43  
`\leftmarginii`, 43  
`\leftmarginiii`, 43  
`\leftmarginiv`, 43  
`\leftmarginv`, 43  
`\leftmarginvi`, 43  
`leftquote` (tralics option), 202  
`\Leftrightarrow`, 91  
`\leftrightharpoonup`, 91  
`\leftskip`, 38  
`\lengthtest`, 173  
`\leq`, 91  
`\leqno`, 86  
`\leqslant`, 91  
`\let`, 29  
`\lfloor`, 90, 92  
`\lfoot`, 189  
`\lg`, 91  
`\lgroup`, 90, 92  
`\lhead`, 189  
`\lim`, 91  
`\liminf`, 91  
`\limits`, 89, 101  
`\limsup`, 91  
`\line`, 233, 263  
`\linebreak`, 58  
`\linepenalty`, 39  
`\lineskip`, 38  
`\lineskiplimit`, 42  
`\linethickness`, 233, 263  
`\linewidth`, 44  
`list`, 187  
`\listparindent`, 39  
`\ll`, 91  
`\lmoustache`, 90, 92  
`\ln`, 91  
`\lnot`, 91  
`\LoadClass`, 161  
`\LoadClassWithOptions`, 161  
`\localisation`, 221  
`\log`, 91  
`log-file` (tralics option), 202  
`\long`, 21, 25, 62  
`\Longleftarrow`, 92  
`\longleftarrow`, 92  
`\Longleftrightarrow`, 92  
`\longleftrightarrow`, 92  
`\longmapsto`, 92  
`\Longrightarrow`, 92  
`\longrightarrow`, 92  
`\Loop`, 64  
`\loop`, 64, 84, 171  
`\looseness`, 40  
`\lor`, 91  
`\lower`, 105  
`\lowercase`, 81, 143  
`\lq`, 281  
`\lsc`, 190  
  
`\@M`, 279  
`\@m`, 279  
`\mag`, 40  
`\mainmatter`, 183  
`\makeatletter`, 28  
`\makeatother`, 28  
`\makebox`, 95, 230  
`makedvi` (raweb action), 247  
`makefo` (raweb action), 247  
`makehtml` (raweb action), 247  
`\MakeLowercase`, 144  
`makepdf` (raweb action), 247  
`\MakeUppercase`, 144  
`manual` (bibtex entry type), 114  
`\mapsto`, 91  
`mar` (bibtex field), 114  
`\marginparpush`, 43  
`\marginparsep`, 43  
`\marginparwidth`, 43  
`\mark`, 180

---

`\markboth`, 282  
`\markright`, 282  
`\marks`, 279  
masterthesis (bibtex entry type), 114  
masterthesis (bibtex entry type), 114  
`math`, 88  
`\mathaccent`, 105, 137  
`\mathattribute`, 104  
`\mathbb`, 13, 93  
`\mathbbm`, 94  
`\mathbf`, 13, 93  
`\mathbin`, 15, 101  
`\mathcal`, 13, 93  
`\mathchar`, 104  
`\mathchardef`, 29, 104  
`\mathchoice`, 16, 92, 95, 101  
`\mathclose`, 15, 101  
`\mathcode`, 44, 105, 132  
`\mathfrak`, 93  
`\mathinner`, 15, 101  
`\mathit`, 93  
`\mathmi`, 104  
`\mathmn`, 104  
`\mathmo`, 104  
`\mathnormal`, 93  
`\mathop`, 15, 100, 101  
`\mathopen`, 15, 101  
`\mathord`, 15, 101  
`\mathpunct`, 15, 101  
`\mathrel`, 15, 101  
`\mathring`, 92  
`\mathrm`, 93  
`\mathsf`, 93  
`\mathsurround`, 42  
`\mathtt`, 93  
mathvariant (tralics option), 202  
`\@mathversion`, 41  
`\mathversion`, 13, 41  
`matrix`, 89  
`\max`, 91  
`\maxdeadcycles`, 40  
`\maxdepth`, 42  
`\maxdimen`, 44  
`may` (bibtex field), 114  
`\mbox`, 95, 100, 230  
`\mdseries`, 54  
`\meaning`, 83, 166  
`\medbreak`, 58  
`\medmuskip`, 38, 55  
`\medskip`, 56  
`\medskipamount`, 39  
`\message`, 181  
`\MessageBreak`, 191  
`mfenced_separator_val`, 258  
`\mho`, 91  
`\@Mi`, 279  
`\mid`, 91  
`\middle`, 279  
`\@Mii`, 279  
`\@Miii`, 279  
`\min`, 91  
`minipage`, 232  
`\@minus`, 281  
`misc` (bibtex entry type), 114  
`\@Miv`, 279  
`\mkern`, 56, 95, 192  
`\@MM`, 279  
`mm` (unit), 45  
`\@mod`, 91  
`\mod`, 282  
`\models`, 91  
`module`, 222  
`\moduleref`, 223  
`\m@ne`, 279  
`\month`, 40  
`month` (bibtex field), 115  
`moreinfo`, 222  
`\motcle`, 234  
`\motscle`, 9  
`motscle`, 210, 222  
`\moveleft`, 185  
`\moveright`, 185  
`\mp`, 91  
`mpfootnote` (counter), 42  
`\mskip`, 56, 95  
`\mu`, 90  
`\muexpr`, 42  
`\multicolumn`, 96, 243  
`\multimap`, 91  
`\multiply`, 38, 48  
`\multipt`, 233, 263  
`\multispan`, 84  
`multline`, 88  
`\muskip`, 17, 44  
`\muskipdef`, 29  
`\mutoglu`, 42, 275  
  
`\nabla`, 90  
`\@namedef`, 32  
`\@nameuse`, 32  
`\natural`, 90, 146  
`\@ne`, 62  
`\ne`, 91  
`\narrow`, 91

---

`\NeedsTeXFormat`, 162  
`\neg`, 91  
`\neq`, 91  
`\newboolean`, 68, 172  
`\newbox`, 32  
`\newcolumnntype`, 241  
`\newcommand`, 24  
`\newcount`, 32  
`\newcounter`, 32, 48  
`\newdimen`, 32  
`\newenvironment`, 26  
`\newhelp`, 32  
`\newif`, 68  
`\newinsert`, 32  
`\newlanguage`, 32  
`\newlength`, 49, 69  
`\newline`, 57  
`\newlinechar`, 40  
`\newmuskip`, 32  
`\newread`, 32  
`\newsavebox`, 32  
`\newskip`, 32  
`\newtheorem`, 8, 173  
`\newtoks`, 32, 47  
`\newwrite`, 32  
`\NG`, 144  
`\ng`, 144  
`\ni`, 91  
`\@nil`, 22  
`\@nnil`, 283  
`\No`, 145  
`\no`, 145  
`no_footnote_hack`, 248  
`\noalign`, 244  
`nobibyearerror` (tralics option), 202  
`nobibyearmodify` (tralics option), 202  
`\noboundary`, 181  
`\nobreak`, 191  
`\nobreakspace`, 55, 93, 139, 146  
`\nocentering`, 38, 181, 232  
`\nocite`, 111, 115  
`noconfig`, 202  
`\node`, 193  
`\nodebox`, 194  
`\nodecircle`, 194  
`\nodeconnect`, 194  
`\nodecurve`, 194  
`\nodeoval`, 194  
`\nodepoint`, 194  
`\nodetriangle`, 194  
`noentnames` (tralics option), 202  
`noetex` (tralics option), 202  
`\noexpand`, 32, 34, 67, 83, 166  
`nofirst`, 142, 143  
`noindent`, 142, 143  
`\noindent`, 57, 88  
`\nolimits`, 89, 101  
`\nolinebreak`, 58, 101  
`\@nomathml`, 41, 99  
`nomathml` (tralics option), 202  
`nomathvariant` (tralics option), 202  
`\nonscript`, 100  
`\nonstopmode`, 190  
`\nonumber`, 101  
`\noopsort`, 117  
`\nopagebreak`, 58  
`\normalbaselineskip`, 39  
`\normalfont`, 9, 54  
`\normallineskip`, 39  
`\normallineskiplimit`, 43  
`\normalsize`, 54, 248  
`nostraightquotes` (tralics option), 202  
`\NOT`, 173  
`\not`, 91, 173  
`note` (bibtex field), 115  
`\notin`, 91  
`\@notprerr`, 192  
`\notrivialmath`, 41  
`notrivialmath` (tralics option), 202  
`noundefmac` (tralics option), 202  
`nov` (bibtex field), 114  
`noxmlerror` (tralics option), 202  
`nozerowidthelt` (tralics option), 202  
`nozerowidthspace` (tralics option), 202  
`\nu`, 90  
`\null`, 282  
`\nulldelimiterspace`, 42  
`\nullfont`, 34  
`\number`, 32, 48, 62, 78, 83, 166  
`number` (bibtex field), 115  
`\numberedverbatim`, 143  
`\Numero`, 145  
`\numero`, 145  
`\numexpr`, 42, 276  
`\narrow`, 91  
  
`\O`, 144, 145  
`\o`, 144, 145  
`\obeylines`, 282  
`\obeyspaces`, 282  
`oct` (bibtex field), 114  
`\oddsidemargin`, 43  
`\odot`, 91  
`\OE`, 144

---

`\oe`, 144  
`oe1` (tralics option), 202  
`oe1a` (tralics option), 202  
`oe8` (tralics option), 202, 203  
`oe8a` (tralics option), 202  
`\of`, 95  
`\og`, 145  
`\oint`, 91  
`\Omega`, 90  
`\omega`, 90  
`\ominus`, 91  
`\omit`, 243  
`\@ne`, 62, 279  
`\@onlypreamble`, 158  
`\openin`, 176  
`\openout`, 178  
`\operatorname`, 104  
`\oplus`, 91  
`\OptionNotUsed`, 162  
`\OR`, 173  
`\or`, 83, 167, 173  
organization (bibtex field), 115  
`\oslash`, 91  
`\otimes`, 91  
`\outer`, 21  
`\output`, 39  
output-dir (tralics option), 202  
output-file (tralics option), 203  
`\outputpenalty`, 40  
`\oval`, 233, 263  
`\over`, 16, 23, 99  
`\overbrace`, 92  
`\overfullrule`, 43  
`\overleftarrow`, 92  
`\overline`, 92  
`\overrightarrow`, 92  
`\overset`, 92  
`\overwithdelims`, 99  
`\owns`, 91  
  
`\P`, 145  
`\p@`, 43  
`\PackageError`, 191  
`\PackageErrorNoLine`, 191  
`\PackageInfo`, 191  
`\PackageWarning`, 191  
`\PackageWarningNoLine`, 191  
page (counter), 42, 283  
`\pagebreak`, 58  
`\pagedepth`, 44  
`\pagediscards`, 279  
`\pagefillstretch`, 44  
  
`\pagefillstretch`, 44  
`\pagefilstretch`, 44  
`\pagegoal`, 44  
`\pageref`, 182, 283  
pages (bibtex field), 115  
`\pageshrink`, 44  
`\pagestretch`, 44  
`\pagetotal`, 44  
`\paperheight`, 44  
`\paperwidth`, 44  
`\par`, 57, 80  
`\paragraph`, 182, 252  
paragraph (counter), 42  
`\paragraphmark`, 283  
`\parallel`, 91  
param (tralics option), 203  
`\parbox`, 232, 240  
`\parfillskip`, 38  
`\parindent`, 42, 57  
`\parsep`, 39  
`\parshape`, 39  
`\parshapedimen`, 42, 277  
`\parshapeindent`, 42, 277  
`\parshapelength`, 42, 277  
`\parskip`, 38  
`\part`, 182, 252  
part (counter), 42  
`\partial`, 90  
participant, 222, 223  
participante, 223  
participantes, 223  
participants, 223  
`\partopsep`, 39  
`\PassOptionToClass`, 162  
`\PassOptionToPackage`, 162  
`\patterns`, 191, 197  
`\pausing`, 40  
pc (unit), 45  
`\penalty`, 36, 180  
`\perp`, 91  
`\pers`, 223  
`\@persA`, 223  
`\persA`, 223  
`\@persB`, 224  
`\persB`, 223  
phdthesis (bibtex entry type), 114  
`\Phi`, 90  
`\phi`, 90  
`\Pi`, 90  
`\pi`, 90  
picture, 230  
`\@plus`, 281

`\pm`, 91  
`pmatrix`, 89  
`\pmod`, 282  
`\pod`, 282  
`\postdisplaypenalty`, 40  
`\pounds`, 146  
`\Pr`, 91  
`preamble` (bibtex keyword), 114  
`\prec`, 91  
`\preceq`, 91  
`\predisplaydirection`, 41, 279  
`\predisdisplaypenalty`, 39  
`\predisplaysize`, 42  
`\pretolerance`, 39  
`\prevdepth`, 44  
`\prevgraf`, 44  
`\prime`, 15  
`\ProcessOptions`, 162  
`\proclaim`, 8  
`\prod`, 91  
`profession_vals`, 247  
`\project`, 221  
`\projet`, 221  
`\projlim`, 91  
`\propto`, 91  
`\protect`, 34, 191  
`\protected`, 21, 273  
`\provideboolean`, 172  
`\providecommand`, 25  
`\ProvidesClass`, 161  
`\ProvidesFile`, 161  
`\ProvidesPackage`, 161  
`ps` (tralics option), 203  
`\psfig`, 190  
`\Psi`, 90  
`\psi`, 90  
`pt` (unit), 45  
`publisher` (bibtex field), 115  
`\put`, 232, 263  
`\p@xxx`, 48  
  
`\qbezier`, 232  
`\qqquad`, 55, 90, 146  
`\quad`, 55, 90, 146  
`quotation`, 191, 259  
`quote`, 191, 259  
  
`\r`, 84, 135  
`ra-debug` (tralics option), 203  
`\radical`, 105  
`\raggedleft`, 191  
`\raggedright`, 191  
  
`\raise`, 105  
`\RALabel`, 191  
`\rangle`, 90, 92  
`RAsection`, 223  
`\RAstartprojet`, 222  
`\ratio`, 49, 172  
`rawhtml`, 191  
`\rbrace`, 90, 92  
`\rbrack`, 281  
`\rceil`, 90, 92  
`\Re`, 90  
`\read`, 176  
`\readline`, 278  
`\real`, 49, 172  
`\@reevaluate`, 192  
`\ref`, 182  
`refer` (bibtex type), 111, 115  
`\refercite`, 111  
`\refname`, 124, 282  
`\refstepcounter`, 50, 84  
`\relax`, 34, 46, 48, 64, 94, 177, 276  
`\relpenalty`, 39  
`\renewcommand`, 25  
`\renewenvironment`, 25  
`\repeat`, 64, 67  
`\RequirePackage`, 161  
`\RequirePackageWithOptions`, 161  
`\rfloor`, 90, 92  
`\rfoot`, 189  
`\rgroup`, 90, 92  
`\rhead`, 189  
`\rho`, 90  
`\right`, 7, 94  
`\Rightarrow`, 91  
`\rightarrow`, 91  
`\rightharpoondown`, 91  
`\rightharpoonup`, 91  
`\rightthyphenmin`, 41  
`\rightline`, 191  
`\rightmargin`, 43  
`rightquote` (tralics option), 203  
`\rightskip`, 38  
`\rm`, 54, 93, 100  
`\rmfamily`, 53  
`\rmoustache`, 90, 92  
`\@Roman`, 48  
`\Roman`, 48, 84  
`\@roman`, 48  
`\roman`, 48, 84  
`\Romannumeral`, 48, 83, 167  
`\romannumeral`, 48, 78, 83, 166  
`\root`, 92

---

`\rotatebox`, 231, 254  
`\rq`, 281  
`\rrrt`, 181  
  
`\S`, 145  
`\@sanitize`, 283  
`\SaveVerb`, 84, 139  
`\savingdiscards`, 41, 279  
`\savinghyphcodes`, 41, 278  
`\sb`, 29  
`\sbox`, 76  
`\sc`, 54  
`\scalebox`, 231, 253  
`\scaleput`, 232, 263  
`\scantokens`, 273  
school (bibtex field), 115  
`\scriptscriptfont`, 34  
`\scriptscriptstyle`, 15, 95, 100  
`\scriptscritpfont`, 53  
`\scriptsize`, 54, 248  
`\scriptspace`, 42  
`\scriptstyle`, 15, 95, 100  
`\scritpfont`, 53  
`\scrollmode`, 190  
`\scshape`, 54, 249  
`\searrow`, 91  
`\sec`, 91  
`\@secondoftwo`, 84, 166  
`\section`, 6, 182, 252  
section (counter), 42  
section\_vals, 247  
`\sectionmark`, 283  
`\selectfont`, 9, 52  
sep (bibtex field), 114  
series, 115  
`\seriesdefault`, 281  
`\setboolean`, 68, 172  
`\setbox`, 29  
setcounter, 48  
`\setcounter`, 84, 172  
`\setlanguage`, 181  
`\setlength`, 49, 84, 171  
`\setminus`, 91  
`\@setmode`, 283  
`\sf`, 54, 93  
`\sfcode`, 44, 132  
`\sffamily`, 53, 249  
`\shapedefault`, 281  
`\sharp`, 90, 146  
shell-escape (tralics option), 203  
`\shipout`, 44, 181  
`\show`, 36, 225  
  
`\showbox`, 35, 225  
`\showboxbreadth`, 36, 40  
`\showboxdept`, 36, 40  
`\showgroups`, 273  
`\showlists`, 225  
`\showthe`, 37, 38, 225  
`\showtokens`, 278  
`\Sigma`, 90  
`\sigma`, 90  
silent (tralics option), 203  
`\sim`, 91  
`\simeq`, 91  
`\sin`, 14, 91  
`\sinh`, 91  
`\sixin@n`, 279  
`\skewchar`, 34, 41  
`\skip`, 17, 44, 69  
`\skipdef`, 29  
`\sl`, 54, 93  
`\slash`, 145  
`\sloppy`, 191  
`\slshape`, 54, 249  
`\small`, 54, 248  
`\smallbreak`, 58  
`\smallint`, 91  
`\smallskip`, 56  
`\smallskipamount`, 39  
`\smile`, 91  
`\so`, 139  
`\SortNoop`, 117  
`\sortnoop`, 117  
`\sp`, 29  
sp (unit), 45  
space, 5, 7, 22, 26, 61, 73, 80  
`\space`, 54, 55, 74, 282  
`\spacefactor`, 40  
`\@spaces`, 282  
`\spaceskip`, 38, 40  
`\spadesuit`, 91  
`\span`, 244  
`\special`, 36, 191, 197  
split, 88  
`\splitbotmark`, 83, 166  
`\splitbotmarks`, 166, 279  
`\splitdiscards`, 279  
`\splitfirstmark`, 83, 166  
`\splitfirstmarks`, 166, 279  
`\splitmaxdepth`, 42  
`\splittopskip`, 38  
`\sqcap`, 91  
`\sqcup`, 91  
`\sqrt`, 92

$\backslash$ sqssubset, 91  
 $\backslash$ sqssubseteq, 91  
 $\backslash$ sqsupset, 91  
 $\backslash$ sqssubseteq, 91  
 $\backslash$ square, 91  
 $\backslash$ SS, 144, 145  
 $\backslash$ ss, 144, 145  
 $\backslash$ st, 139  
 $\backslash$ stackrel, 92  
 $\backslash$ star, 91  
 $\backslash$ stepcounter, 50  
 $\backslash$ @stpelt, 50, 84  
 $\backslash$ string, 25, 32, 34, 83, 166  
string (bibtex keyword), 114  
 $\backslash$ strip@prefix, 84, 167  
subequations, 191  
 $\backslash$ subfigure, 254  
 $\backslash$ subitem, 191  
 $\backslash$ subparagraph, 182, 252  
subparagraph (counter), 42  
 $\backslash$ subparagraphmark, 283  
 $\backslash$ subsection, 6, 182, 252  
subsection (counter), 42  
 $\backslash$ subsectionmark, 283  
 $\backslash$ subset, 91  
 $\backslash$ subseteq, 91  
 $\backslash$ subsubsection, 182, 252  
subsubsection (counter), 42  
 $\backslash$ subsubsectionmark, 283  
 $\backslash$ succ, 91  
 $\backslash$ succeq, 91  
 $\backslash$ sum, 91  
 $\backslash$ sup, 91  
 $\backslash$ supset, 91  
 $\backslash$ supseteq, 91  
 $\backslash$ surd, 90  
 $\backslash$ swarrow, 91  
  
 $\backslash$ T, 84, 135  
 $\backslash$ tabcolsep, 43  
table, 238  
 $\backslash$ tabskip, 38  
tabular, 238, 244  
 $\backslash$ tagcurve, 233, 263  
 $\backslash$ tan, 91  
 $\backslash$ tanh, 91  
 $\backslash$ tau, 90  
 $\backslash$ tbinom, 95  
tel (tralics option), 203  
tela (tralics option), 203  
te8a (tralics option), 203  
techreport (bibtex entry type), 114  
  
 $\backslash$ @tempdima, 44  
 $\backslash$ @tempskipa, 39  
 $\backslash$ @tempskipb, 39  
 $\backslash$ @temptokena, 39  
 $\backslash$ TeX, 145  
 $\backslash$ texorpdfstring, 5  
 $\backslash$ text, 95, 100  
 $\backslash$ textacutedbl, 147  
 $\backslash$ textascendercompwordmark, 149  
 $\backslash$ textasciicircum, 147  
 $\backslash$ textasciicaron, 147  
 $\backslash$ textasciibreve, 147  
 $\backslash$ textasciicaron, 147  
 $\backslash$ textasciicedilla, 147  
 $\backslash$ textasciicircum, 145  
 $\backslash$ textasciidieresis, 146  
 $\backslash$ textasciigrave, 145  
 $\backslash$ textasciimacron, 147  
 $\backslash$ textasciitilde, 146  
 $\backslash$ textasteriskcentered, 148  
 $\backslash$ textbackslash, 145  
 $\backslash$ textbaht, 147  
 $\backslash$ textbardbl, 147  
 $\backslash$ textbf, 54  
 $\backslash$ textbigcircle, 148  
 $\backslash$ textblank, 149  
 $\backslash$ textborn, 149  
 $\backslash$ textbraceleft, 146  
 $\backslash$ textbraceright, 146  
 $\backslash$ textbrokenbar, 146  
 $\backslash$ textbullet, 147  
 $\backslash$ textcapitalcompwordmark, 148  
 $\backslash$ textcelsius, 148  
 $\backslash$ textcent, 146  
 $\backslash$ textcentoldstyle, 148  
 $\backslash$ textcircledP, 148  
 $\backslash$ textcolonmonetary, 147  
 $\backslash$ textcompwordmark, 148  
 $\backslash$ textcopyleft, 148  
 $\backslash$ textcopyright, 146  
 $\backslash$ textcurrency, 146  
 $\backslash$ textdagger, 147  
 $\backslash$ textdaggerdbl, 147  
 $\backslash$ textdblhyphen, 149  
 $\backslash$ textdegree, 147  
 $\backslash$ textdied, 149  
 $\backslash$ textdiscount, 149  
 $\backslash$ textdiv, 146  
 $\backslash$ textdivorced, 149  
 $\backslash$ textdollar, 145  
 $\backslash$ textdollaroldstyle, 148  
 $\backslash$ textdong, 148



---

`\textdownarrow`, 148  
`\texteightoldstyle`, 148  
`\textellipsis`, 147  
`\textemdash`, 147  
`\textendash`, 147  
`\textestimated`, 148  
`\texteuro`, 19, 84, 148  
`\textexclamdown`, 146  
`\textfiveoldstyle`, 148  
`\textfloatsep`, 39  
`\textflorin`, 147  
`\textfont`, 34, 53  
`\textfouroldstyle`, 148  
`\textfraction`, 281  
`\textfractionsolidus`, 147  
`\textfrenchfranc`, 147  
`\textgravedbl`, 147  
`\textgreater`, 146  
`\textguarani`, 149  
`\textheight`, 43  
`\textinterrobang`, 147  
`\textinterrobangdown`, 149  
`\textit`, 54, 248  
`\textlangle`, 148  
`\textlbrackdbl`, 148  
`\textleaf`, 149  
`\textleftarrow`, 148  
`\textless`, 146  
`\textlira`, 148  
`\textlnot`, 146  
`\textlquill`, 147  
`\textmarried`, 149  
`\textmd`, 54  
`\textmho`, 148  
`\textmu`, 147  
`\textmusicalnote`, 148  
`\textnaira`, 148  
`\textnineoldstyle`, 148  
`\textnormal`, 54  
`\textnospace`, 145  
`\textnumero`, 148  
`\textohm`, 148  
`\textonehalf`, 147  
`\textoneoldstyle`, 148  
`\textonequarter`, 147  
`\textonesuperior`, 147  
`\textopenbullet`, 148  
`\textordfeminine`, 146  
`\textordmasculine`, 147  
`\textparagraph`, 147  
`\textperiodcentered`, 147  
`\textpertenthousand`, 147  
`\textperthousand`, 147  
`\textpeso`, 148  
`\textpilcrow`, 147  
`\textpm`, 147  
`\textquestiondown`, 147  
`\textquotedblleft`, 147  
`\textquotedblright`, 147  
`\textquoteleft`, 147  
`\textquoteright`, 147  
`\textquotesingle`, 148  
`\textquotestraightbase`, 148  
`\textquotestraightdblbase`, 148  
`\textrangle`, 148  
`\textbrackdbl`, 148  
`\textrecipe`, 148  
`\textreferencemark`, 147  
`\textregistered`, 147  
`\textrightarrow`, 148  
`\textrm`, 53  
`\textrquill`, 147  
`\textsc`, 54  
`\textsection`, 146  
`\textservicemark`, 148  
`\textsevenoldstyle`, 148  
`\textsf`, 53  
`\textsixoldstyle`, 148  
`\textsl`, 54  
`\textsofthyphen`, 147  
`\textsterling`, 146  
`\textstyle`, 15, 95, 100  
`\textsubscript`, 98, 249  
`\textsuperscript`, 54, 98, 249  
`\textsurd`, 148  
`\textthreeoldstyle`, 148  
`\textthreequarters`, 147  
`\textthreequartersemdash`, 148  
`\textthreesuperior`, 147  
`\texttildelow`, 148  
`\texttimes`, 146  
`\texttrademark`, 148  
`\texttt`, 53  
`\texttwelveudash`, 148  
`\texttwooldstyle`, 148  
`\texttwosuperior`, 147  
`\textunderscore`, 84, 146  
`\textup`, 54  
`\textuparrow`, 148  
`\textvisiblespace`, 139, 145  
`\textwidth`, 43  
`\textwon`, 148  
`\textyen`, 146  
`\textzerooldstyle`, 148

`\TeXeTstate`, 41, 279  
`\tfrac`, 92, 101  
`\TH`, 144, 145  
`\th`, 144, 145  
`\thanks`, 182  
`\the`, 7, 32, 37, 38, 83  
`thebibliography`, 124  
`\@themark`, 282  
`\theme`, 221  
`theme_vals`, 247  
`\theorembodyfont`, 173  
`\theoremheaderfont`, 173  
`\theoremstyle`, 173  
`\thepage`, 283  
`\Theta`, 90  
`\theta`, 90  
`\thexxx`, 50  
`\thicklines`, 233, 263  
`\thickmuskip`, 38, 55  
`\thinlines`, 233, 263  
`\thinmuskip`, 38, 55  
`\thinspace`, 282  
`\@thirdofthree`, 282  
`\thr@@`, 279  
`\tilde`, 92  
`\time`, 40  
`\times`, 91  
`\tiny`, 54, 248  
`title` (bibtex field), 115  
`\to`, 91  
`\toappear`, 184  
`\today`, 282  
`\toks`, 44  
`\toksdef`, 29  
`\tolerance`, 39  
`\top`, 90  
`\topfraction`, 281  
`\toplevelsection`, 158  
`\topmargin`, 43  
`\topmark`, 83, 166  
`\topmarks`, 166, 279  
`topnumber` (counter), 42  
`\topsep`, 39  
`\topskip`, 38  
`totalnumber` (counter), 42  
`tpa_status` (tralics option), 203  
`\tracingall`, 225  
`\tracingassigns`, 41, 269  
`\tracingcommands`, 40, 225, 269  
`\tracinggroups`, 41, 269  
`\tracingifs`, 41, 269  
`\tracinglostchars`, 40, 225, 269  
`\tracingmacros`, 40, 225  
`\tracingmath`, 41, 225  
`\tracingnesting`, 41, 269  
`\tracingonline`, 40, 225  
`\tracingoutput`, 40, 225  
`\tracingpages`, 40, 225  
`\tracingparagraphs`, 40, 225  
`\tracingrestores`, 40, 225  
`\tracingscantokens`, 41, 269  
`\tracingstats`, 40, 225  
`\tralicsversion`, 83, 167  
`\triangle`, 90  
`\triangleleft`, 91  
`\triangleright`, 91  
`trivialmath` (tralics option), 203  
`true` (unit prefix), 45  
`\tt`, 5, 54, 93  
`\ttfamily`, 53, 249  
`\tw@`, 62  
`\tw@`, 279  
`type` (bibtex field), 115  
`type` (tralics option), 203  
`\typeout`, 181  
`\u`, 84, 135  
`\uccode`, 44, 132  
`\uchyph`, 40  
`\ul`, 139  
`\UndefineShortVerb`, 139  
`\underbrace`, 92  
`\underleftarrow`, 92  
`\underline`, 92  
`\underrightarrow`, 92  
`\underset`, 92  
`\unexpanded`, 166, 275  
`\unhbox`, 181  
`\unhcopy`, 181  
`\unitlength`, 43, 229  
`\unkern`, 181  
`\unless`, 66, 167, 272  
`\unnumberedverbatim`, 143  
`\unpenalty`, 181  
`unpublished` (bibtex entry type), 114  
`\unskip`, 55, 181  
`\unvbox`, 181  
`\unvcopy`, 181  
`\Uparrow`, 90, 92  
`\uparrow`, 90, 92  
`\Updownarrow`, 90, 92  
`\updownarrow`, 90, 92  
`\uplus`, 91  
`\uppercase`, 143

---

`\upshape`, 54, 249  
`\Upsilon`, 90  
`\upsilon`, 90  
`\UR`, 221  
`ur_vals`, 247  
`\url`, 181, 247, 255  
`url` (bibtex field), 115  
`url_font`, 247  
`\urlfont`, 247  
`use_all_sizes`, 248  
`use_font_elt`, 248  
`\usecounter`, 187  
`\usepackage`, 158, 161  
`usequotes` (tralics option), 203  
`\UseVerb`, 84, 139  
`\useverb`, 167  
`utf8` (tralics option), 204  
`utf8output` (tralics option), 204  
  
`\V`, 84, 135  
`\v`, 84, 135  
`\vadjust`, 181, 197  
`\valign`, 244  
`\value`, 48, 84  
`\varepsilon`, 90  
`\varkappa`, 90  
`\varphi`, 90  
`\varpi`, 90  
`\varrho`, 90  
`\varsigma`, 90  
`\vartheta`, 90  
`\vbadness`, 40  
`\vbox`, 184  
`\vcenter`, 105  
`\Vdash`, 91  
`\vdash`, 91  
`\vdots`, 92  
`\vec`, 92  
`\vector`, 233, 263  
`\vee`, 91  
`\verb`, 28  
`Verbatim`, 140  
`verbatim`, 140  
`\verbatimfont`, 140  
`\verbatimnumberfont`, 140  
`\verbatimprefix`, 140  
`verbose` (tralics option), 204  
`\verbprefix`, 140  
`version` (tralics option), 204  
`\Vert`, 90, 92  
`\vert`, 90, 92  
`\vfill`, 56  
  
`\vfill`, 56  
`\vfilneg`, 56  
`\vfuzz`, 42  
`\@viiipt`, 281  
`\@viipt`, 281  
`\@viipt`, 281  
`\@viipt`, 281  
`Vmatrix`, 89  
`vmatrix`, 89  
`\@vobeyspaces`, 282  
`\voffset`, 43  
`volume` (bibtex field), 115  
`\@vpt`, 281  
`\vrule`, 192  
`\vsize`, 42  
`\vskip`, 56, 95  
`\vspace`, 56, 95  
`\vsplit`, 185  
`\vss`, 56  
`\vtop`, 185  
  
`\wd`, 44  
`\wedge`, 91  
`\@whiledim`, 84, 169  
`\whiledo`, 169  
`\@whilenum`, 84, 169  
`\@whilesw`, 84, 169  
`\widehat`, 92  
`\widetilde`, 92  
`\widowpenalty`, 39  
`\@width`, 281  
`\widthof`, 172  
`\wp`, 90  
`\wr`, 91  
`wrapfigure`, 238  
`\write`, 34, 36, 178  
  
`\@xbegintheorem`, 173  
`\xbox`, 113, 184  
`\xdef`, 32, 33  
`\Xi`, 90  
`\xi`, 90  
`\@xiipt`, 281  
`\@xiipt`, 281  
`\@xivpt`, 281  
`\xleaders`, 184  
`xml` (tralics option), 204  
`xml-expanded-project-name`, 206  
`xml_accueil_name`, 255  
`xml_anchor_name`, 262  
`xml_arc_name`, 263  
`xml_backmatter_name`, 252  
`xml_bezier_name`, 263

xml\_biblio, 259  
 xml\_bigcircle\_name, 263  
 xml\_box\_name, 253  
 xml\_bpers\_name, 261  
 xml\_caps\_name, 249  
 xml\_caption\_name, 206, 254  
 xml\_cell\_name, 260  
 xml\_circle\_name, 263  
 xml\_cit\_name, 261  
 xml\_citation\_name, 261  
 xml\_citetype\_name, 261  
 xml\_cleaders\_name, 262  
 xml\_closecurve\_name, 263  
 xml\_composition\_ra\_name, 206, 255  
 xml\_curve\_name, 263  
 xml\_dashline\_name, 263  
 xml\_div0\_name, 252  
 xml\_div1\_name, 252  
 xml\_div2\_name, 252  
 xml\_div3\_name, 252  
 xml\_div4\_name, 252  
 xml\_div5\_name, 252  
 xml\_div6\_name, 252  
 xml\_dottedline\_name, 263  
 xml\_draline\_name, 263  
 xml\_fbox\_name, 253  
 xml\_figure\_env\_name, 254  
 xml\_figure\_name, 259  
 xml\_font\_bold, 249  
 xml\_font\_it, 249  
 xml\_font\_large, 248  
 xml\_font\_large1, 248  
 xml\_font\_large2, 248  
 xml\_font\_large3, 248  
 xml\_font\_large4, 248  
 xml\_font\_large5, 248  
 xml\_font\_medium, 249  
 xml\_font\_normalsize, 248  
 xml\_font\_roman, 249  
 xml\_font\_sansserif, 249  
 xml\_font\_sc, 249  
 xml\_font\_slanted, 249  
 xml\_font\_small, 248  
 xml\_font\_small1, 248  
 xml\_font\_small2, 248  
 xml\_font\_small3, 248  
 xml\_font\_small4, 248  
 xml\_font\_tt, 249  
 xml\_font\_upright, 249  
 xml\_footnote\_name, 248  
 xml\_frontmatter\_name, 252  
 xml\_glo\_name, 251  
 xml\_gloitem\_name, 251  
 xml\_graphics\_name, 254  
 xml\_head\_name, 252  
 xml\_hl\_name, 249  
 xml\_index\_name, 262  
 xml\_item\_name, 250  
 xml\_keywords\_name, 253  
 xml\_labelitem\_name, 250  
 xml\_leaders\_name, 262  
 xml\_leg\_name, 254  
 xml\_line\_name, 263  
 xml\_linethickness\_name, 263  
 xml\_list\_name, 250  
 xml\_mainmatter\_name, 252  
 xml\_mbox\_name, 254  
 xml\_multiput\_name, 263  
 xml\_oldstyle\_name, 249  
 xml\_oval\_name, 263  
 xml\_overline\_name, 249  
 xml\_pers\_name, 261  
 xml\_picture\_name, 262  
 xml\_pre\_name, 262  
 xml\_prenote\_name, 261  
 xml\_project\_name, 206, 255  
 xml\_put\_name, 263  
 xml\_ref\_name, 261  
 xml\_rotatebox\_name, 254  
 xml\_row\_name, 260  
 xml\_scalebox\_name, 253  
 xml\_scaleput\_name, 263  
 xml\_scaption\_name, 206, 254  
 xml\_so\_name, 249  
 xml\_st\_name, 249  
 xml\_sub\_name, 249  
 xml\_subfigure\_name, 254  
 xml\_sup\_name, 249  
 xml\_table\_env\_name, 254  
 xml\_Table\_name, 254  
 xml\_table\_name, 260  
 xml\_tableofcontents\_name, 262  
 xml\_tagcurve\_name, 263  
 xml\_term\_name, 253  
 xml\_texmath\_name, 262  
 xml\_texte\_name, 254  
 xml\_theindex\_name, 262  
 xml\_theorem\_head, 257  
 xml\_theorem\_name, 257  
 xml\_thicklines\_name, 263  
 xml\_thinlines\_name, 263  
 xml\_topic\_name, 254  
 xml\_topic\_title, 206  
 xml\_ul\_name, 249

---

xml\_underline\_name, 249  
xml\_vector\_name, 263  
xml\_xleaders\_name, 262  
xml\_xref\_name, 255  
xml\_xtheorem\_name, 256  
\XMLaddatt, 36, 187  
xmlall (tralics option), 204  
\XMLcurrentid, 36, 41, 112  
xmlelement, 186  
\xmlelt, 184  
\xmleptyelt, 184  
xmlfo (tralics option), 204  
xmlhtml (tralics option), 204  
\XMLlastid, 36, 41, 112  
\xmllatex, 184  
xmllint (tralics option), 204  
xmlonly, 191  
\XMLsolvecite, 112  
xmltex (tralics option), 204  
\@xpt, 281  
\xscale, 232, 263, 281  
\xscaley, 232, 263, 281  
\xspace, 84, 166  
\xspaceskip, 38, 40  
\@xvipt, 281  
\@xxpt, 281  
\@xxvpt, 281

\year, 40  
year (bibtex field), 115  
year (bibtex type), 111, 115  
year (tralics option), 204  
\yearcite, 111  
\yscale, 232, 263, 281  
\yscalex, 232, 263, 281

\z@skip, 39  
\zap@space, 84, 167  
\z@, 43, 279  
\zeta, 90



# Bibliography

- [1] David Carlisle, Michel Goossens, and Sebastian Rahtz. De XML à PDF avec `xmltex` et Passive $\TeX$ . In *Cahiers Gutenberg*, number 35-36, pages 79–114, 2000.
- [2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The  $\LaTeX$  companion*. Addison Wesley, 1993.
- [3] José Grimm. Outils pour la manipulation du rapport d'activité. Technical Report RT-0265, Inria, 2002.
- [4] Donald E. Knuth. *The  $\TeX$ book*. Addison Wesley, 1984.
- [5] Philippe Louarn. Une expérience d'utilisation de LaTeX : le Rapport d'activité de l'INRIA. *Cahiers Gutenberg*, (0):17–24, apr 1988.
- [6] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The  $\LaTeX$  companion, second edition*. Addison Wesley, 2004.
- [7] The Unicode Consortium. *The Unicode Standard, version 4.0*. Addison Wesley, 2003.
- [8] W3C. Extensible Markup Language (XML) 1.0 (Third Edition). <http://www.w3.org/TR/REC-xml/>, 1998. Third edition published in 2004.
- [9] W3C. Extensible Markup Language (XML) 1.1. <http://www.w3.org/TR/xml11/>, 2004.





# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	A short history of the Raweb . . . . .	3
1.2	Birth of Tralics . . . . .	4
1.3	Main objectives . . . . .	5
1.4	Notes on the distribution . . . . .	6
1.5	An example . . . . .	6
1.6	Some remarks on the <b>Translation</b> . . . . .	8
1.7	Category codes and characters . . . . .	10
1.8	Considerations about mathematics . . . . .	12
1.9	Some subtleties of T <sub>E</sub> X . . . . .	16
1.10	Language options . . . . .	18
<b>2</b>	<b>Expansion</b>	<b>21</b>
2.1	Defining new commands . . . . .	21
2.2	Defining commands in L <sup>A</sup> T <sub>E</sub> X . . . . .	24
2.3	Some small examples . . . . .	28
2.4	Variables in T <sub>E</sub> X . . . . .	34
2.5	All the variables . . . . .	38
2.6	Using the variables . . . . .	45
2.7	Counters . . . . .	48
2.8	Fonts . . . . .	51
2.9	Spaces . . . . .	54
2.10	Conditional expansion . . . . .	58
2.10.1	Constructing commands dynamically . . . . .	59
2.10.2	Iterating over lists . . . . .	59
2.10.3	Mapping a command . . . . .	61
2.10.4	Creating a list via pattern matching . . . . .	61
2.10.5	A variant of the previous problem . . . . .	63
2.10.6	Loops . . . . .	64
2.11	Conditionals in T <sub>E</sub> X . . . . .	66
2.11.1	Syntax of the conditionals . . . . .	66
2.11.2	Examples of conditional commands . . . . .	71

2.11.3	Testing the next token . . . . .	72
2.11.4	Reading a space . . . . .	73
2.11.5	Variants of the Map problem . . . . .	74
2.11.6	More examples . . . . .	76
2.11.7	Producing N asterisks in a row . . . . .	78
2.12	A nontrivial command <code>\verb</code> . . . . .	79
2.13	Expandable tokens . . . . .	83
<b>3</b>	<b>Mathematics</b>	<b>85</b>
3.1	Introduction . . . . .	85
3.2	The basic objects . . . . .	88
3.3	Parsing a math formula . . . . .	92
3.4	Translation of arrays . . . . .	96
3.5	Trivial math . . . . .	97
3.6	Conversion to XML . . . . .	99
3.7	Final math mode hacks . . . . .	103
3.8	Extensions . . . . .	104
<b>4</b>	<b>Translating a bibliography</b>	<b>107</b>
4.1	Introduction . . . . .	107
4.2	Citing a document . . . . .	110
4.3	Using <code>Tralics</code> instead of <code>BibTeX</code> . . . . .	113
4.4	The format of a name . . . . .	119
4.5	Commands for the <code>bbl</code> . . . . .	122
4.6	Other commands . . . . .	124
<b>5</b>	<b>Other commands</b>	<b>129</b>
5.1	Character encoding . . . . .	129
5.2	New encoding scheme . . . . .	132
5.3	Changing the input encoding . . . . .	133
5.4	Characters and Accents . . . . .	135
5.5	Verbatim material . . . . .	139
5.6	Case change . . . . .	143
5.7	Simple commands . . . . .	144
5.8	The <code>fp</code> package . . . . .	149
5.9	Action before translation . . . . .	158
5.10	Classes and packages . . . . .	159
5.11	Expandable commands . . . . .	166
5.12	Other expandable commands . . . . .	169
5.13	Other non-expandable commands . . . . .	180
5.14	Special commands . . . . .	192
5.15	Trees . . . . .	193
5.16	Linguistic macros . . . . .	195

5.17	Special parsing rules . . . . .	196
<b>6</b>	<b>Running Tralics</b>	<b>199</b>
6.1	Introduction . . . . .	199
6.2	The command line arguments . . . . .	199
6.3	Configuration files . . . . .	204
6.3.1	The standard configuration file . . . . .	205
6.3.2	The old configuration file . . . . .	206
6.3.3	The ra.tcf file . . . . .	209
6.3.4	The RR.tcf file . . . . .	211
6.3.5	The RR.plt file . . . . .	212
6.3.6	Sample files . . . . .	213
6.4	The action before translation . . . . .	213
6.4.1	Files and Paths . . . . .	213
6.4.2	Finding the configuration . . . . .	214
6.4.3	Old behaviour . . . . .	216
6.4.4	Preparing the translation . . . . .	217
6.5	Translating the Raweb . . . . .	218
6.6	Tracing commands . . . . .	224
6.7	Pictures and friends . . . . .	229
6.8	The title page . . . . .	234
6.9	Array and Tables . . . . .	238
6.10	Actions declared in the configuration file . . . . .	246
6.11	Trace of titlepage . . . . .	267
6.12	Extensions . . . . .	269
6.13	Bootstrap code . . . . .	279
6.14	Standard packages . . . . .	284
6.15	Images . . . . .	284
6.16	The puzzle . . . . .	285



---

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