

***EuroTEX***

Proceedings of the  
**Ninth European  
TEX Conference**

September 4–8, 1995  
Arnhem, The Netherlands

Sponsored by  
Elsevier Science



***EuroTEX***

Proceedings of the  
**Ninth European  
TEX Conference**

September 4–8, 1995  
Arnhem, The Netherlands

Edited by  
Wietse Dol

## **Organised by**

*NTG*, Dutch language oriented T<sub>E</sub>X users group  
P.O. Box 394  
1740 AJ Schagen  
The Netherlands  
email: [ntg@nic.surfnet.nl](mailto:ntg@nic.surfnet.nl)

## **Organizing committee**

Erik Frambach  
Theo Jurriens  
Ruud Koning  
Kees van der Laan  
Piet van Oostrum  
Simon Pepping  
Piet Tutelaers

## **Program committee**

Rosemary Baily  
Johannes Braams  
Michel Goossens  
Chris Rowley

## **Colophon**

These proceedings were typeset using  $\LaTeX$  running  $\LaTeX_{\epsilon}$  and  $\text{emT}_{\text{E}}\text{X}$ 's huge T<sub>E</sub>X compiler. The fonts used are the CMBRIGHT family designed by Walther Schmidt. The document was printed on a Xerox Docutech 135 600 dpi PostScript printer.

## Contents

**Graphics for T<sub>E</sub>X: a new implementation 1**

Andrey V. Astrelin

**T<sub>E</sub>X Plotter – a program for creating 2D and 3D pictures 5**

A.S. Berdnikov and S.B. Turtia

**VFCOMB – a program for design of virtual fonts 11**

A.S. Berdnikov and S.B. Turtia

**The status of Babel 17**

Johannes L. Braams

**Upages – plain T<sub>E</sub>X for professionals 27**

Stanislav Brabec

**A practical introduction to SGML 35**

Michel Goossens and Janne Saarela

**From L<sup>A</sup>T<sub>E</sub>X to HTML and back 105**

Michel Goossens and Janne Saarela

**pa<sub>al</sub>: formatting Pascal using T<sub>E</sub>X 169**

Pedro Palao Gostanza and Manuel Núñez García

**Beyond the bounds of paper and within the bounds of screens; the perfect match of T<sub>E</sub>X and Acrobat 181**

J. Hagen

**PPCHT<sub>E</sub>X: typesetting chemical formulas in T<sub>E</sub>X 197**

J. Hagen and A.F. Otten

**L<sup>A</sup>T<sub>E</sub>X, HTML and PDF, or the entry of T<sub>E</sub>X into the world of hypertext** 223

Yannis Haralambous and Sebastian Rahtz

**The release 1.2 of the Cork encoded DC fonts and the text companion symbol fonts** 239

Jörg Knappen

**A METAFONT–EPS interface** 257

Bogusław Jackowski

**Use of T<sub>E</sub>X as database with AnyT<sub>E</sub>X** 273

Kees van der Laan

**Indexing in T<sub>E</sub>X with AnyT<sub>E</sub>X** 279

Kees van der Laan

**A Russian style for Babel: problems and solutions** 289

Olga Lapko and Irina Makhovaya

**Data with δ<sub>α</sub>T<sub>E</sub>X** 295

Andries Lenstra, Steven Kliffen and Ruud Koning

**Modifying L<sup>A</sup>T<sub>E</sub>X** 309

L<sup>A</sup>T<sub>E</sub>X3 Project Team

**The proposed T<sub>E</sub>X Directory Structure** 315

Joachim Schrod

**Occam's Razor and macro management** 317

Laurent Siebenmann

**A package for Church-Slavonic typesetting** 331

Andrey Slepuhin

**The W95 environment** 339

Antonín Strejc

**MusiX<sub>T</sub>E<sub>X</sub>, even more beautiful than MusicT<sub>E</sub>X for music typesetting** 351

Daniel Taupin

**$\epsilon$ -TEX: a 100%-compatible successor to TEX**  
Following humbly in the footsteps of the Grand Wizard 359  
Philip Taylor

**Adobe™ Acrobat 2.0™**  
Beyond the bounds of paper 371  
Wiegert Tierie

**Typesetting commutative diagrams 391**  
Gabriel Valiente Feruglio

**Conversion of the Euler Metafonts into the PostScript Type1 font language 425**  
Erik-Jan Vens

**When METAFONT does it alone 431**  
Jiří Zlatuška



## Graphics for T<sub>E</sub>X: a new implementation

Andrey V. Astrelin

Moscow

Russia

astr@lcm.math.msu.su

Graphic capacity is the branch insufficiently developed in the basic T<sub>E</sub>X version. That leads to the large amount of the graphic packages for T<sub>E</sub>X. Some of them became standard (such as L<sup>A</sup>T<sub>E</sub>X PICTURE package), but they are the most weak of all. The developed packages differ in the level of graphic information representation in the .dvi file. There are 3 levels:

1. The graphic information is represented via characters from some special fonts (like `line10` and `circle10` from L<sup>A</sup>T<sub>E</sub>X). It may be done in two ways:
  - i. fixed fonts are created once and for all;
  - ii. each picture requires creating new fonts.
2. The graphic information is represented via `rule` commands (black rectangles). Example of this approach is the tables drawing (which is a form of graphics).
3. The graphic information is presented by the `\special` commands and processed by particular driver. Examples are `pc1` commands in PCTeX, line drawing and graphic files output in EmTeX and PostScript commands for the drivers generating PostScript output.

Selecting ways (1i) and (2), we have restricted opportunities for the picture creation, (the small set of the primitives requires a great number of them for the picture creation, that may result in T<sub>E</sub>X and printer memory overflow), and in (1ii) and (3) we have the compatibility problems.

For our implementation we chose the (3) level (`\special` commands usage) with the postprocessing of .dvi file.

In the low level of implementation the graphic command are represented in the form `\special{GR:commands}`, where `commands` are one or more command of the special language. Every command is represented by a character with some number arguments following it. During interpretation there are the following work variables:

“Points”: one may keep up to  $2^{15} - 1$  points inside a page and use them for the spline drawing;

“Drawings”: there are registers numbered from  $-2^{15}$  to  $2^{15} - 1$ , containing drawings. Register 0 contains the current drawing and is an implicit argument in the most commands.

There are two types of drawings: paths and areas. Initially every drawing is constructed as a path, i.e. set of splines. One may add a spline to the current path, transform a path to an area by the `draw` command, that is to draw a path with the pen `N`, or implicitly convert a set of closed paths to an area (like the METAFONT command `fill` do).

The drawings in the registers with positive indices are fixed in the page: when we use them as arguments of the command `load` or area commands (see below), they remain in the same place of the page where they were drawn, and the registers with the negative indices contain the replaceable drawings: the reference point where the `save` command was executed is the base point of drawing, and if we restore it at a different point, it moves to a corresponding vector.

#### Below in the list of graphic commands:

`e` erase drawing 0;  
`g` go to the reference point and use it as the start base point of the next spline;  
`c` use the reference point as the control point of the current spline;  
`b` use the reference point as the end base point of the current spline and as the start base point of the next one;  
`tN` assign the reference point coordinates to the point register `N`;  
`@N` execute the following commands using the point `N` as reference point;  
`sN` save the current drawing in the register `N`;  
`lN` load the current drawing from the register `N`;  
`fN,a,b,c,d` transform drawing with the matrix  $((a/N \ b/N) \ (c/N \ d/N))$ ;

#### Path commands:

`dN` draw the current path with the pen from register `N` (`N` should be negative);  
`hN,N1,N2,...` replace a path with a hatch; `N1/N,N2/N,...` are the parts of path length, that are included/not included in the hatch;  
`HN1,N2,...` replace a path with a hatch; `N1,N2,...` are the lengths (in `sp`) of path segments that are included/not included in the hatch;

#### Area commands:

`+N` create an union of the current drawing and the drawing contained in the register `N`;

- \*N create an intersection of the current drawing and the drawing contained in the register N;
- N subtract the drawing contained in the register N from the current drawing;
- \_ output the drawing 0 to .dvi file.

The .dvi file with the commands mentioned above is processed by the special program and post processor. As a result we get new .dvi file in which all the graphic `\special's` are removed and some commands `\special{em:graph $$$NNNN.pcx}` are inserted. Also some graphic files are created.

The second level of the product is a set of T<sub>E</sub>X macros that may be used to place the graphic `\special's` to the proper places of the page. They are written in the way to reduce the .dvi file (and T<sub>E</sub>X memory) size. There are also some macros for placing T<sub>E</sub>X boxes into the drawings.

The third level is the library of graphic macros for users. Now it is not fully developed and we shall not describe it.

In future we plan to use some alternative forms of the graphic output. Among them are automatic generation of .pk fonts; and generation of PostScript output. Also there are some ideas of replacing the `\special's` form of graphic information by the direct output to the text file for postprocessing, but in that way we shall lose the possibility of usage of T<sub>E</sub>X-dependent points as spline base or control points, and there will be some problems with graphic representation: if we include the graphic file at some point, we may lose top and left sides of picture and if we use characters of a font to be created, we need multiple passes of composing.



# **T<sub>E</sub>X Plotter – a program for creating 2D and 3D pictures**

A.S. Berdnikov and S.B. Turtia

Institute for Analytical Instrumentation  
Rizskii pr. 26  
198103 St.Petersburg  
Russia  
berd@ianin.spb.su, turtia@ianin.spb.su

## **Abstract**

The MS DOS program which creates 2D and 3D T<sub>E</sub>X pictures for the plots of functions of two variables  $f(x, y)$  is described. In comparison with GNUPLOT this program enables to plot the equilines (2D view) and the surface (3D view) pictures correctly and without memory overflow even for complex cases. The input is the ASCII file which contains the data points  $(X_{ij}, Y_{ij}, Z_{ij})$  of the function  $z = f(x, y)$  calculated over non-regular quadrangular mesh. The output is the ASCII file which contains the required picture in T<sub>E</sub>X format. The program has a flexible menu driven user interface and enables to create and to preview the output pictures with a variety of styles. At the time being the program supports L<sup>A</sup>T<sub>E</sub>X commands, EPIC/EEPIC macros and emT<sub>E</sub>X specials. In future the program should support T<sub>E</sub>X graphical tools like MFPI<sub>C</sub>, PiCT<sub>E</sub>X and EPS-files.

## **1 Introduction**

The development of this program was induced by the fact that the well known program GNUPLOT *cannot* draw properly the equilines for functions of two variables. For example, Figure 1 demonstrates equilines of function

$$f(x, y) = x^2 + y^2 - \cos(18x) - \cos(18y)$$

created by GNUPLOT 3.5 for the region  $x \in [-1, +1]$ ,  $y \in [-1, +1]$ . As it can be seen this picture is wrong: it does not reflect the symmetry  $x \longleftrightarrow y$  of the function. Figure 2 shows the plot of the function  $f(t) = t^2 - \cos(18t)$ , and the correct plot of equilines is shown on Figure 3.

There are also other drawbacks during the processing of 2D function with the help of GNUPLOT: “out of memory” messages, no possibility to draw something over the plot

of equilines, no possibility to mark the points over the plot and to supply it with some text, restricted set of line styles as compared with  $\text{\TeX}/\text{\LaTeX}$  possibilities, etc. Since our work is mostly related with calculation of electrostatic and magnetostatic fields, the equiline plots are the pictures which are indispensable. For this reason we decided to develop our own program which creates the output 2D and 3D plots compatible with  $\text{\TeX}$  in the way we like (and without such errors as shown on Figure 1). The result is the program  $\text{\TeX}$  Plotter, the preliminary version of which is described here.

## 2 Principal algorithms

It seems that the decision done by the designers of `GNUPLOT` to calculate and to store the whole equiline in advance results to most problems with 2D and 3D plots created by this program. This approach enables to smooth and to process the whole equiline before plotting, but simultaneously it results to memory problems and the errors like shown on Figure 1.

We selected the different approach which does not enable good smoothing of individual equilines as well as correct plotting with dotted and dashed equilines, but which causes no memory problems and conserves the symmetry of input data. Using this algorithm the function values are calculated at the nodes of non-regular quadrangular mesh, and the equilines inside each quadrangle are processed separately. Although the equilines inside different quadrangles are calculated independently, the pieces of equilines for neighbouring quadrangles are connected smoothly if the approximation of the function is smooth in the whole region.

Let us consider the quadrangular mesh which is characterized by the node coordinates  $(x_{ij}, y_{ij})$ ,  $i = 1 \dots N_x$ ,  $j = 1 \dots N_y$ . The neighbouring nodes  $(x_{ij}, y_{ij})$ ,  $(x_{i+1,j}, y_{i+1,j})$ ,  $(x_{i,j+1}, y_{i,j+1})$ ,  $(x_{i+1,j+1}, y_{i+1,j+1})$  forms the quadrangular mesh cell, and the value of the function  $z = f(x, y)$  at the node point  $(x_{ij}, y_{ij})$  is equal to  $z_{ij}$ .

The coordinates  $(x, y)$  can be considered as the pair of parametric functions  $X(p, q)$ ,  $Y(p, q)$ , which transforms the rectangular region  $(p, q)$  into curvilinear region  $(X, Y)$  so that the nodes  $(p_i, q_j)$  of the rectangular mesh are transformed into the points  $(x_{ij}, y_{ij})$  of the quadrangular mesh. Analogously the function  $z = f(x, y)$  is considered as the parametric function  $z = Z(p, q)$  which has the value  $z_{ij}$  for the point  $(p_i, q_j)$ .

The equilines are calculated separately inside each rectangular cell  $p_i \leq p \leq p_{i+1}$ ,  $q_j \leq q \leq q_{j+1}$  which results to piecewise presentation of the equilines where the different pieces are not necessarily joined together. As it was already mentioned these pieces are joined together after drawing, and the connection of equilines is continuous for continuous functions  $X(p, q)$ ,  $Y(p, q)$  and  $Z(p, q)$ , and is smooth for smooth functions  $X(p, q)$ ,  $Y(p, q)$  and  $Z(p, q)$ .

The continuous approximation can be constructed as the piecewise bilinear function, and the smooth approximation can be constructed as the piecewise bicubic function using well known numerical algorithms. It is essential that the approximation is *local*,

i.e., depends only on the function values at the mesh nodes next to the considered mesh cell.

The input data for T<sub>E</sub>X PLOTTER is the ASCII data file where each line contains the values  $X_{ij}$ ,  $Y_{ij}$  and  $Z_{ij}$  corresponding to quadrangular mesh used for plotting. In most cases this quadrangular mesh is actually the rectangular one so that  $X_{ij} \equiv X_i$  and  $Y_{ij} \equiv Y_j$ . The first lines of the input file contain the information about the size of the mesh  $N_x \times N_y$ , and, may be, the parametrization  $(p_i, q_j)$  which can be selected as the uniform by default. The symbolic expressions for plotted functions are not supported: it is assumed that the User can create small program using his favourite computer language which produces the ASCII file with the function data. Instead of it in future versions of T<sub>E</sub>X PLOTTER more attention will be paid to filtering, smoothing and fitting procedures.

### 3 Style options

The menu driven user interface enables to specify the input data file as well as all the parameters which define the style of the output plot. It is possible to vary the position and the size of the plot, the title and text captions and their position, font style and size, numbering and drawing of the axis, etc. The preview menu item enables to see the whole plot or its fragment using suitable magnification.

The output of T<sub>E</sub>X PLOTTER is the ASCII file which describes the 2D or 3D plot using T<sub>E</sub>X commands: L<sup>A</sup>T<sub>E</sub>X `\rule` commands, or E<sub>P</sub>I<sub>C</sub>/E<sub>E</sub>P<sub>I</sub>C `\drawline` macros, or e<sub>m</sub>T<sub>E</sub>X specials `\special{em:moveto}` and `\special{em:lineto}`. It is expected that in future the program will support the output which is compatible with such graphical facilities as M<sub>F</sub>P<sub>I</sub>C, P<sub>I</sub>C<sub>T</sub><sub>E</sub>X and Encapsulated PostScript files.

The smoothness of the plot is defined by the bilinear or bicubic approximation of the parametric functions  $X(p, q)$ ,  $Y(p, q)$  and  $Z(p, q)$  and by the number of subdivisions used to calculate equilines. The plot range and the equiline levels can be selected automatically by the program or can be defined by the User. The equilines corresponding to different levels can be drawn using various line thickness. Dotted and dashed lines are not supported. Although it is possible to organize plotting so that the pieces of dashed and dotted equilines are connected correctly, the present version of T<sub>E</sub>X PLOTTER does not realize it.

The preview function draws the text strings with the characters scaled approximately like the T<sub>E</sub>X text at the output plot. The preview function does not support *math* mode, the character metric information and ligature/kerning data of the T<sub>E</sub>X fonts. In spite of it the results are close to the real output especially as compared with the GNUPLOT previewing.

It is possible to draw over the equiline plot the arbitrary geometrical objects, including straight lines, circles and arcs, point markers and text strings. All such geometrical objects are listed in a separate ASCII file. The essential feature is that all these objects use just the same coordinate system as the input data, and it is very easy to overlap the

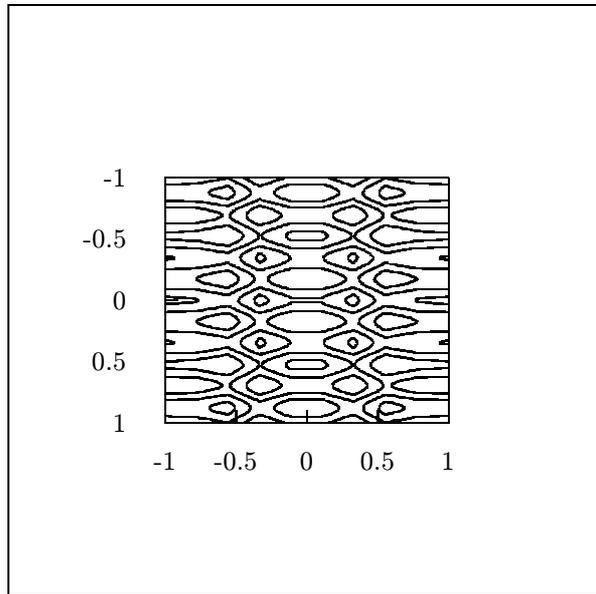


Figure 1: Equilines for the function  $f(x, y) = x^2 + y^2 - \cos(18x) - \cos(18y)$  in a region  $x \in [-1, +1]$ ,  $y \in [-1, +1]$  as calculated by GNUPLOT 3.5.

equiline plot with external markers and drawings. The output  $\text{T}_{\text{E}}\text{X}$ -file contains comments emphasizing the logical pieces of the plot so that the manual corrections of the picture can be introduced if necessary.

#### 4 Conclusion

The program  $\text{T}_{\text{E}}\text{X PLOTTER}$  is written in *Borland Pascal* using the *User's Window Tools* interface library. It works under MS DOS 3.3 and higher, does not require extended memory and can be used even on 286 computers although 386/486/586 are preferable. The program is distributed "as it is" in executable form without source code to eliminate the problems with intermediate versions and mutant source codes. It is planned to distribute the final version of the program with the source code, but the present version is far from it. The suggestions, corrections and noticed errors are welcomed.

#### 5 Acknowledgements

This research was partially supported by a grant from the Dutch Organization for Scientific Research (NWO grant No 07-30-007).

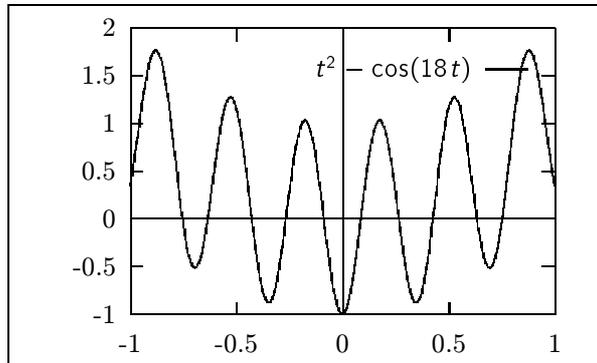


Figure 2: Function  $f(t) = t^2 - \cos(18t)$  in a region  $[-1, +1]$  (calculated by GNUPLOT 3.5).

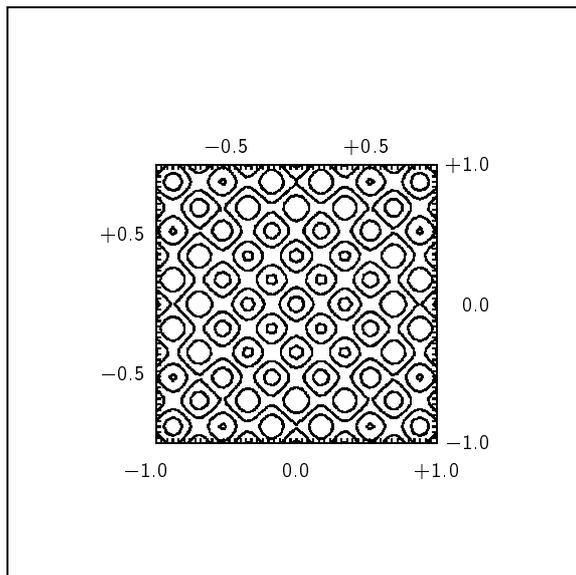


Figure 3: Equilines for the function  $f(x, y) = x^2 + y^2 - \cos(18x) - \cos(18y)$  in a region  $x \in [-1, +1], y \in [-1, +1]$  (calculated by T<sub>E</sub>X PLOTTER.)



# VFComb – a program for design of virtual fonts

A.S. Berdnikov and S.B. Turtia

Institute for Analytical Instrumentation  
Rizskii pr. 26, 198103 St.Petersburg, Russia  
berd@ianin.spb.su, turtia@ianin.spb.su

## Abstract

The MS DOS program which enables to simplify the design of the virtual fonts is described. Its main purpose was to facilitate the integration of CM-fonts with cyrillic LL-fonts created by O. Lapko and S. Strelkov but it can be used for various applications. It uses the information from TFM-files (converted to ASCII form by TFtoPL) and the ASCII data files created by the User on its input, and produces the VPL-file on its output which can be converted to the virtual font using VPtoVF. The characteristic feature of the program is that it can assemble the font ligature tables and user defined ligature tables for the characters extracted from various fonts and combine the metric information from various TFM-files. VFComb supports the full syntax of PL-files and VPL-files as it was defined by D.E. Knuth and adds new commands like symbolic variables or conditional operators, which simplifies the creation and the debugging of the virtual fonts.

## 1 Introduction

This work has been inspired by the CyrTUG project to create the new cyrillic version of  $\text{\TeX}$  format files with the cyrillic fonts which are more close to the standards used in FSU than that created at Washington University (WN-family of cyrillic fonts). According to recommendations by D.E. Knuth [1] and following the experience of the other national TUGs the mechanism of the virtual fonts<sup>1</sup> has been selected as the base tool to add new (cyrillic) characters to already existing CM-fonts.

Unfortunately the software which was at CyrTUG's disposal at this moment was not well suited for this purpose. For the first version of new fonts the VF-files have been produced by the utility TFMerge 2.4 (IHEP  $\text{\TeX}$ ware, Protvino) with subsequent manual corrections, and the TFM-files have been produced by the separate METAFONT session

---

1. It is assumed below that you are familiar with the mechanism of the virtual fonts and with the paper [1]. If it is not so, it is highly recommended to read it before you proceed further.

which actually creates the real font containing both CM and LL characters [2]. The full process was time consuming and not effective.

The program VFComb described here has been designed to facilitate the process of combining real fonts into the virtual font and to make the iterations on creation and debugging of the final font more simple and less time consuming. The main points are:

- to get the metric information and the original ligature tables from the text VPL-files converted from the binary TFM-files by TFFtoPL utility;
- to add the user defined information which describes the relation between the characters from the virtual font and the characters from the real fonts;
- to add the user defined ligature tables which describe the ligature and kerning data for the pairs of characters corresponding to different real fonts;
- to create the VPL-file which combines all these data;
- to convert the ASCII VPL-file to the binary TFM-file and VF-file which forms the virtual font data.

The most of steps are performed automatically, and data files prepared by the User have flexible syntax so that just the same input files can be used to create full generic family of fonts rather than one single font.

## 2 Input and output data

The following input data must be supplied:

- the name of the virtual font (actually the name of the output VPL-file);
- the header data which is specified at the beginning of the virtual font;
- the assignment (if necessary) the numerical values to symbolic variables used in specification of other user defined data;
- the table(s) for mapping the characters of the real fonts into the characters of the virtual font;
- the additional ligature table(s) (if present) which contain the ligature and kerning information for the pairs of characters corresponding to different real fonts;
- the metric information, ligature and kerning tables, etc., for the characters of the real fonts;
- the options which control the process of creation of the virtual font.

The output of the program is the VPL-file which contains all information necessary to create the virtual font using the utility VPtoVF and the LOG-file which summarizes all printed messages and all operations performed by VFComb.

The output result depends on the mode of operation of the program. The mode is defined by the special `OPTION` data specified in input files. The most important options are connected with the algorithms of processing of ligature and kerning tables. the following options are available:

- to discard the ligature tables of the real fonts;
- to include in the virtual font the ligature tables of the real fonts;

- to include in the virtual font only those characters which are declared explicitly in user defined data, and discard the elements of the ligature tables which correspond to non-included characters of the real font;
- to add to the virtual fonts new characters not described in user defined data if these characters are described in ligature tables for already included characters, or they are connected in one chain with already included characters using specifications `NEXTLARGER` and `VARCHAR`.

These modes can be declared both as the global modes and as the modes for the particular real font.

All the files are the ordinary ASCII files which enables to control and to correct the operations performed by VFComb manually. To get the text files which describes the real fonts it is necessary to convert the corresponding binary TFM-files to ASCII form using the utility `TFtoPL`. The other way is to prepare these text files by hand following [1]. In this case it is possible that actually there is no real font corresponding to that used in virtual font. Although this situation looks strange, it enables to use the recursive tricks described in [1] where one virtual character refers to another virtual character of the same virtual font or of the other virtual font.

### 3 Mapping table and other user defined data

The files which contain the user defined data are specified in the command line using the parameter `/t:filename` when VFComb starts. It is possible to specify several data files using several parameters `/t` – in this case the data files are read one after another. It is also possible that the data file contains the explicit command to read another data file (like `\input` in `TEX` or `METAFONT`). The data files can be placed in the current directory, or can be specified using the full path name, or can be searched implicitly in one of the directories specified for VFComb as the source for missing data.

The user defined files can contain the following information:

- (`COMMENT ...`) – comment part which is skipped.
- (`VARIABLE ...`) – defines the symbolic names with some numerical values which are used later in user data specifications.
- (`OPTION ...`) – specifies various modes of combining real fonts into the virtual font.
- (`CHARACTER ...`) – the description of the virtual font characters. These tables can refer on the character from some real font or describe the sequence of DVI-commands corresponding to virtual font character.
- (`LIGTABLE ...`) – defines the additional ligature and kerning data which is included into virtual font. These tables are prepared following the syntaxis of PL and VPL-files described in [1].
- (`DISCARD ...`) – describes the characters from the real fonts which are not included in the virtual font at any case.

- (MAPFONT ...) – defines the mode of operation with particular real font.

The following examples give the illustration of some possibilities:

- CHARACTER H OF (SELECTFONT D 0) (SETCHAR 0 37)) – the character hex(0F) of the virtual font is mapped to the character oct(037) of the real font with index 0 (the relation between the names and the indices of the real fonts is described later).
- CHARACTER 0 40 (SELECTFONT D 3) (SETCHAR C w) (DVI ...)) – the character oct(040) of the virtual font is mapped to the sequence of DVI-commands, but its size parameters are equivalent to that of the character “w” of the real font with the index 3.
- (CHARACTER D 128 (DISCARD)) – the character 128 in the virtual font is not used and moreover, VFComb cannot use this character code when it adds to the virtual font new characters not described explicitly in user defined data.
- (OPTION (INCLUDELIG)) – include in the virtual font the characters of the real fonts if these characters are encountered in the ligature table for some already included character or if they are connected in one chain with already included characters using specifications NEXTLARGER and VARCHAR.
- (DISCARD (SELECTFONT D 0) (SETCHAR D 7)) – do not include in the virtual font the character 7 from the real font 0 even when it is encountered in the ligature table of some already included character.
- (MAPFONT D 2 (NOINCLUDE)) – do not include in the virtual font the characters of the real font 2 if these characters are not described explicitly in user defined data.
- (MAPFONT D 1 (7BIT)) – include in the virtual font all the characters 0–127 of the real font with index 1 assigning them just the same codes in the virtual font if the different mapping is not described explicitly in user defined data.

Formally there is no distinction between the file with the mapping table and the file with the ligature and kerning data, and all data can be specified as one continuous stream. It is better to keep different data blocks in different files so that the same file can be used to create various virtual fonts like it is done with METAFONT source files.

## 4 Specification of the real fonts

The real fonts which are used to create the virtual font are specified in the mapping table and user defined ligature tables using symbolic numbers. The relation of the symbolic number and the real font name is performed in the command line when VFComb starts.

Each font is specified using the separate parameter */f:filename*. The first parameter */f* corresponds to *zero* real font, the second parameter */f* corresponds to the first real font, etc. The number of real fonts specified in the command line is to be not less than the number of font indices used in the mapping table.

Each specification of the font name */f* causes the program to search for the corresponding PL-file (the ASCII file converted from TFM-file using the utility TFtoPL) and to load the font metric and ligature information containing in it. The font name inserted

in virtual font specification is the name of the file minus the extension and the path information.

It is a rare chance that it is necessary to modify the original font information – in most cases it is used “as it is”. Nevertheless to get full compatibility with the syntax of VPL-files [1] the font can be followed by the parameter */a* which specifies its magnification (it corresponds to the parameter `FONTAT` in VPL-file) and by the parameter */d* which specifies the explicit search directory (it corresponds to the parameter `FONTAREA` in VPL-file). The details can be found in VFComb manual and in [1].

## 5 The header of the virtual font

The header of each  $\TeX$  font (i.e., the header of corresponding TFM-file) contains the following parameters [1]:

`HEADER`, `CHECKSUM`, `SEVENBITSAFEFLAG`, `DESIGNSIZE`,  
`DESIGNUNITS`, `CODINGScheme`, `FAMILY`, `FACE`,  
`FONTDIMEN` and its sub-parameters (`SLANT`, `SPACE`, `QUAD`, etc.).

By default all header parameters of the virtual font except the parameters `CHECKSUM`, `SEVENBITSAFEFLAG`, `HEADER` and `FONTDIMEN/PARAMETER` are copied from the header of the first real font. The User can change the index of the real font used for this purpose and/or specify new header parameters which are read from the external file using standard syntax of PL-files. The parameters `SEVENBITSAFEFLAG`, `HEADER` and `FONTDIMEN/PARAMETER` are always ignored with corresponding warning messages. The interpretation of the parameters `DESIGNSIZE` and `DESIGNUNITS` is more complex and is described in VFComb manual (fortunately in most cases these header parameters are not modified). All the other header parameters except `CHECKSUM` are copied directly to the header of the virtual font.

The parameter `CHECKSUM` is treated separately: it is ignored when read from the header of the real font, and it is inserted in the header of the output VPL-file when read from additional header file. Generally there is no reason to specify the value `CHECKSUM` explicitly because by default it is calculated automatically when converting VPL-file to VF-file and TFM-file. In most cases the `CHECKSUM` value specified by the User is wrong which causes the warning messages of DVI-drivers when the virtual font is used. The exception is the explicit specification of zero value by (`CHECKSUM D 0`) which switches off the check of the control sum value at all (the latter operation is potentially dangerous because the incorrect or damaged TFM-file of the virtual font cannot be eliminated).

## 6 Additional features

Some extensions of the syntax of PL and VPL-files are introduced to make the process of preparing the user defined data more simple, namely:

- symbolic names for numerical constants (operator `VARIABLE`);
- dynamic loading of new data files (operator `LOAD`);
- binary data "`B binconst`" and symbolic values "`V varname`" which substitute numerical constants like "`D dec-value`", "`H hex-value`", "`O oct-value`", etc., in PL-files and VPL-files and user defined data;
- pseudo-arithmetic expressions which substitute numerical constants in PL-files and VPL-files and user defined data;
- conditional operators `IF-THEN-ELSE` which enable to include or to skip some portion of the data stream depending on some conditions;
- operator `(END)` which stops the processing of the current data stream before *end-of-file* is encountered.

The following example shows how the value of the variable `CODE` defines which value is assigned to the variable `DIMCH` and what external data file is loaded:

```
(IF-EQ V CODE D 0)
    \quad (VARIABLE (REAL DIMCH R 0.31415))
    \quad (LOAD NEWTABLE.TBF)
(ELSE)
    \quad (VARIABLE (REAL DIMCH R 2.7182))
    \quad (LOAD OLDTABLE.TBF)
(ENDIF)
```

The first operator `(IF-EQ V CODE D 0)` performs the comparison of two numerical values: "`V CODE`" and "`D 0`". The first value is the value of the symbolic variable `CODE`, and the second value is the decimal integer 0. Depending on the result of comparison the real value 0.31415 or 2.7182 is assigned to the real variable `DIMCH`, and one of two files `newtable.tbf` or `oldtable.tbf` is loaded like their contents is typed at this place.

## 7 Conclusion

The program `VFCOMB` is written in Borland Pascal and is distributed "as it is" together with the source code and  $\text{\LaTeX}$  manual. The program as well as the manual is far from perfect state, and all suggestions about corrections and noticed errors are welcomed.

## 8 Acknowledgements

This research was partially supported by a grant from the Dutch Organization for Scientific Research (NWO grant No 07-30-007).

## References

- [1] D. Knuth. Virtual fonts: More fun for grand wizards. *TUGBoat*, 11.1:13–23, 1993.
- [2] O. Lapko and S. Strelkov. `MAKEFONT` as a part of `CyrTUG-emTeX` package.

# The status of Babel

Johannes L. Braams

`j.l.braams@research.ptt.nl`

## 1 Introduction

In this article I will give an overview of what has happened to babel lately. First I will briefly describe the history of babel; then I will introduce the concept of 'shorthands'. New ways of changing the 'language' have been introduced and babel can now easily be adapted for local needs. Finally I will discuss some compatibility issues.

---

[The status of babel](#)

[Overview](#)

1. Introduction
2. Shorthands
3. Switching the language
4. Adapting babel for local usage
5. Loading hyphenation patterns
6. Compatibility

---

September 1995

JLB

2

## 2 A brief history of Babel

The first ideas of developing a set of macros to support typesetting documents with  $\text{\TeX}$  in languages other than English developed around the time of the Euro $\text{\TeX}$  conference in Karlsruhe (1989). Back then I had created support for typesetting in Dutch by stealing `german.tex` (by Hubert Partl c.s.) and modifying it for Dutch conventions. This worked, but I was not completely satisfied as I hate the duplication of code. Soon after that I found that more 'copies' of `german.tex` existed to support other languages. This led

me to the idea of creating a package that combined these kind of language support packages. It would have to consist of at least two 'layers': all the code the various copies of `german.tex` had in common in one place, loaded only once by  $\TeX$ , and a set of files with the code needed to support language specific needs. During the Karlsruhe conference the name 'babel' came up in discussions I had. It seemed an appropriate name and I stucked to it.

The status of babel

A brief history of babel

- First ideas at Euro $\TeX$ '89 Karlsruhe
- First published in TUGboat 12-2
- Update article in TUGboat 14-1
- Presentation of new release at Euro $\TeX$ '95

September 1995

JLB

3

After the conference I started to work on "babel, a multilingual style-option system for use with  $\LaTeX$ ' standard document styles". The first release with support for about half a dozen languages appeared in the first half of 1990. In TUGboat volume 12 number 2 an article appeared describing babel. Soon thereafter people started contributing translations for the 'standard terms' for languages not yet present in babel. The next big update appeared in 1992, accompanied by an article in TUGboat volume 14 number 1. The main new features were that an interface was added to 'push' and 'pop' macro definitions and values of registers. Also some code was moved from language files to the core of babel. In 1994 some changes were needed to get babel to work with  $\LaTeX_{\epsilon}$ . As it turned out a lot of problems were still unsolved, amongst which the incompatibility between babel and the use of  $\tau 1$  encoded fonts was most important.

Therefore babel version 3.5 has appeared. It's main features are:

The status of babel

New features of babel 3.5

- complete rewrite of the way active characters are dealt with;
- new ways to switch the language;
- A language switch is also written to the `.aux` file;
- possibility to 'configure' the language specific files;

September 1995

JLB

3

These changes are described in the remainder of this article.

- extended syntax of `language.dat`;
- compatibility with both the `inputenc` and `fontenc` packages;
- new languages (breton, estonian, irish, scottish, lower and upper sorbian).

### 3 Shorthands and active characters

During babel's lifetime the number of languages for which one or more characters were made active has grown. Until babel release 3.4 this needed a lot of duplication of code for each extra active character. A situation with which I obviously was not very happy as I hate the duplication of code. Another problematic aspect of the way babel dealt with active characters was the way babel made it possible to still use them in the arguments of cross referencing commands. Babel did this with a trick that involves the use of `\meaning`. This resulted in the fact that the argument of `\label` was no longer expanded by  $\TeX$ .

- duplication of code
- arguments of cross referencing macros were no longer expanded

Because of these problems I set out to find a different implementation of active characters. A starting point was that a character that is made active should remain active for the rest of the document; only its definition should change. During the development of this new implementation it was suggested in discussions I had within the  $\text{L}\text{A}\text{T}\text{E}\text{X}3$  team to devise a way to have 'different kinds' of active characters. Out of this discussion came the current 'shorthands'.

#### Shorthands

A shorthand is a sequence of one or two characters that expands to arbitrary  $\text{T}\text{E}\text{X}$  code.

These shorthands are implemented in such a way that there are three levels of shorthands:

---

The status of babel Level of Shorthands

---

- user level
- language level
- system level

---

September 1995 JLB 8

Shorthands can be used for different kinds of things as can be seen from the following list:

---

The status of babel Examples of Shorthands

---

- the character ~ is redefined by babel as a one character, system level shorthand;
- In some languages shorthands such as "a are defined to be able to hyphenate the word;
- In some languages shorthands such as ! are used to insert the right amount of white space.

---

September 1995 JLB 9

When you want to use or define shorthands you should keep the following in mind:

---

The status of babel Dealing with shorthands

---

- User level takes precedence over language level;
- Language level takes precedence over system level;
- One character shorthands take precedence over two character shorthands;
- Shorthands are written unexpanded to .aux files.

---

September 1995 JLB 10

In the following table an overview is given of the various shorthand characters that are used for different languages.

---

```

~ system, catalan, estonian, galician, spanish
: breton, francais, turkish
; breton, francais
! breton, francais, turkish
? breton, francais
" catalan, danish, dutch, estonian, finnish, galician
  german, polish, portuguese, slovene, spanish
  upper sorbian
‘ catalan (optional)
’ catalan, galician, spanish (optional)
^ esperanto
= turkish

```

Note that the acute and grave characters are only used as shorthand characters when the options `activeacute` and `activegrave` are used.

- 
- `\useshorthands`
  - `\defineshorthand`
  - `\languageshorthands`

On the user level three additional commands are available to deal with shorthands, they are listed in slide 12. The command `\useshorthands` takes one argument, the character that should become a shorthand character. The command `\defineshorthands` takes two arguments, the first argument being the shorthand character sequence, the second argument being the code the shorthand should expand to.

Finally, the command `\languageshorthands` (which takes one argument, the name of a language) can be used to switch to the shorthands of another language, *not* switching other language facilities. Of course this only works if both languages were specified as an option when loading the package `babel`.

- `\initiate@active@char`
- `\bbl@activate`
- `\bbl@deactivate`
- `\declare@shorthand`

In slide 13 the low level commands to deal with shorthands are shown. The command `\initiate@active@char` is used to tell L<sup>A</sup>T<sub>E</sub>X that the character in its argument is going to be used as a shorthand character. It makes the character active, but lets it expand to its non-active self. This is changed by `\bbl@activate` and reset by `\bbl@deactivate`. The command `\declare@shorthand` is the internal command for (and also used by) `\defineshorthand`; it has *three* arguments: the first argument is the name of the language for which to define the shorthands, the other two are the same as for `\defineshorthand`.

One of the goals of the introduction of shorthands was to reduce the amount of code needed in the language definition files to support active characters. This goal has been reached; when a language needs the double quote character (") to be active all one has to put into the language definition file are code fragments such as in the following slide:

```
\initiate@active@char{"}
\addto\extrasdutch{%
  \languageshorthands{dutch}%
  \bbl@activate{"}}
...
\declare@shorthand{dutch}{" ' }{%
  \textormath{\quotedblbase{}}%
  {\mbox{\quotedblbase}}}
...
```

Apart from removing a lot of code from language definition files by introducing shorthands, some other code has been moved to `babel.def` as well. In some language

definition files it was necessary to provide access to glyphs that are not normally easily available (such as the `\quotedblbase` in the example in slide 14). Some of these glyphs have to be 'constructed' for OT1 encoding while they are present in T1 encoded fonts. Having all such (encoding dependent) code together in one place has the advantage these glyphs are the same for all the language definition files of babel; also maintenance is easier this way.

## 4 Switching the language

Until release 3.5 babel only had *one* command to switch from one language to another: `\selectlanguage`. It takes the name of the language to switch to as an argument; the command is used as a declaration and always switches everything.<sup>1</sup> Two new ways to switch to another language have now been introduced.

The status of babel

Switching the language

- `\selectlanguage`
- `\foreignlanguage`
- environment 'otherlanguage'

September 1995

JLB

15

The command `\foreignlanguage` is meant to be used when a short piece of text (such as a quote) comes from another language. This text should not be longer than *one* paragraph. For the text the hyphenation patterns and the specials are switched. The command has two arguments: the name of the language and the text from that language.

The environment `otherlanguage` switches the same aspects as `\selectlanguage` does. The difference is that the switches are local to the environment whereas one either has to use `\selectlanguage` in a group to get the same effect or one has to issue multiple `\selectlanguage` commands. Also the environment is one of the things needed to enable the development of language definition files that support right-to-left typesetting.

An aspect of some multilingual documents might be that they have section titles or figure captions in different languages. For this to work properly babel now writes some information on the `.aux` file when a language switch from either `\selectlanguage` or the `otherlanguage` environment occurs. Babel knows about the `.toc`, `.lof` and `.lot` files; if you have added an extra table of contents you should be aware of this.

<sup>1</sup> that means the hyphenation pattern, the `\lefthyphenmin` and `\righthyphenmin` parameters, the translations of the words, the date and the specials

Babel writes a language switch command to the .aux file and the standard table of contents/tables/figures files

## 5 Adapting Babel for local usage

In the past people have had to find ways to adapt babel to document classes that have been developed locally (implementing house style for instance). Some have found ways to that without changing any of the babel files, others have modified language definition files and have found themselves having to make these changes each time a new release of babel is made available.<sup>2</sup> It has been suggested to provide an easier way of doing this. Therefore I have copied the concept of configuration files from L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  and introduced language configuration files. For each language definition file that is loaded L<sup>A</sup>T<sub>E</sub>X will try to find a configuration file. Such files have the same name as the language definition file; except for their extension which has to be .cfg.

- L<sup>A</sup>T<sub>E</sub>X looks for a .cfg file for each language definition file loaded;
- the .cfg file is loaded at the *end* of the language definition file.

## 6 Loading hyphenation patterns

An important change to the core of babel is that the syntax of `language.dat` has been extended. This was suggested by Bernard Gaulle, author of the package `french`. His package supports an enhanced `language.dat` in which one can optionally indicate that a file with a list of hyphenation exceptions has to be loaded. It is also possible to have more than one name for the same hyphenation pattern register.

<sup>2</sup> which doesn't happen too often ;-).

---

```

% File      : language.dat
% Purpose  : specify which hyphenation patterns
%           to load while running iniTeX
=american
english hyphen.english exceptions.english
=USenglish
french fr8hyph.tex
english ukhyphen.tex
=UKenglish
=british
dutch  hyphen.dutch
german hyphen.german

```

As you can see in slide 15, which presents an example of a `language.dat`, an equals sign (=) on the beginning of a line now has a significant meaning. It tells  $\text{\LaTeX}$  that the name which follows the equals sign is to be an alternate name for the hyphenation patterns that were loaded last. As you probably expect, there is one exception to this rule: when the first (non-comment) line starts with an equals sign it will be an alternate name for the *next* hyphenation patterns that will be loaded. Hence, in the example the hyphenation patterns stored in the file `hyphen.english` will be known to  $\text{\TeX}$  as: 'american', 'english' and 'USenglish'. You can also see in slide 15 that for 'english' an extra file is specified. It will be loaded after `hyphen.english` and should contain some hyphenation exceptions.

## 7 Compatibility with other formats

This new release of babel has been tested with  $\text{\LaTeX}2_{\epsilon}$ ; with and without either of the packages `inputenc` or `fontenc`. There should no longer be any problems using one (or both) of these packages together with babel.

This release has also been tested with PLAIN  $\text{\TeX}$ , it should not provide any problems when used with that format. Therefore babel version 3.5 should not pose any problems when used with any format which is based on PLAIN  $\text{\TeX}$ ; this has *not* been tested by me though. Some provisions are made to make babel 3.5 work with  $\text{\LaTeX}$  2.09; but not all features may work as expected as I haven't tested this fully.

---

The status of babel

Compatibility

- tested with L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  and inputenc/fontenc
- tested with Plain T<sub>E</sub>X
- *not* tested with L<sup>A</sup>T<sub>E</sub>X 2.09

---

September 1995

JLB

16

Please note that although it was necessary to copy parts of `inputenc` and `fontenc` this does not mean that you get T1 support in PLAIN T<sub>E</sub>X, simply by adding `babel`. To achieve that much more work is needed.

# Upages – plain T<sub>E</sub>X for professionals

Stanislav Brabec

Vyšehradská 27  
Praha 2 – Nové Město (Prague)  
Česká republika  
utx@k332.feld.cvut.cz

## Abstract

When I have started my professional typography works in plain T<sub>E</sub>X, I found many things, which are done in each document. Some of them are language specific or trivial, but there exists many topics, which are strongly untrivial, and often required. T<sub>E</sub>X has many limitation, but there is (in recent time) nothing better in whole the world. Thus, we have powerful macro language but we haven't easy way to do many things: references, contents, page offsetting, interpretation of text token by token, cooperation with PostScript devices, device independent color and line drawing capabilities, easy box rotation and landscaping, making sheets and booklets, making other margins than 1 in, creating cropmarks, color signatures, color separations etc.

Certainly, there is many powerful macro systems, but they are very big, often slow, and cancels many capabilities of plain T<sub>E</sub>X. I has been particularly inspired by them, and particularly by some macros for plain T<sub>E</sub>X. But many of these macros are incompatible, if you want to use two of them, because one overwrites settings of the second.

This all is good reason to write powerful macros these things instead of creating trivial macros twice a month. Then you needn't spend much time to correct bugs caused by these trivial macros. That's why I have written my `upages.tex` macros. This chapter doesn't want to be a manual to these macros, but only an introduction with examples.

`upages.tex` macros consists on more parts. In this text I will describe the most powerful and interesting parts of them. I hope that my macros will greet many plain T<sub>E</sub>Xists. They makes easy to prepare documents and to make hooks and patches.

## 1 Programmer structures

As I noted above, many macros are incompatible. The reason is simple: two macros redefining the same macro or variable often causes malfunctions. Programmer structures

gives you very neat and safe way to write such macros. For example, if you want to change your output routine to make at time landscape and mirror, it can be done easy by macro `\redef`. These structures helps to you in creating safe macros.

$\TeX$  is a macro language. To simplify many macros, it is often useful to have some programmer structures: stack, safe way to redefine some macros, tokens etc. `upages` brings you many macros for those rules. Some of them allow you to redefine `\par`, other to add and remove some tokens in `\everypar`, other push something to stack and return it back etc.

Another topic are macros called only in a special cases. `upages` brings new way to include these macros (mechanism is particularly similar to Pascal's *forward*) when they are required. It can save much memory.

These macros are simple for use, as shows following examples.

```
% Example of upages macro for internal definition with '@'.
% Internal definitions ends by \pull.
\def\internals{\pushedthe\catcode'\@=\letter}
% Save current font, then select other and return preceding.
\pushthefont \it text \pull
% Save current color, then type in red, then restore it.
\puththecolor \Red text \pull
% Add something to macro, then remove it (nesting required).
\redef\par\oldpar{(Here ends paragraph.)\oldpar}
... \par \restoredef\par\oldpar
% Add something to \everypar, then remove it (nesting not required).
\addtoks\everypar\oldeverypar{(Here starts paragraph.)\oldeverypar}
... \par \restoretoks\everypar\oldeverypar
% Permanently add contents of \toks0 to everyjob.
\etotokse\everyjob{\the\toks0}
% Number pushing. This is only naughty example of these macros.
Next page is \#\pushthe\pageno\incr\pageno\number\pageno\pull.
% Preliminary definition of macro \hebtext#1 in file hebrew
\friend\hebtext#1{}\hebrew}
```

These macros have many variations for different objects and types of requirements. No example above is using grouping. But end of group properly returns original meanings and cancels new stack items.

## 2 New output routines and device dependent setup

Classic `\plainoutput` performs easy output of single pages with margins 1 in. Man wants to typeset more general documents with different paper sizes and margins. Also output devices are not absolutely the same and often prints with different offsets, and

paper which we are using often has other dimensions than final book size. And, finally we are preparing documents with previewer, then we prints it on our printers, and final result is printed as mirror output on transparent materials or films. All these devices have different reference point, device size etc. Certainly, in most cases you can change offsets in command line, but this requires to know values for each document. If you want to create a booklet with new size, you must take a calculator and make some preprints, until you have correct position. But this all can do  $\TeX$ . You indeed loses device independence of `dvi` file, but it isn't interesting to you when printing the document on your (oneness) printer is required. You can be happy, when you will send your file to the printer, and the document is exactly positioned without any proofs.

These macros gives you such possibilities. You only need one test to set reference point once forever for each output device or driver. You will do simple test, then set device preferences, and after setup you are ready.

Other part of these macros works with paper formats. Now you needn't any calculator to compute correct margins. You can easy set them! You needn't to remember paper formats –  $\TeX$  does it for you.

Following example shows you the mechanism of usage of these drivers.

Example of setup files:

```
% Offset driver example.
\def\@myhpljv{%
\comment={Driver prints A4 with certain HP LJ5.}%
\devheight=297mm
\devwidth=210mm
\devhoffset=24,2mm
\devvoffset=26,0mm}

% Special driver example. (particular)
\def\@dvips{%
\def\beg@rotate#1{\rotstart{#1 rotate}}%
\def\beg@trnleft{\rotstart{270 rotate}}%
\let@end@rotate\rotfinish%
\let@end@flip\rotfinish%
\def\beg@flip{\rotstart{-1 1 scale}}%
etc...

% Example of mode definitions.
% \newmode{special driver}{offset driver}{preferences}
\newmode\preview{PasTeX}{}{}
\newmode\preprint{dvips}{myhpljv}{\pageinfo\turnmarks}
\newmode\film{dvips}{Linotype}{\mirroroutput\cropmarks\cutmarks
\pageinfo\colorsamples\CMYKseparation}
```

```

% Example of paper size driver.
\def\USLetter{%
\totalwidth=8.5in
\totalheight=11in
\usedriver{USLetter}}
Following example shows, how easy is document setup:
\mode\preview % or \mode\preprint \mode\film etc.
\inmargin=14mm
\outmargin=17mm
\topmargin=15mm
\botmargin=21mm \withoutfootline
\doubleside
\booklet
\DIN A6

```

...and you will be sure to obtain what you want on all devices. No calculations, no proof prints. Described commands also rounds number of pages to be dividable by four to allow create a booklet. For such things you have there powerful mechanism of vacate pages generating.

### 3 Referencing system

References are common problem of T<sub>E</sub>X. Expansion method causes a lot of troubles with making them. Typical problem is references to pictures with picture number in `\inserts`. All simple methods can fail. You will need immediately expand number of picture, “~” mustn’t be expanded at all, and, finally `\folio` needs to be expanded in time of `\shipout`. Thus I have suggested macros expanding tokens just in described order. These reference macros are based on powerful “interpretation” macros described below. These macros gives you chance to make easy references. Also “back references” are supported.

Example of reference macros:

```

\def\chapter#1{\incr\chapno\centerline{\bf#1}%
\totoc{\number\chapno: #1 .. \folio\par}}

```

This macro can be used simply as showed:

```

\chapter{Contents}
\inserttoc % inserts table of contents
\begintoc % starts writing of contents

\chapter{Introduction}

```

If you don't want to have chapter "0: Contents" in contents, you can simply use macros `\suspendtoc` and `\restoretoc`.

But this is not the only you can do: Command `\newref` generates reference macros for any file you want. It can be a list of images, index, cross references etc.

## 4 Interpretation macros

These macros are most powerful, difficult, mysterious and dirty<sup>2</sup> macros. They are true combination of most dirty tricks in the  $\TeX$ book! They takes token by token and operates with them. In these macros you are defining "processor", "eaters", "rollers" and "terminal". This macros allow you to write simply such macros as spaced text, making small letters to small caps, special expansions (see references), superprotection macros, macros for reverse order typesetting or anything other what you want. Such typesetting is many times slower than regular  $\TeX$  typesetting, but makes possible many things. There are many types of macros: `\everytoken`, `\everyetoken`, `\everyatom`, `\everygeatom` etc. Differences between them are in interpretation of begin and end group characters and way to get tokens and expand them. Although these macros are versatile, it is not simple to write such interpreters: You must know way of `\if` conditions expansion, group counting etc. But it is a chance for you.

If you sometimes wanted to type something in hebrew, you certainly found, how difficult it is without `xet`. In following example you can see, how easy it is with interpreters. Macro for such things is trivial! Watch the following:

```
\newtoks\revtoks
\def\revorder{\revtoks={}%
\everyxeatom{\totoksb\revtoks{##1}}{\the\revtoks}}
{This is in normal order \revorder and this is in reverse order.}
```

That's all folks!

## 5 Footnotes

As noted above, `upages` has quite new output module. In this case it is easy to improve footnote style to make easy to change it (in plain  $\TeX$  it is a rule for wizards). You are only changing `\footchar`, `\footdenotator`, `\footnotestyle`, `\endfootnotestyle`, `\footstrutbox`, and fonts (`\footnotefont`, `\footidentfont` and `\footdenotfont`). Following example shows, how to define footnotes:

```
\font\footnotefont=helv at 8pt
\footidentfont=helv at 7pt \let\footdenotfont=\footidentfont
\note{This is numbered note.}
```

In complex you can change footnote baselines, styles, denotation and identifier positioning.

## 6 Miscellaneous

This part will show you other macros from `upages`. `upages` contains many macros for easy work:

- `\reparfill` improves `\parfillskip` to prevent last line in paragraph from unwanted design.
- `\raggedleft`, `\raggedright`, `\raggedcenter`, `\raggedrldiag` etc. for different paragraph shapes.
- `\farnoindent` system for cases as `\farnoindent\medskip`, which allows `\removelastskip`.
- `\,`, `\>`, `\;`, `\!` makes to be defined outside math.
- `\inxy` and `\inyx` etc. to define transposable macros with `\xbox` and `\ybox`.
- `\framebox`, `\rectangle` etc. for drawing boxes.
- `\spaced` etc. for typing spaced text (using interpreters).
- `\hatebreak` and other penalization.
- `\setfont` for rarely used fonts.
- `\gridoriented` and whole mechanism to make grid oriented typesetting and rounding easier.

## 7 Device dependent functions and PostScript

To prevent changes in documents, this system suggest a set of independent macros to work with graphics, images, color etc. There is many programs, many device dependent `\special` commands. Good interface will prevent troubles and makes  $\text{\TeX}$  source code as portable as possible. I'm not only, who suggests such interface. New  $\text{\LaTeX3}$  will have such interface, macro package `PSTricks` has it's own. I hope that in nearest time will be ready single versatile special driver for all those systems, including all required functions (or warning messages). This part of `upages` is in my recent development. Following examples will work (I hope) with `upages v2`:

```
\linestyle{1pt}{\Black\fullline}
\fillstyle{\Red}
\ellipse{1cm}{3cm}{40}
\bitmappicture{mypic}
\rotated{30}\flipped\bgcolor\hbox{\Blue{Hallo}}
\setglobal{\mirror}
```

## **8 Future**

Professional can want output with sheets containing 2, 4 or 8 pages, film saving mode canceling exposition of empty pages etc. Certainly, special commands for professional work with color would be welcome. (How perfect would be commands such as: `\screenangles`, `\CMYKseparation`, `\undercolorremoval` etc.!) But this is in recent time future. The whole purpose of this work is to get power to create books with color pictures in T<sub>E</sub>X.

My final destination is:

Typographer → T<sub>E</sub>X → dvips → exposition unit

Is there any reason to use suspicious programs, when we have T<sub>E</sub>X?



# A practical introduction to SGML

Michel Goossens and Janne Saarela

CERN, CN Division  
CH-1211 Geneva 23  
Switzerland  
goossens@cern.ch, saarela@cern.ch

## Abstract

SGML, the *Standard Generalized Markup Language*, deals with the structural markup of electronic documents. It was made an international standard by ISO in October 1986. SGML soon became very popular thanks in particular to its enthusiastic acceptance in the editing world, by large multi-national companies, governmental organizations, and, more recently, by the ubiquity of HTML, *HyperText Markup Language*, the source language of structured documents on WWW. This article discusses the basic ideas of SGML and looks at a few interesting tools. It should provide the reader with a better understanding of the latest developments in the field of electronic documents in general, and of SGML/HTML in particular.

## 1 Why SGML?

Since the late eighties we have witnessed an ever quickening transition from book publishing exclusively on paper to various forms of electronic media. This evolution is merely a reflection of the fact that the computer and electronics have made inroads into almost every facet of human activity. In a world in which one has to deal with an ever-increasing amount of data is support of the computer is a particularly welcome alternative, for the preparation of telephone directories, dictionaries, or law texts – to mention just a few examples. In such cases it is not only the volume of the data that is important, but also the need for it to be kept constantly up-to-date.

Once data have been stored in electronic form one can derive multiple products from a single source document. For instance, an address list can be turned into a directory on paper, but it can also be put on cdrom, as a data-base allowing interactive or e-mail access on the Internet or to print a series of labels. Using a set of law texts or a series of articles on history marked up in SGML, one can first publish a textbook containing complete law texts, or a historic encyclopedia, and then provide regular updates or

extract a series of articles on a given subject; one can also offer a consultation service on Internet, via gopher, www or develop a hypertext system on cdrom.

All these applications suppose that the information is not saved in a format that is only suited for printing (for example, WYSIWYG), but that its logical structure be clearly marked.

To recapitulate, the strong points of a generic markup (in SGML) are the following:

- the quality of the source document is improved;
- the document can be used more rationally, resulting in an improved life-cycle;
- the publishing costs are reduced;
- the information can be easily reused, yielding an added value to the document (printed, hypertext, data base).

### 1.1 The origins of SGML

In order to treat documents electronically it is essential that their logical structure be clearly marked. On top of that, to ensure that documents are really interchangeable, one had to develop a common language to implement this type of representation.

A big step forward was the publication by ISO (the International Standards Organization, with headquarters in Geneva, Switzerland) in October 1986 of SGML as Standard ISO8879 [15]. Because SGML had been officially endorsed by ISO, the Standard was quickly adopted by various national or international organizations and by the large software developers. One can thus be fairly confident that SGML is here to stay and that its role in electronic publishing will continue to grow.

### 1.2 Who uses SGML?

With the appearance of new techniques and needs linked to the constantly increasing importance of electronic data processing, the traditional way of exchanging documents has been drastically changed. Today, SGML has become an ubiquitous tool for document handling and text processing.

First among the application areas we will consider in which SGML is at present actively used is the work of the American Association of Publishers (AAP). The AAP (see [2] to [4]) selected three types of documents in the field of publishing: a book, a series publication, and an article. For each of these a *document type definition* (DTD, see below, especially Section 4) has been developed. Together, the AAP and the EPS (European Physical Society) have proposed a standard method for marking up scientific documents (especially tables and mathematical documents). This work forms the basis of ISO 12083.

Another application actively developed during the last few years is the CALS (*Computer-aided Acquisition and Logistic Support*) initiative of the American Department of Defense (DoD). This initiative aims at the replacement of paper documents by electronic media for the documentation of all arms systems. The DoD decided that all

documentation must be marked up in SGML, thus also making (the frequent) revisions a lot easier.

A few other examples of the use of SGML are:<sup>1</sup>

- the Publications Office of the European Communities (FORMEX);
- the Association of German editors (Börsenverein des Deutschen Buchhandels);
- the British Library with “SGML: Guidelines for editors and publishers” and “SGML: Guidelines for authors”;
- in France, the *Syndicat national de l'édition* and the *Cercle de la librairie*, two associations of French publishers, have defined an application for the French editing world [20];
- the ISO Publishing Department and the British Patents Office (HMSO);
- Oxford University Press and Virginia Polytechnic (PhD, USA);
- the Text Encoding Initiative (classic texts and comments);
- the technical documentation of many major computer manufacturers or scientific publishers, for instance the DocBook or other dedicated DTDs used by IBM, HP, OSF, O'Reilly, etc.
- many text processing and data base applications have SGML input/output modules (filters), for example, Frame, Interleaf, Microsoft, Oracle, Wordperfect;
- McGraw-Hill (Encyclopedia of Science and Technology);
- the electronics industry (Pinnacle), the aerospace industry and the airlines (Boeing, Airbus, Rolls Royce, Lufthansa, etc.), the pharmaceutical industry;
- press agencies;
- text editors and tools with direct SGML interfaces, such as Arbortext, EBT, ExotERICA, Grif, Softquad;
- and, of course, HTML and www!

## 2 SGML basic principles

SGML is a standard method of representing the information contained in a document independently of the system used for input, formatting, or output.

SGML uses the principle of logical document *markup*, and applies this principle in the form of the definition of a *generalized* markup language. SGML in itself does not define *per se* a markup language, but provides a framework to construct various kinds of markup languages, in other words SGML is a *meta-language*.

### 2.1 Different types of markup

The “text-processing” systems that have found their way into almost every PC or workstation nowadays are mostly of the WYSIWYG type, i.e., one specifically chooses the “presentation” or “formatting” characteristics of the various textual elements. They

---

1. See also the “SGML Web Page” at the URL <http://www.sil.org/sgml/sgml.html> for more information on who uses SGML and why.

can be compared to older formatting languages, where specific codes were mixed with the (printable) text of the document to control the typesetting on the micro level. For example, line and page breaks, explicit horizontal or vertical alignments or skips were frequently used to compose the various pages. Generally, these control characters were extremely application-specific, and it was difficult to treat sources marked up in one of these systems with one of the others. On the other hand, this type of markup does a very good job of defining the specific physical representation of a document, and for certain kinds of documents it might be more convenient for obtaining a given layout, in allowing a precise control of line and page breaks. This approach makes viewing and printing documents particularly easy, but re-using the source for other purposes can be difficult, even impossible.

To successfully prepare a document for use in multiple ways it is mandatory to clearly describe its logical structure by eliminating every reference to a physical representation. This is what is understood under the term *logical* or *generic* markup. The logical function of all elements of a document – title, sections, paragraphs, tables, possibly bibliographic references, or mathematical equations – as well as the structural relations between these elements, should be clearly defined.

Figure 1 shows a few examples of marking up the same text. One clearly sees the difference between *specific* markup, where precise instructions are given to the text formatter for controlling the layout (for example, the commands `\vskip` or `.sp`), and *generic* markup, where only the logical function (chapter or beginning of paragraph) is specified.

## 2.2 Generalized logical markup

The principle of logical markup consists in *marking* the structure of a document, and its definition has two different phases:

1. the definition of a set of “tags” identifying all elements of a document, and of formal “rules” expressing the relations between the elements and its structure (this is the role of the DTD);
2. entering the markup into the source of the document according to the rules laid out in the DTD.

Several document instances can belong to the same document “class”, i.e., they are described by the same DTD – in other words they have the same logical structure. As an example let us consider two source texts of an article (see Figure 2), where the specific structures look different, but the logical structure is built according to the same pattern: a title, followed by one or more sections, each one subdivided into zero or more subsections, and a bibliography at the end. We can say that the document instances belong to the *document class* “article”.

To describe the formal structure of all documents of type “article” one has to construct the *Document Type Definition* (or DTD). of the document class “article”. A DTD is expressed in a language defined by the SGML Standard and identifies all

*Specific markup*

T<sub>E</sub>X

```

\vfil\eject
\par\noindent
{\bf Chapter 2: Title of Chapter}
\par\vskip\baselineskip

```

Script

```

.pa
.bd Chapter 2: Title of Chapter
.sp

```

*Generic or logical markup*

L<sub>A</sub>T<sub>E</sub>X

```

\chapter{Title of Chapter}
\par

```

HTML (SGML)

```

<H1>Title of Chapter</H1>
<P>

```

Figure 1: Different kinds of markup

the elements that are allowed in a document belonging to the document class being defined (sections, subsections, etc). The DTD assigns a name to each such structural element, often an abbreviation conveying the function of the element in question (for example, “sec” for a section). If needed, the DTD also associates one or more descriptive *attributes* to each element, and describes the relations between elements (for example, the bibliography always comes at end of the document, while sections can, but need not contain subsections). Note that the relations between elements do not always have to be hierarchical, for instance the relation between a section title and a cross-reference to that title three sections further down is not a hierarchical type of relation. In general, DTDs use element attributes to express these kinds of cross-link.

Having defined the DTD one can then start marking up the document source itself (article A or article B), using the “short” names defined for each document element. For instance, with “sec” on form the *tag* <sec> for marking the start of a section and </sec> to mark its end, and similarly one has <ssec> and </ssec> for subsection, and so on.

Article A	Article B
=====	=====
Title	Title
Section 1	Section 1
Subsection 1.1	Subsection 1.1
Subsection 1.2	Subsection 1.2
Section 2	Subsection 1.3
Section 3	Section 2
Subsection 3.1	Subsection 2.1
Subsection 3.2	Subsection 2.2
Subsection 3.3	
Subsection 3.4	
Bibliography	Bibliography

Figure 2: Two instances of the same document class “article”

```

<article>
<tit>An introduction to SGML</tit>
<sec>SGML: the basic principles</sec>
<P> ...
<ssec>Generalized logical markup</ssec>
<P> ...

```

### 2.3 A few words about the DTD

If one wants to apply the latest powerful data processing techniques to electronic documents, using the information about their structure, one must have ways to ensure that they are marked up without mistakes. One must also ensure that the structure of a document instance is coherent: a document must obey the rules laid out for documents of the given document class, according to the DTD for that class.

To fulfil all these aims a DTD defines:

- the *name* of the elements that can be used;
- the *contents* of each element (Section 4.2);
- *how often* and in what order each element can occur (Section 4.2);
- if the begin or end tag can be *omitted* (Section 4.2);
- possible *attributes* and their default values (Section 4.3);
- the name of the *entities* that can be used (Section 4.4).

### 3 Transmitting the information relative to a document

The aim of SGML is to represent the information contained in a document. Already in Section 2.2 we have explained that SGML operates in two stages to define the structure of a document:

- a declaration phase;
- a utilization phase, where the document source is marked up using declared elements, attributes and entities.

This basic principle is used for the transmission of *all the information related to the document to be exchanged*.

The basic character set is ASCII, as defined by international Standard ISO/IEC 646. One can change the character set by changing this declaration at the beginning of the parsing of the document, when the SGML declaration associated to the DTD is read in (see Appendix B.)

A document can contain symbols or characters that cannot be entered directly on the keyboard, such as Greek letters or mathematical symbols, or even illustrations, photos, or parts of another document. This functionality is implemented through the use of entity references (see Section 4.4).

The markup system is based on a set of delimiters, special symbols, and keywords with special meaning.<sup>2</sup> For instance when “sec” identifies the element “Section”, then in the document source `<sec>` is the tag marking the beginning of a Section, with the delimiters “<” and “>” indicating, respectively, the tag start and end. Similarly, the formal structure of the document (described by the DTD) has its own language defined by the SGML Standard.

More generally, the SGML Standard does not define once and for all the structure of a document and all elements that it can contain, i.e., the delimiters and special symbols, but merely specifies the construction rules they have to follow. Also, SGML does not fix the markup language, but offers an *abstract syntax*, allowing one to construct particular syntax instances as needed. The Standard proposes an example syntax, called the *reference concrete syntax*, used throughout this article. We can thus safely state that SGML is a *meta-language*.

### 4 The structure of a DTD

To better understand how SGML works we propose to examine a real example of a modern SGML application, namely HTML level 2, which corresponds to the functionality offered by popular HTML viewing programs, such as Mosaic, Netscape or Lynx. The complete DTD of HTML2 is shown in Appendix A starting on page 76. To make it easier to identify the various parts of the DTD the lines have been numbered.

---

2. These symbols can also be redefined at the beginning of the document

Before starting to parse a DTD the SGML declaration is read in by the parser. For HTML this declaration is shown in Appendix B on page 86. It defines the character set, special characters and option settings used in the DTD and allowed in the document instance. For instance, in the area of markup minimization, the parameter `OMITTAG` (Line 66) has the value `YES`, which allows tag minimization, i.e., under certain circumstances (specified in the DTD) tags can be omitted, as explained in Section 4.2. If, on the other hand, the value is specified as `NO` then tag minimization is disallowed altogether.

The DTD defines all elements, their possible attributes and the entities associated with a given document class (HTML2 in our example).

Inside a DTD the start of a declaration is noted by the sequence “<!” and its termination by “>”. Certain sections of a DTD are identified (marked) by a keyword to ensure they are handled correctly, or to (de)activate their contents according to the value of the keyword (`IGNORE` or `INCLUDE`). The notation for the beginning, respectively the end of such a *marked section* is “<![ keyword [” and “]]>” (see Lines 37–39, and 303–305).

#### 4.1 Comments

It is always a good idea to include comment lines inside document sources or DTDs, whose presence will make them more readable and help in their future maintenance.

An SGML comment has the form:

```
<!-- text of the comment -->
```

The comment is limited by the double hyphen signs, `--`, and can span several lines, as seen, for instance in Lines 1–11 and 28–35.

#### 4.2 The elements

##### *An element declaration*

Each element belonging to the logical structure of a document must be declared. This declaration specifies the *name* of the element, as well as, between parentheses, its *content model*, i.e., which elements can or must be part of the element in question.

```
<!ELEMENT name n m (content model)>
```

For instance Lines 614 and 616 are equivalent to the declaration:<sup>3</sup>

```
<!ELEMENT HTML 0 0 (HEAD, BODY)>
```

The part between the element name “HTML” and the content model “(HEAD, BODY)” describes the minimization possibilities for the `<HTML>` tag (see “Omitting tags” below). The present declaration specifies that an HTML document contains a “HEAD” followed by a “BODY”. Line 533 and the definition of the parameter entity on Lines 548–551 specify further that the document head must contain a “TITLE” and can contain a few more elements (`ISINDEX`, `BASE`, `META`, etc).

---

3. The form used in the DTD at line 616 uses a parameter entity, see Section 4.4.

<i>symbol</i>	<i>description</i>
,	all must appear and in the order indicated (ordered “and”)
&	all must appear but any order is allowed (unordered “and”)
	one and only one can appear (exclusive “or”)
+	element must appear once or more
?	optional element (0 or one)
*	element can appear once or more

Table 1: Order and choice operators

### Omitting tags

It is possible that under certain circumstances one can infer automatically from the context that an omitted tag is present. This possibility must be declared for each element between the element's name and its content model in the form of two blank separated characters, corresponding, respectively, to the omittag characteristics of the start and end tag. There are only two possible values, namely a hyphen “-” indicating that the tag *must* be present (cannot be omitted), and an uppercase letter O “O” signifying that it may be omitted. For example, for numbered (OL) and unnumbered (UL) lists and their elements (LI) one has (from Lines 379 and 411, resp.):<sup>4</sup>

```
<!ELEMENT (OL|UL) - - (LI)+>
<!ELEMENT LI - O %flow>
```

The two blank-separated hyphens, “- -”, on the first line specify that one must *always* use the begin and end tags for the list declarations (<OL>...</OL> and <UL>...</UL>) while the “- O” on the second line indicate that the end tag for the members of a list (<LI>...) may be omitted.

### The contents model

As already mentioned, the content model uses order and choice operators (see Table 1 for a list).

We already encountered the operator of choice (|), which specifies that one of the elements can be present (but not more than one at a time). Let us now turn our attention to another example with a description list (<DL>) as declared on Line 357 as:

```
<!ELEMENT DL - - (DT*, DD?)+>
```

This indicates that for a description list the start tag <DL> and end tag </DL> must always be present, and that the list can contain one or more occurrences ((...)+) of zero or more <DT> tags (DT\*) that can be *followed* (,) by at most one <DD> tag (DD?).

An element with multiple members that can appear in any order is defined on Lines 548–553. These lines essentially stipulate that an HTML head can contain, in any order,

4. The meaning of the symbols | and + is explained in Section 4.2, see especially Table 1; the definition of the parameter entity %flow can be found on Line 313, see also Section 4.2.

a title (TITLE), zero or one <ISINDEX>, <BASE>, and <NEXTID> tags, and zero or more <META> and <LINK>:

```
<!ELEMENT HEAD 0 0 (%head.content)>
<!ENTITY % head.content
    "TITLE & ISINDEX? & BASE? &
    (%head.extra)">
<!ENTITY % head.extra
    "NEXTID? & META* & LINK*">
```

An element can contain other elements, characters, or both (in the latter case one speaks of a *mixed content*).

One can specify to the SGML parser the type of characters that can be used. The following reserved names are defined for that purpose:

PCDATA *parsed character data*.

The characters are supposed to have been treated by the parser and can thus no longer contain entity references or tags. For instance, on Line 557 an HTML title is defined as:

```
<!ELEMENT TITLE - - (#PCDATA)>
```

RCDATA *replaceable character data*.

The parser can expect to find only characters or entity references, i.e., (begin and end) tags are forbidden.

CDATA *character data*.

No further processing is needed by the SGML parser (nevertheless, the data might be processed by another program, for instance PostScript). A telephone number in a letterhead could be declared thus:

```
<!ELEMENT TEL CDATA>
```

ANY The element can contain data of type PCDATA or *any* other element defined in the DTD.

EMPTY The element has an *empty content*. It can, however, be qualified by possible attributes (see Section 4.3). An example of this is the <IMG> tag and its attributes as defined on Lines 233–240.

Certain elements can be used anywhere in the document source. In this case it is convenient to declare them as *included* in the element document. More generally, an element can be contained in the content model of another element and can be part of any of the element's constituents. In this case the syntax +(...) is used. Similarly, one can *exclude* certain elements from the element being defined by using the syntax -(...). For instance, the electronic HTML form is defined on Line 457 as follows:

```
<!ELEMENT FORM - - %body.content
    -(FORM) +(INPUT|SELECT|TEXTAREA)>
```

This states that the <FORM> element can contain everything specified by the parameter entity %body.content (Lines 430, 267, 146, and 309–311). Moreover, all these elements

<i>keyword</i>	<i>value of attribute</i>
CDATA	textual data (any characters)
ENTITY(IES)	general entity name(s)
ID	an SGML element identifier
IDREF(S)	value(s) of element identifier reference(s)
NAME(S)	SGML name(s)
NMTOKEN(S)	nominal lexical token(s)
NOTATION	notation name
NUMBER(S)	number(s)
NUTOKEN(S)	numeric lexical token(s)

Table 2: Keywords for attribute types

can contain, *at any level* the tags <INPUT>, <SELECT>, or <TEXTAREA>. On the other hand, forms are not recursive, since the <FORM> tag cannot contain itself (-(FORM)).

### 4.3 Attributes

All possible attributes of all elements in a DTD must be explicitly declared in the same DTD. For reasons of clarity and convenience, attribute declarations normally immediately follow the declaration of the element they refer to.

An attribute declaration consists of:

- the name of the element(s) that it refers to;
- the name of the attribute;
- either the *attribute type*, specified as one of the keywords shown in Table 2, or, between parentheses, the list of values the attribute can take;
- a default value (one of the possible values specified between quotes, or one of the keywords shown in Table 3).

An attribute declaration thus takes the following form:

```
<!ATTLIST element_name
    attribute_1 (values) "default"
    attribute_2 (values) "default"
    ...
>
```

For instance, the list declaration (<DL>) (Lines 357–362) defines an attribute “compact” to indicate that the members of a list should be typeset more densely.

```
<!ATTLIST DL COMPACT (COMPACT) #IMPLIED
```

This declaration specifies that the only possible value is COMPACT and that the system (the parser) will provide a default value (#IMPLIED, see Table 3).

One might also wish to specify numeric information, for instance, the <PRE> tag (Lines 317–320) has an attribute to specify the width of the line:

```
<!ATTLIST PRE WIDTH NUMBER #IMPLIED
```

<i>keyword</i>	<i>description</i>
#FIXED	The attribute has a fixed value and can take only that value.
#REQUIRED	The value is mandatory and must be specified by the use.
#CURRENT	If no value is specified, then the default value will be the the last specified value.
#CONREF	The value will be used for cross-references.
#IMPLIED	If no value is specified, the parser will assign a value.

Table 3: Keywords for attribute default values

The attribute type is an “(integer) number” (keyword: NUMBER) and if no value is specified then the parser will supply a default (#implied).

As a last example let us once more look at the element <IMG> (image) and its attributes (Lines 234–240), whose definitions correspond essentially to the following declaration:

```
<!ATTLIST IMG
  SRC    %URI;           #REQUIRED
  ALT    CDATA           #IMPLIED
  ALIGN  (top|middle|bottom) #IMPLIED
  ISMAP  (ISMAP)        #IMPLIED
  . . . .
```

The first line references the parameter entity %URI (see Lines 73–84) that defines a *Uniform Resource Identifier*. This attribute is *mandatory* (#REQUIRED). The other attributes are optional and have a system-defined default value (#IMPLIED). In the case of the alignment attribute (ALIGN) a choice of any of three values if possible.

#### 4.4 Entities

Entities can be used for the following purposes:

- The definitions of abbreviated notations to ease repetitive text strings (general entities); for example,
 

```
<!ENTITY TUG "\TeX{} Users Group">
```
- The definition of notations to input special characters, accents or symbols (general character entities). An example of character entities can be found on Lines 102–105;
 

```
<!ENTITY amp CDATA "&#38;"
  -- "&" (ampersand) -->
```

 ISO has defined several standard character entity sets, for instance, for national characters (see Appendix D), graphical symbols, mathematics, etc.
- The inclusion of external files (external entities).
- The definition of variables in a DTD (parameter entities).

It is important to note that, contrary to element and attribute names, which are case insensitive and can be specified in upper, lower, or mixed case, entity names are *case-sensitive*, and one must take care to specify them precisely as they are defined.

General entities are declared in the DTD. An entity declaration first specifies a symbolic name for the entity, followed by its contents. The latter can contain tags, entity references, etc., that will be interpreted when the entity is expanded.

To refer to an entity one makes use of an *entity reference*, which takes the form:

```
&entity_name;
```

For example, if one wants to use the entity "TUG" defined above, one should type in the document source the string of characters &TUG; and the parser replaces this by the string "T<sub>E</sub>X Users Group".

The data associated with an entity can be in another (external) file (*external* entity). This kind of entity can be used to include in the source document being parsed a table or figure (or any kind of data) that was prepared with another application. Instead of including the complete contents of the file in the declaration, one merely specifies the name of the file where the data is stored. The filename must be preceded by the keyword "SYSTEM", for example, for the unix operating system one might have a declaration of the form:

```
<!ENTITY article SYSTEM
  "/usr/goossens/tug/sgmlart.sgml">
```

Inside a DTD one frequently uses *parameter* entities that allow one to considerably increase the modularity of the definition of the various elements defined in the DTD. Simple examples are (Lines 89, 91, 175);

```
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
<!ENTITY % list " UL | OL | DIR | MENU " >
<!ENTITY % text "#PCDATA | A | IMG | BR">
```

These entities are used, for instance, on Lines 212, 267, 430.

```
<!ELEMENT ( %heading ) - - (%text;)+>
```

#### 4.5 Other DTDs

In order to get a better idea of what DTDs for more complex documents look like, we shall briefly discuss the HTML3, DocBook and ISO12083.

##### HTML3

As its name indicates, HTML3 is a successor to the present HTML Standard (also known as HTML2, and discussed in detail in the previous sections). HTML3<sup>5</sup> builds upon HTML2 and provides full backwards compatibility. *Tables* have been one of the most requested features; HTML3 proposes a rather simple table model that is suitable for rendering on a very wide range of output devices, including braille and speech synthesizers.

---

5. See URL <http://www.hpl.hp.co.uk/people/dsr/html/CoverPage.html>.

*Inline figures* are available and provide for client-side handling of hot zones whilst cleanly catering for non-graphical browsers. Text can flow around figures and full flow control for starting new elements is possible.

Mathematics support for equations and formulae in HTML3 mainly uses T<sub>E</sub>X's box paradigm. The implementation uses a simple markup scheme, that is still powerful enough to cope with over 90% of the most common cases. Filters from T<sub>E</sub>X and other word processing systems will allow one to easily convert existing sources into HTML3.

As HTML is most often used to present information on-screen, it is important to allow some positioning control for the various elements in a document. Therefore, HTML3 includes support for customized lists; fine positioning control with entities like `&emsp`; , horizontal tabs, and alignment of headers and paragraph text.

As well as this, many other often-requested features have been included, most notably a style-sheet mechanism, which counters the temptation to continually add more presentation features by giving the user almost full control over document rendering, and taking into account the user's preferences (window size, resource limitations such as availability of fonts)

The HTML3.0 Internet draft specification is being developed by the IETF (Internet Engineering Task Force) taking into account the following guidelines:

- interoperability and openness;
- simplicity and scalability;
- platform independence;
- content, not presentation markup;
- support for cascaded style sheets, non-visual media, and different ways of creating HTML.

To illustrate the use of this DTD one can look at the table and mathematics parts of the HTML3 DTD (see Appendix E) and at the markup examples and the generated output (Figures 4 and 6).

### *DocBook*

The DocBook DTD<sup>6</sup> defines structural SGML markup for computer documentation and technical books. It is supported by the Davenport Group, an association of software documentation producers established to promote the interchange and delivery of computer documentation using SGML and other relevant standards.

The primary goal in developing the DTD was to filter existing software documentation into SGML. It describes the structures the collaborators of the Davenport group and other producers and consumers of software documentation have encountered in processing large bodies of documentation. The DocBook DTD uses a book model for the documents. A book is composed of book elements such as Prefaces, Chapters, Appendices, and Glossaries. Five section levels are available and these may contain paragraphs, lists, index entries, cross references and links.

---

6. See URL <ftp://ftp.ora.com/pub/davenport/docbook/fullguide.sgm>.

```

<TABLE BORDER>
<TR> <TD>R1 C1</TD><TD>R1 C2</TD><TD>R1 C3</TD>
</TR>
<TR> <TD>R2 C1</TD><TD>R2 C2</TD><TD>R2 C3</TD>
</TR>
</TABLE>

<TABLE BORDER>
<TR> <TD ROWSPAN=2><EM>R12 C1</EM></TD>
<TD>R1 C2</TD><TD>R1 C3</TD>
</TR>
<TR> <TD>R2 C2</TD><TD>R2 C3</TD>
</TR>
<TR> <TD>R3 C1</TD><TD COLSPAN=2><EM>R3 C23</EM></TD>
</TR>
</TABLE>

<TABLE BORDER>
<TR> <TH COLSPAN=2>Head 1-2</TH>
<TH COLSPAN=2>Head 3-4</TH>
</TR>
<TR> <TH>Head 1</TH><TH>Head 2</TH>
<TH>Head 3</TH><TH>Head 4</TH>
</TR>
<TR> <TD>R3 C1</TD><TD>R3 C2</TD>
<TD>R3 C3</TD><TD>R3 C4</TD>
</TR>
<TR> <TD>R4 C1</TD><TD>R4 C2</TD>
<TD>R4 C3</TD><TD>R4 C4</TD>
</TR>
</TABLE>
<P>
<TABLE BORDER>
<TR> <TH COLSPAN=2 ROWSPAN=2></TH>
<TH COLSPAN=2>Background</TH>
</TR>
<TR> <TH>Blue</TH><TH>Yellow</TH>
</TR>
<TR> <TH ROWSPAN=2>Text</TH>
<TH>Red</TH><TD>fair</TD><TD>good</TD>
</TR>
<TR> <TH>Green</TH><TD>bad</TD><TD>good</TD>
</TR>
</TABLE>

```

Figure 3: HTML3 example of tables (source)

R1 C1	R1 C2	R1 C3	
R2 C1	R2 C2	R2 C3	
<i>R12 C1</i>	R1 C2	R1 C3	
	R2 C2	R2 C3	
R3 C1	<i>R3 C23</i>		
Head 1-2		Head 3-4	
Head 1	Head 2	Head 3	Head 4
R3 C1	R3 C2	R3 C3	R3 C4
R4 C1	R4 C2	R4 C3	R4 C4
		Background	
		Blue	Yellow
Text	Red	fair	good
	Green	bad	good

Figure 4: HTML3 example of tables (result with the Mosaic browser)

The DTD also leaves room for localizations. The user of the DTD is free to give own content models for appendixes, chapters, equations, indexes, etc.

#### *The AAP effort and ISO 12083*

The American Association of Publishers (AAP) has been working since the publication of the SGML Standard in 1985 on promoting SGML as an electronic standard for manuscript preparation. This document, developed over several years as the "AAP Standard," was later promoted to by the Electronic Publishing Special Interest Group (EPSIG) and the AAP as "the Electronic Manuscript Standard," and is now a NISO (National Information Standards Organization) publication. The AAP/EPSIG application is SGML-conforming, and provides a suggested tag set for authors and publishers. It defines the format syntax

```

<!DOCTYPE html PUBLIC
  "-//IETF//DTD HTML 3.0//EN//">
<HTML>
<TITLE>A Math Sampler</TITLE>
<BODY>
<H1>Formulae by examples</H1>
<MATH>x<SUP>I</SUP>y<SUP>J</SUP>
      z<sup align=center>K</sup>&thinsp;
  <BOX>( <LEFT>1 + u<OVER>v<RIGHT> )</BOX>
</MATH>
<P><MATH><BOX>[ <LEFT>x + y<RIGHT> ]</BOX>&thinsp;
      <BOX><LEFT>a<RIGHT></BOX>&thinsp;
      <BOX>| | <LEFT>b<RIGHT> | |</BOX></MATH>
<P><MATH>int<SUB>a</SUB><SUP>b</SUP>
      <BOX>f(x) <over>1+x</BOX>&thinsp;
sin (&thinsp;x<SUP>2</SUP>+1)&thinsp;dt</MATH>
<P><MATH>
  <box>d&sigma; <over>d&epsi; </box>
=<box>2&pi; Zr<sub>0</sub><sup>2</sup>m
  <over>&beta; <sup>2</sup>(E-m) </box>
[<box>(&gamma; -1) <sup>2</sup>
  <over>&gamma; <sup>2</sup></box>
+<box>1<over>&epsi; </box>]
</MATH>
</BODY>
</HTML>

```

Figure 5: HTML3 example of simple mathematics (source)

of the application of SGML publication of books and journals. The Standard achieves two goals. First, it establishes an agreed way to identify and tag parts of an electronic manuscript so that computers can distinguish between these parts. Second, it provides a logical way to represent special characters, symbols, and tabular material, using only the ASCII character set found on a standard keyboard.

For several years the AAP and the EPS (European Physical Society) have been working on a standard method for marking up scientific documents. Their work has been the basis for International Standard ISO 12083, the successor to the AAP/EPSIG Standard, and four DTDs have been distributed by EPSIG as the "ISO" DTDs.<sup>7</sup>

7. They can be found at the URL <http://www.sil.org/sgml/gen-apps.html#iso12083DTDs>.

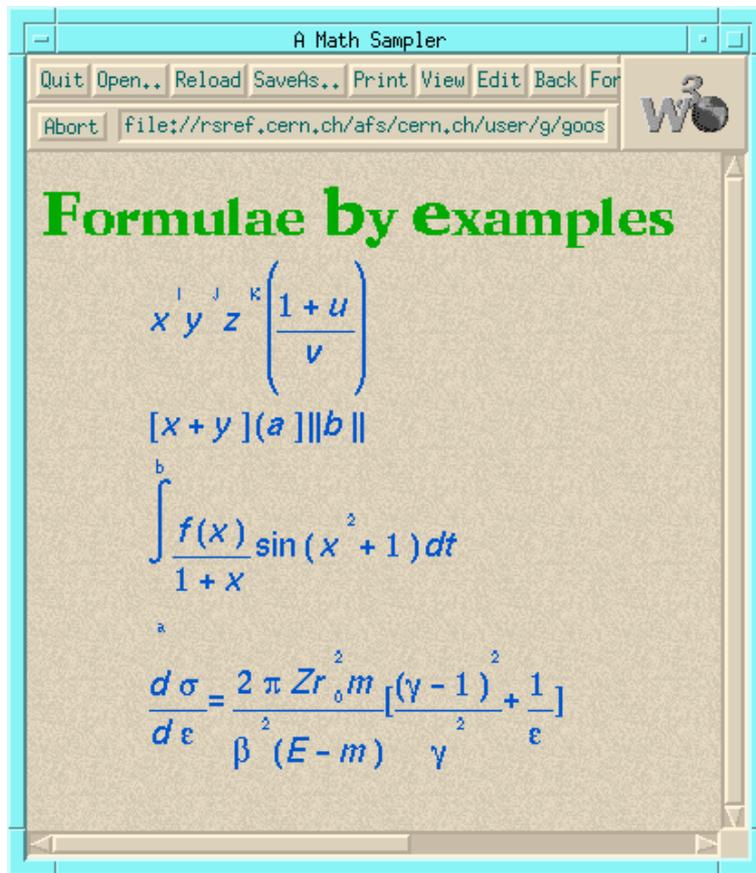


Figure 6: HTML3 example of simple mathematics (result with the arena browser)

This DTD has a basic book structure consisting of chapters, sections and subsections down to six levels. The mathematics part is, however, of some interest since it can be compared to HTML3.

#### *The ISO 12083 table model*

The ISO 12083 table model consists of the following elements (see Figure 7 for the relevant part of the DTD):

<table>	the table element;
<np>	number;
<title>	title;
<tbody>	table body;

```

<!-- ++++++ -->
<!-- Tables -->
<!-- ++++++ -->

<!ELEMENT table - - (no?, title?, tbody) -(%i.float;) >
<!ELEMENT tbody - 0 (head*, tsubhead*, row*) >
<!ELEMENT row - 0 (tstub?, cell*) >
<!ELEMENT tsubhead - 0 %m.ph; >
<!ELEMENT (tstub|cell) - 0 %m.pseq; >

```

Figure 7: Part of the ISO 12083 DTD relating to simple tables

```

<head> head;
<tsubhead> table subhead;
<row> row;
<tstub> table stub;
<cell> cell.

```

This table model does not support spanning rows or columns. It does, however, support subhead elements that can be used to give more granularity to the table contents. An example of a marked-up table is shown below.

```

<table>
  <no>1<title>Capitals in Europe
  <tbody>
    <row><cell>Helsinki<cell>Finland
    <row><cell>Rome<cell>Italy
    <row><cell>Bern<cell>Switzerland
  </tbody>
</table>

```

Only the simple table model discussed above is part of the basic ISO 12083 DTD as distributed. There also exists a complex table model [3] that allows the user to treat more complex tabular material.

#### *The ISO 12083 mathematics model*

The mathematics model in ISO 12083 consists of the following element categories:

##### **character transformations**

```
<bold>, <italic>, <sansser>, <typewrit>, <smallcap>, <roman>;
```

##### **fractions**

```
<fraction>, <num>, <den>;
```

##### **superiors, inferiors**

```
<sup>, <inf>;
```

**embellishments**

`<top>`, `<middle>`, `<bottom>`;

**fences, boxes, overlines and underlines**

`<mark>`, `<fence>`, `<post>`, `<box>`,  
`<overline>`, `<undrline>`;

**roots**

`<radical>`, `<radix>`, `<radicand>`;

**arrays**

`<array>`, `<arrayrow>`, `<arraycol>`,  
`<arraycel>`;

**spacing**

`<hspace>`, `<vspace>`, `<break>`, `<markref>`;

**formulas**

`<formula>`, `<dformula>`, `<dformgrp>`.

The model has basically the same elements as the HTML3 model, but is more visual. Emphasis is on creating fences at the right places inside a formula, whereas the HTML3 model uses `<left>` and `<right>` elements. A simple example is:

```
<formula>
  S = &sum;<inf>n=1</inf><sup>10</sup>
      <fraction>
        <num>1</num>
        <den>
          <radical>3<radix>n</radical>
        </den>
      </fraction>
</formula>
```

The complete DTD is shown in Appendix F, which shows the file `math.dtd` that is part of the ISO 12083 DTD set.

## 5 SGML editors

Several solutions exist to enter SGML or HTML markup into a document, but an editor that is SGML-aware is probably the best solution. Several (mostly commercial) products exist (see [16], [17], and [18]), but in the remaining part of this section we shall have a look at a public domain solution based on the Emacs editor with the `psgml` application and on the Grif-based Symposia editor.

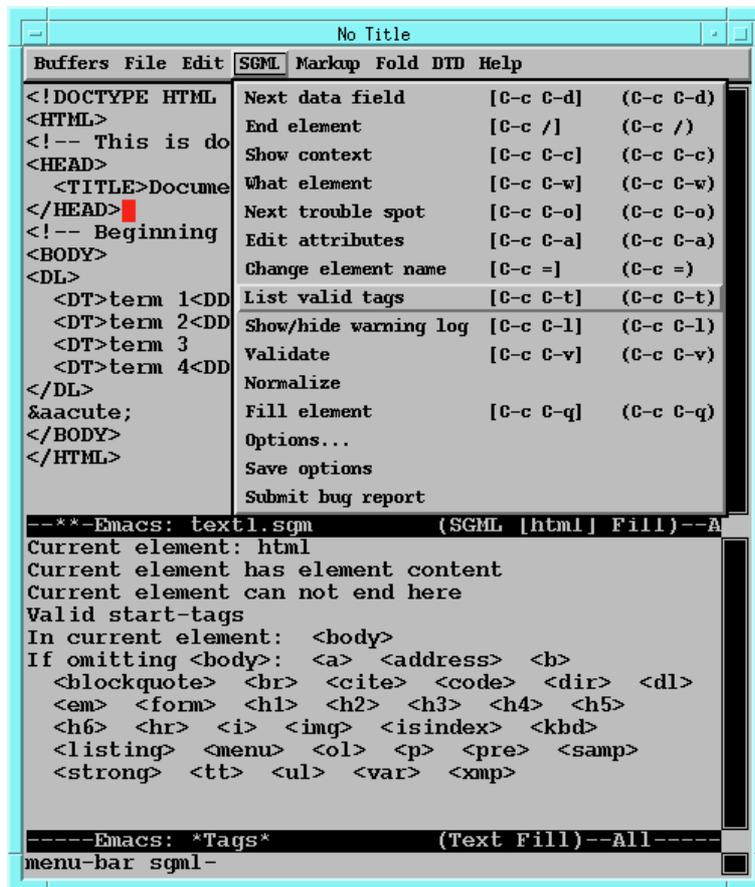


Figure 8: Emacs in psgml mode

## 5.1 Emacs and PSGML

A major mode for editing SGML documents, `psgm1`<sup>8</sup>, works with the latest versions of GNU Emacs. It includes a simple SGML parser and accepts any DTD. It offers several menus and commands for inserting tags with only the contextually valid tags, identification of structural errors, editing of attribute values in a separate window with information about types and defaults, and structure-based editing.

Figure 8 shows the first HTML test example, to be discussed later (see example `test1.html` in Section 6.2). Both the `psgm1` mode and the `nsgmls` program, discussed below, use a catalog file whose structure is defined by the SGML Open consortium to

8. The `psgm1` home page is at the URL [http://www.lysator.liu.se/projects/about\\_psgml.html](http://www.lysator.liu.se/projects/about_psgml.html).

ESC C-SPC	sgml-mark-element
ESC TAB	sgml-complete
ESC C-t	sgml-transpose-element
ESC C-h	sgml-mark-current-element
ESC C-@	sgml-mark-element
ESC C-k	sgml-kill-element
ESC C-u	sgml-backward-up-element
ESC C-d	sgml-down-element
ESC C-b	sgml-backward-element
ESC C-f	sgml-forward-element
ESC C-e	sgml-end-of-element
ESC C-a	sgml-beginning-of-element
C-c C-u	Prefix Command
C-c RET	sgml-split-element
C-c C-f	Prefix Command
C-c C-w	sgml-what-element
C-c C-v	sgml-validate
C-c C-t	sgml-list-valid-tags
C-c C-s	sgml-unfold-line
C-c C-r	sgml-tag-region
C-c C-q	sgml-fill-element
C-c C-p	sgml-parse-prolog
C-c C-o	sgml-next-trouble-spot
C-c C-n	sgml-up-element
C-c C-l	sgml-show-or-clear-log
C-c C-k	sgml-kill-markup
C-c C-e	sgml-insert-element
C-c C-d	sgml-next-data-field
C-c C-c	sgml-show-context
C-c C-a	sgml-edit-attributes
C-c =	sgml-change-element-name
C-c <	sgml-insert-tag
C-c /	sgml-insert-end-tag
C-c -	sgml-untag-element
C-c #	sgml-make-character-reference

Figure 9: Emacs key-bindings with psgml

locate the SGML declarations and DTDs (see Appendix C). Thanks to the name of the DTD declared on the `<!DOCTYPE>` declaration and that catalog file, `psgml` loads the HTML2 DTD into memory and can then handle the HTML source file. In the Figure, all the elements that can occur at the position of the pointer are shown. Figure 9 shows the more important key combinations for quickly calling some functions. For instance, the sequence `C-c C-t` (`sgml-list-valid-tags`) was used to obtain the list in the lower part of Figure 8. As a last technical (but important) detail, in order to function properly, two variables should be defined in the `psgml` initialization file `psgml.el`, namely `sgml-system-path`, a list of directories used to look for system identifiers, and `sgml-public-map`, a mapping from public identifiers to file names.<sup>9</sup>

## 5.2 Symposia

At the Third International World Wide Web Conference “Technology, Tools and Applications”<sup>10</sup>, which took place in Darmstadt, Germany, from 10 - 13 April 1995, Vincent Quint and collaborators discussed their authoring environment for SGML texts in general, and HTML on WWW in particular.<sup>11</sup> Their approach is based on the Grif editor, which can work with any DTD. They announced that a version with the HTML3 DTD will be made available freely under the name of Symposia. Grif (and Symposia) allow the user to enter text in a wysywig way, but entered elements are validated against the DTD. An example is given in Figure 10, which shows us to be in insert mode in the first column on the first row of the table, where we input the word “text”, whilst Figure 11 shows the generated SGML(HTML) source, hidden from the user, but available for any kind of treatment that one would like to do on the document.

## 6 SGML utilities

As SGML is now actively used in many applications in the field of document production (see Section 1.2 and [17]) several commercial and publicly available solutions are now available to increase the productivity, user-friendliness, and ease of using SGML systems. This section reviews a few of the more interesting publicly available tools.

### 6.1 Validating an SGML document with NSGMLS

It is often important and useful to be able to validate an SGML (and hence HTML) document. This can, for instance, be achieved with the publicly available SGML parser

---

9. See the documentation coming with `psgml` for more details.

10. An overview of the papers is at the URL <http://www.igd.fhg.de/www/www95/papers/>.

11. Their paper is available at the URL <http://www.igd.fhg.de/www/www95/papers/84/EditHTML.html>.

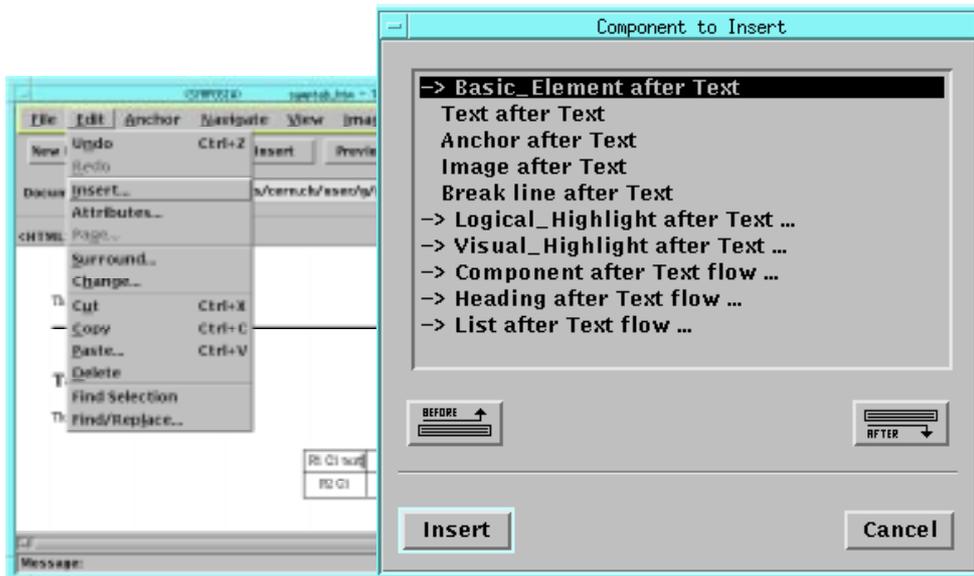


Figure 10: Inserting text in an SGML document with Symposia

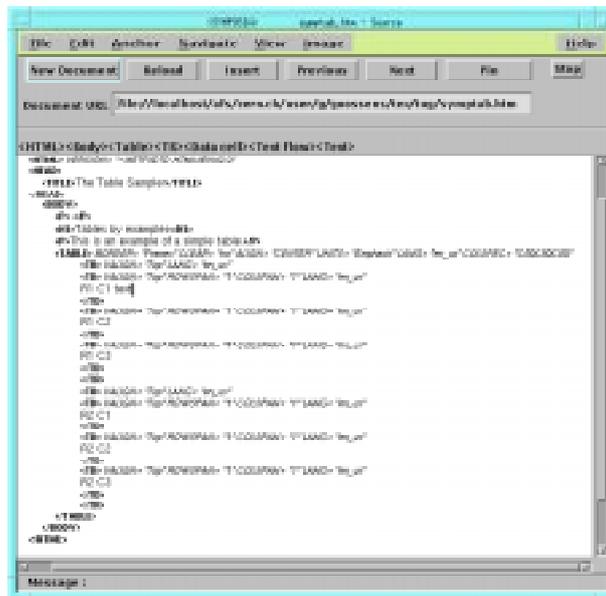


Figure 11: SGML source of the document shown in Figure 10

`nsgmls`, which is part of `sp`<sup>12</sup>, a system developed by James Clark ([jjc@jclark.com](mailto:jjc@jclark.com)), and a successor to his older `sgmls`<sup>13</sup> and `arcsgml`, written by Charles Goldfarb, who is considered by many as the father of SGML, and who is also the author of "The SGML Handbook" [5] describing the SGML Standard in great detail, a reference work that every serious SGML user should possess.

The `nsgmls` parser can be called with the syntax:

```
nsgmls [ -deglprsvx ] [ -alinktype ]
        [ -ffile ] [ -iname ] [ -mfile ]
        [ -tfile ] [ -warning_type ]
        [ filename... ]
```

`nsgmls` needs at least four files to run:

- the catalog file, which describes how the SGML file's `<!DOCTYPE>` declaration is mapped to a filename (see below);
- the SGML declaration, defining the character set used by subsequent files, and the sizes of various internal limits, such as the permitted length of identifiers, as well as what features of SGML are used, such as tag minimization (see the start of Section 4 on page 41 and Appendix B);
- the DTD for the document type;
- an SGML or HTML document instance.

## 6.2 The `<!DOCTYPE>` declaration

The `<!DOCTYPE>` declaration has three parameters, as shown in the following example.

```
<!DOCTYPE html PUBLIC
    "-//IETF//DTD HTML//EN">
```

The first parameter specifies the name of the document class according to which the document instance (the user's source file) is marked up. The second parameter is either `SYSTEM` or `PUBLIC`. With the `SYSTEM` keyword the next parameter contains the filename of the DTD, but since actual filenames are system-dependent, this syntax should be discouraged in favour of the `PUBLIC` keyword. In this case, the whereabouts of the DTD are defined via an external entity reference. The SGML Standard does not itself define how the mapping between this entity reference and an external file is defined, but SGML

12. `sp` is available at the URL <http://www.jclark.com/sp.html>. For more information about other publicly available SGML software, have a look at the the public SGML software list at the URL <http://www.sil.org/sgml/publicSW.html>. More generally, on the SGML Web Page at <http://www.sil.org/sgml/> one finds entry points to all the above, plus many examples of DTDs, more information about SGML, Hytime, DSSSL, etc.

13. `sgmls` is written in highly portable C code, whilst `nsgmls` is C++ with extensive template use, which limits the portability and makes the installation of the latter somewhat more complicated. Also the executable module of `sgmls` is about half the size of the one of `nsgmls`. See the comments of Nelson Beebe at the URL <http://www.math.utah.edu/~beebe/sp-notes.html> for the current situation with implementing `nsgmls` on several architectures.

Open has proposed the format of a catalog file in which those mappings are specified. A few examples are shown below.

```
PUBLIC "-//IETF//DTD HTML//EN"
    /usr/goossens/sgml/dtds/html.dtd
PUBLIC "ISO 12083:1994//DTD Math//EN"
    /usr/joe/dtds/math.dtd
PUBLIC "-//IETF//ENTITIES Latin 1//EN"
    /use/joe/sgml/dtds/iso-lat1.sgm
```

The first string following the keyword PUBLIC is called a “public identifier”, a name which is intended to be meaningful across systems and different user environments. Formally a public identifier is composed of several fields, separated by a double solidus, “//”. The first part is an “owner identifier” (the first and third entries have a hyphen, -, meaning that these identifiers were not formally registered, and the organization who created the file was the IETF (the Internet Engineering Task Force); the second entry carries an ISO owner identifier. The second part of the public identifier (following the double solidus), is called the “text identifier”. The first word indicates the “public text class” (for example, DTD and ENTITIES), and is followed by the “public text description” (HTML, Latin 1, etc.), then, optionally, after another double solidus one finds the “public text language”, a code from ISO Standard 639 ([9] – EN, for English in our case), and this can be followed by a “display version”, if needed.

The final element is the filename associated with the public identifier specified in the second field.

### HTML examples

It is not our intention to describe the various options of this program in detail, but we shall limit ourselves to showing, with the help of a few simple examples, how this interesting tool can be used.

```
<!DOCTYPE html PUBLIC
    "-//IETF//DTD HTML 2.0//EN">
<HTML>
<!-- This is document test1.html -->
<HEAD>
    <TITLE>Document test1.html</TITLE>
</HEAD>
<!-- Beginning of body of document -->
<BODY>
<DL>
    <DT>term 1<DD>data 1
    <DT>term 2<DD>data 2
    <DT>term 3
    <DT>term 4<DD>data 4<DD>data 4 bis
```

```

</DL>
&acute;
</BODY>
</HTML>

```

Presenting this document to `nsgmls` one obtains the following output in the “Element Structure Information Set” (ESIS) format.

```

> nsgmls -m catalog sgml.decl test1.html
#SDA
AVERSION CDATA -//IETF//DTD HTML 2.0//EN
ASDAFORM CDATA Book
(HTML
(HEAD
ASDAFORM CDATA Ti
(TITLE
-Document test1.html
)TITLE
)HEAD
(BODY
ACOMPACT IMPLIED
ASDAFORM CDATA List
ASDAPREF CDATA Definition List:
(DL
ASDAFORM CDATA Term
(DT
-term 1
)DT
ASDAFORM CDATA LItem
(DD
-data 1\n
)DD
ASDAFORM CDATA Term
(DT
-term 2
)DT
ASDAFORM CDATA LItem
(DD
-data 2\n
)DD
ASDAFORM CDATA Term
(DT

```

```

-term 3\n
)DT
ASDAFORM CDATA Term
(DT
-term 4
)DT
ASDAFORM CDATA LItem
(DD
-data 4
)DD
ASDAFORM CDATA LItem
(DD
-data 4 bis
)DD
)DL
-\n\| [aacute]\|
)BODY
)HTML
C

```

As it should, `nsgmls` parses this program without problems, and shows the different elements it encounters in ESIS format. The meaning of the most common output commands generated by `nsgmls` is as follows.

```

\\      a \;
\n      a record end;
\|      brackets internal SDATA entities;
\nnn    character whose octal code is nnn;
(gi     start of element whose generic identifier is gi, attributes for this element are
        specified with A commands;
)gi     end of element whose generic identifier is gi;
-data   data;
&name   reference to external data entity name;
Aname va1 next element has an attribute name with specifier and value va1 (see Tables
        2 and 3)
#text   application information (can only occur once);
C       signals that the document was a conforming document. It will always be the
        last command output.

```

For incorrect documents `nsgmls` shows an error:

```

<!DOCTYPE html PUBLIC
    "-//IETF//DTD HTML//EN">

```

```

<HTML>
<BODY>
  <P>text inside a paragraph
</BODY>
</HTML>

```

If we present this document to `nsgmls` (placing the HTML DTD shown in the appendix at the beginning of the file) one obtains:

```

> nsgmls -m catalog sgml.decl test2.html
test2.html:4:6:E: \
      element 'BODY' not allowed here
test2.html:7:7:E: \
      end tag for 'HTML' which is not finished
#SDA
AVERSION CDATA "-//IETF//DTD HTML 2.0//EN
ASDAFORM CDATA Book
(HTML
(BODY
-
ASDAFORM CDATA Para
(P
-text inside a paragraph
)P
)BODY
)HTML

```

Note that `nsgmls` indicates at the fourth line that a `<BODY>` tag cannot be used at that particular point (since no mandatory `<HEAD>` element – Line 614 of DTD – was specified). Then, after reading the last (seventh) line containing the `</HTML>` tag, `nsgmls` complains that the HTML document (enclosed inside `<HTML>` tags) is not yet finished.

```

<!DOCTYPE html PUBLIC
  "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<TITLE>title</TITLE>
</HEAD>
<BODY>
<LI>
</BODY>
</HTML>

```

Those only interested in checking the syntax of a document can run `nsgmls` with the `-s` option, so that it will only print the error messages, as with the incorrect HTML file above.

```
> nsgmls -s -m catalog sgml.decl test3.html
test3.html:8:4:E: \
    element 'LI' not allowed here
```

`nsgmls` does not complain until Line 8, where an isolated list member `<LI>` is found. As this is not correct according to the DTD, `nsgmls` signals its disagreement by stating that the `<LI>` tag is not allowed at that point (Lines 379 and 394 of the DTD state that list member elements of type `<LI>` can only be used in lists of type `<OL>`, `<UL>`, `<MENU>`, and `<DIR>`).

### 6.3 Prettyprinting

Nelson Beebe ([beebe@math.utah.edu](mailto:beebe@math.utah.edu)) has developed a program `htmlpty`<sup>14</sup>, written in the lex and C languages, to prettyprint HTML files. Its calling sequence is:

```
htmlpty [-options] [file(s)]
```

where the more interesting options are:

```
-f filename  name output file in comment banner;
-h           display usage summary;
-i nnn      set indentation to nnn spaces per level;
-n          no comment banner;
-w nnn      set output line width to nnn.
```

The program was run on file `test1.html` with the result shown below.

```
> html-pretty -i2 -n test1.html
<!DOCTYPE html PUBLIC
    "-//IETF//DTD HTML//EN">
<HTML>
  <!-- This is document doc1.sgm -->
  <HEAD>
    <TITLE>
      Document test HTML
    </TITLE>
  </HEAD>
  <!-- Beginning of body of document -->
  <BODY>
    <DL>
      <DT>
```

---

14. It is at URL <ftp://ftp.math.utah.edu/pub/misc/htmlpty-x.yy.trz> (choose the latest version `x.yz` offered).

```

        term 1
    </DT>
    <DD>
        data 1
    </DD>
    <DT>
        term 2
    </DT>
    <DD>
        data 2
    </DD>
    <DT>
        term 3
    </DT>
    <DT>
        term 4
    </DT>
    <DD>
        data 4
    </DD>
    <DD>
        data 4 bis
    </DD>
</DL>
&aacute;
</BODY>
</HTML>

```

The program `html-pretty` applies heuristics to detect, and often correct, common HTML errors. It can turn a pure ASCII file into a syntactically-valid HTML file that may then only require a small amount of additional markup to indicate required line breaks.

#### 6.4 SGML document analysis tools

Earl Hook ([ehood@convex.com](mailto:ehood@convex.com)) has developed a set of tools `perlSGML`<sup>15</sup>, based on the `perl` language. They permit the analysis of SGML documents or DTDs.

<code>dtd2html</code>	produces an HTML document starting from an SGML DTD that permits an easy hypertext navigation through the given DTD;
<code>dtddiff</code>	compares two DTDs and shows possible differences;
<code>dtdtree</code>	shows visually the hierarchical tree structure characterizing the relations between the various elements of a DTD;

---

15. This system can be found at the url <ftp://ftp.uci.edu/pub/dtd2html>.

`stripsgml` strips a text from its SGML markup, and attempts to translate entity references by standard ASCII characters.

Let us first look at the `dtddtree` utility. When treating the HTML2 DTD, one obtains a visual representation that is very useful for understanding the relations that exist between the various HTML elements. For each element one explicitly sees the elements it can contain. Three points “...” indicate that the contents of the element has been shown previously. Lines containing entries between brackets signal a list of elements that can be included in – (I) and (Ia) – or are excluded from – (X) and (Xa) – the content model of the element. Figure 12 shows in four columns the (condensed) output generated by the `dtddtree` program when treating the HTML2 DTD. For more clarity most of the repeated blocks have been eliminated and replaced by the string `*|**|**|` at the beginning of a line and a few lines have been cut to make them fit (marked with `***` at the end of the line).

#### *Documenting a DTD*

To document a DTD (and hence a particular SGML language instance) one can use the `dtd2html` utility, which generates, starting from the DTD in question and a file describing all document elements, a hypertext representation (in HTML) of all SGML language elements present in the DTD. This representation makes it easier for users of an SGML-based documentation system to obtain the information relating to an element they need for marking up their document. For example, in the case of HTML2, Figure 13 shows the representation as viewed by the HTML browser `mosaic`.

### **6.5 Searching and index entries**

A search engine for regular expressions for use with the HTML2 DTD is available<sup>16</sup> (Figure 14), as well as an index with more than 1100 entries and phrases<sup>17</sup> (Figure 15).

#### *Checking an HTML document*

For those who do not have `sgmls` or `nsgmls` installed there exists a set of programs `htmlchek`<sup>18</sup>, including heuristic checkers for common style and grammar violations. The programs are available in both `perl` and `awk` versions and syntactically check HTML2 and HTML3 files for a number of possible errors; they can perform local link cross-reference verification, and generate a rudimentary reference-dependency map.

`htmlchek` checks an HTML file for errors, and giving warnings about possible problems;

---

16. <http://hopf.math.nwu.edu/html2.0/dosearch.html>.

17. <http://hopf.math.nwu.edu/html2.0/docindex.html>.

18. The documentation is at the URL <http://uts.cc.utexas.edu/~churchh/htmlchek.html> and the tar file at <ftp://ftp.cs.buffalo.edu/pub/htmlchek/>.



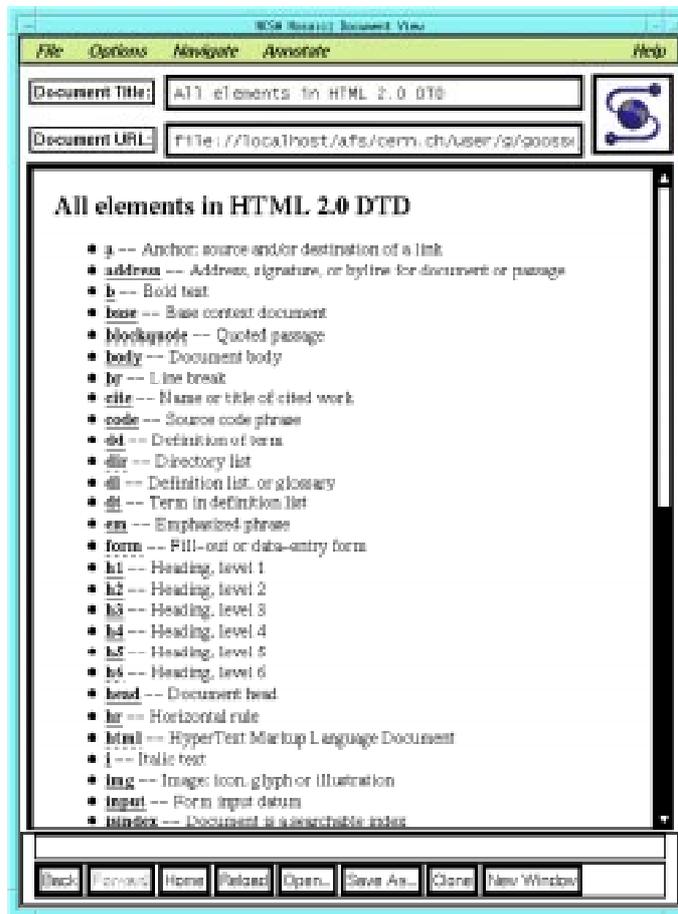


Figure 13: Hypertext description of the elements of a DTD (HTML2) as presented by the HTML browser mosaic

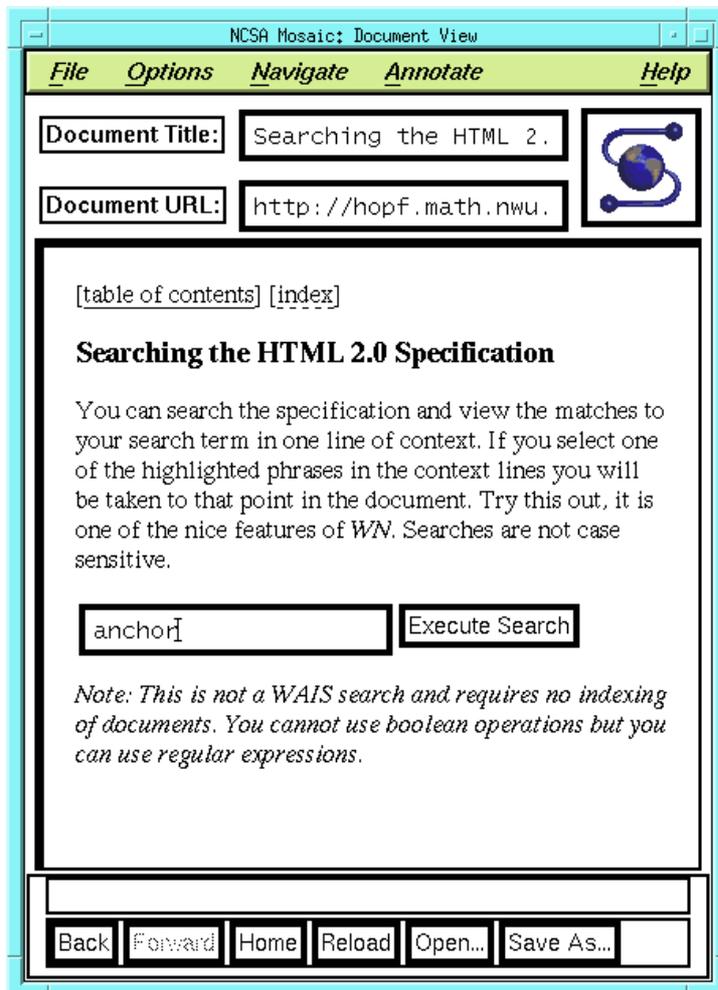


Figure 14: Searching the HTML2 DTD

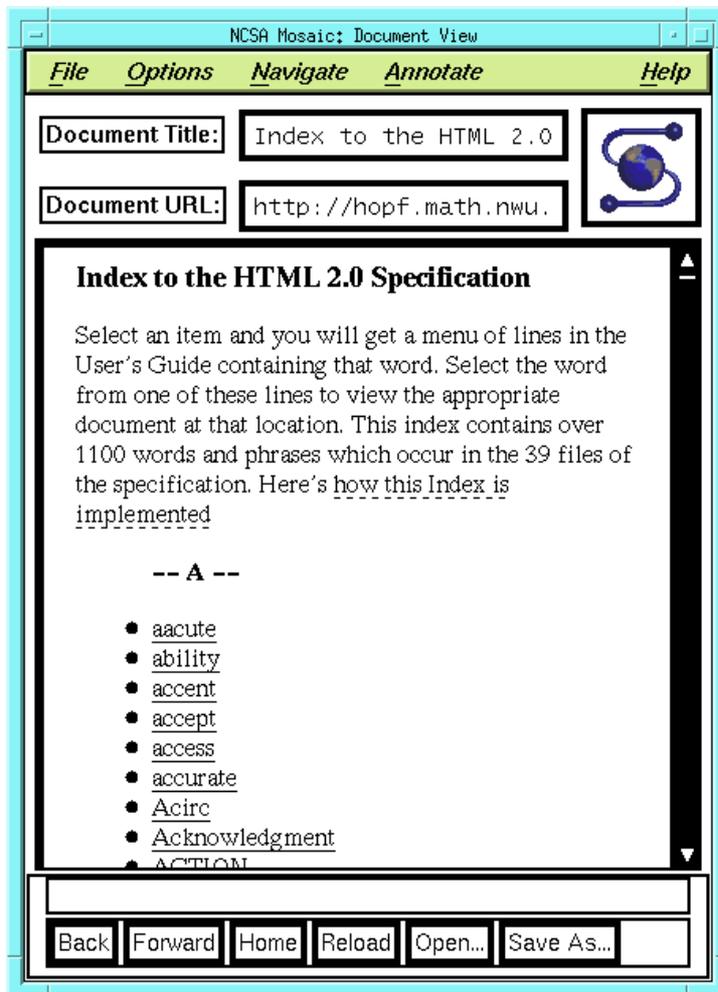


Figure 15: Index entries for the HTML2 DTD

`makemenu` makes a simple menu for HTML files, based on each file's `<TITLE>` tag; it can also make a simple table of contents based on the `<H1>`–`<H6>` heading tags;

`xtraclnk.pl` perl procedure to extract links and anchors from HTML files and to isolate text contained inside the `<A>` and `<TITLE>` elements;

`dehtml` removes all HTML markup from a document; is useful for spell checking;

`entify` replaces 8-bit Latin-1 input by the corresponding 7-bit-safe entity references;

The syntax to use these programs is typically:

```
awk -f htmlchek.awk [opts] infile > outfile
```

```
perl htmlchek.pl [opts] infile > outfile
```

As an example we ran these scripts on the test files of section 6.2 with the results shown below, which are consistent with those obtained previously.

```
> perl dehtml.pl test1.html
Document test HTML
term 1data 1
term 2data 2
term 3
term 4data 4data 4 bis

> awk -f htmlchek.awk test2.html
Diagnostics for file "test2.html":
<body> without preceding <head>...</head>
Warning! at line 4 of file "test2.html"
No <H1> in <body>...</body>
Warning! at line 6 of file "test2.html"
<HEAD> not used in document
Warning! at END of file "test2.html"
<TITLE> not used in document
ERROR! at END of file "test2.html"
Tag P occurred
Tag HTML occurred
Tag BODY occurred
Tag !DOCTYPE occurred

> perl htmlchek.pl test3.html
Diagnostics for file "test3.html":
<LI> outside of list
```

```
ERROR! at line 8 of file "test3.html"
No <H1> in <body>...</body>
Warning! at line 9 of file "test3.html"
Tag !DOCTYPE occurred
Tag BODY occurred
Tag HEAD occurred
Tag HTML occurred
Tag LI occurred
Tag TITLE occurred
```

## 7 DTD transformations

The logical markup of SGML documents makes it possible to transform the markup associated to a DTD into that of another. When translating the markup one has to take into consideration the fact that between some elements a one-to-one mapping may not exist, but that a many-to-one, and one-to-many correspondence has to be considered. It should also be noted that the tools used for this purpose need to be sophisticated, since a normal grammar tool, such as `yacc`, is not suitable for parsing SGML documents.

### 7.1 SGMLS.PL

A translator skeleton, `sgmls.pl`, is included with the `nsgmls` distribution. This `perl` script reads the ESIS output of `nsgmls` and provides a set of routines that can be used for calling user-specified translation routines of each element.

### 7.2 SGMLS.PM and SGMLSPL

David Megginson (University of Ottawa, Canada, `dmeggins@aix1.uottawa.ca`) has developed a more object-oriented approach for the translations, also based on the ESIS output of `nsgmls` and calling event-routines for each element found in the input stream. This package includes a default configuration for translating documents marked up according to the DocBook DTD into HTML or  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  markup.

The `sp` parser provides an application level interface to SGML document handling. The core of `sp` uses C++ and provides a solid class library for parsing SGML documents. The parsing of an SGML document causes events and the user can write handlers to translate them in the appropriate way.

### 7.3 Conversion from DocBook to HTML3

The translation program generates events for each primitive in the source document and these events are handled by calling a corresponding routine. These routines then produce the corresponding HTML/ $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  output. Thanks to its object-oriented flavour

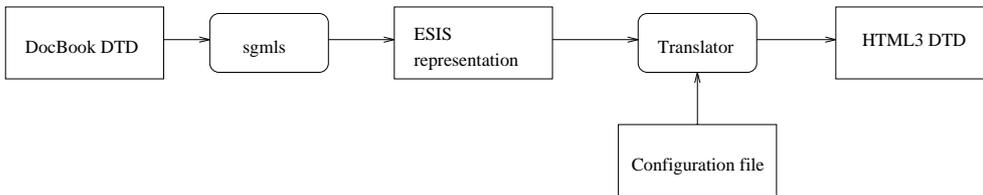


Figure 16: Schematic overview of the DocBook to HTML conversion process

the overall architecture provides solid ground for DTD translations. The following listing gives an idea of how the conversion is implemented. In the example below two elements are translated into  $\text{\LaTeX}$ . When a tag is found that can be translated, the corresponding string is produced.

```

## Program listings appear in verbatim

sgml('<PROGRAMLISTING>',
      "\n\\begin{verbatim}\n");
sgml('</PROGRAMLISTING>',
      "\n\\end{verbatim}}\n");

## Class names appear in typewriter.

sgml('<CLASSNAME>', "{\ttfamily ");
sgml('</CLASSNAME>', "}");

```

This example is extremely simple since the mappings are basically one-to-one. In the more general case, when a document element can be used inside different elements, the substitution is not just a string, but a procedure call, which allows, for instance, backtracking to cope with context-dependent conversion rules that take into account the current context. For instance, the code below shows how, when reaching the  $\text{\<TITLE>}$  end tag, the title information is handled differently, according to whether it occurred inside an article header, section or table element.

```

sgml('<TITLE>',
      sub { push_output 'string'; });

sgml('</TITLE>', sub {
  my $element = shift;
  my $data = pop_output;
  if ($element->in(ARTHEADER)) {
    $title = $data;
  } elsif ($element->in(SECT1) ||

```

```

        $element->in(IMPORTANT)) {
        output "\n\n\\section{$data}\n";
        output "\\label{$id}\n" if $id;
        output "\n";
    } elsif ($element->in(TABLE)) {
        output "\\caption{$data}\n";
        output "\\label{$id}\n" if $id;
        output "\n";
    } else {
        die "No TITLE allowed in "
        . $element->parent->name . "\n";
    }
});

```

A conversion example of an extract from the DocBook DTD manual is given in Appendix G. It shows part of the original DocBook document markup, how it is presented in the ESIS format, finally its translation in HTML3. Figure 16 shows the principle of the translation process.

#### 7.4 Commercial solutions

Several companies provide commercial solutions for doing do the translations: Exoterica, AIS, EBT (Electronic Book Technologies) and Avalanche to mention few.

## 8 Other standards in the area of electronic documents

SGML is part of a vast project conceived by the International Standards Organization (ISO) to develop a model to describe the complete process of creating, exchanging, editing and viewing or printing of electronic documents. This model consists of several standards, some already adopted, others still under discussion (see [7] and [8]).

### SGML (Standard Generalized Markup Language)

ISO 8879, the Standard described in this article is concerned with the creation and editing of documents. A complementary standard is ISO 9069 [10], SDIF, for "SGML Document Interchange Format". ISO/IEC 10744, the Hytime Standard, presents a formalism for the representation of hypermedia documents. The Hytime language ([6], [13]) allows the descriptions of situations that are time dependent (for example CD-I).

### DSSSL (Document Style Semantics and Specification Language)

International Standard ISO 10179 [14], was adopted at the beginning of 1995. It presents a framework to express the concepts and actions necessary for transforming a structurally marked up document into its final physical form. Although this Standard is primarily

targeted at document handling, it can also define other layouts, such as those needed for use with databases.<sup>19</sup>

### SPDL (Standard Page Description Language)

Draft International Standard ISO DIS 10180 [11] defines a formalism for the description of documents in their final, completely typeset, unrevisable form.<sup>20</sup> The structure of the language and its syntax strongly resemble the PostScript language, which is not surprising since PostScript has become the *de facto* standard page description language.

### Fonts

To exchange documents one must also define a font standard. ISO 9541 [12] describes a method for naming and grouping glyphs or glyph collections independently of a particular font language (such as PostScript or Truetype).

### Acknowledgments

We sincerely thank Nelson Beebe (Utah University, [beebe@math.utah.edu](mailto:beebe@math.utah.edu)) for several interesting e-mail discussions and for his detailed reading of the compuscript. His suggestions and hints have without doubt substantially improved the quality of the text. We also want to acknowledge the help of Steven Kennedy (CERN) who proofread the article.

### References

- [1] Association of American Publishers, Electronic Manuscript Series. *Author's Guide to Electronic Manuscript Preparation and Markup (Version 2)*. Association of American Publishers, EPSIG, Dublin, OH, USA, 1989.
- [2] Association of American Publishers, Electronic Manuscript Series. *Markup of mathematical formulas (Version 2)*. Association of American Publishers, EPSIG, Dublin, OH, USA, 1989.
- [3] Association of American Publishers, Electronic Manuscript Series. *Markup of tabular material (Version 2)*. Association of American Publishers, EPSIG, Dublin, OH, USA, 1989.
- [4] Association of American Publishers, Electronic Manuscript Series. *Reference Manual on Electronic Manuscript Preparation and Markup (Version 2)*. Association of American Publishers, EPSIG, Dublin, OH, USA, 1989.
- [5] C.F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.

---

19. More on DSSSL by James Clark is available at the URL <http://www.jclark.com/dsssl/>.

20. More on SPDL can be found at the URL <http://www.st.rim.or.jp/~uda/spdl/spdl.html>.

- [6] C.F. Goldfarb. Hytime: A standard for structured hypermedia interchange. *IEEE Computer*, pages 81–84, August 1991.
- [7] M. Goossens and E. van Herwijnen. Introduction sgml, dsssl et spdl. *Cahiers GUTenberg*, 12:37–56, December 1991.
- [8] M. Goossens and E. van Herwijnen. Scientific text processing. *Journal of Modern Physics C*, 3(3):479–546, June 1992.
- [9] International Organization for Standardization. *Code for the presentation of names of languages*. ISO 639:1988 (E/F), ISO Geneva, 1988.
- [10] International Organization for Standardization. *Information processing – SGML support facilities – SGML Document Interchange Format (SDIF)*. ISO 9069:1988, ISO Geneva, 1988.
- [11] International Organization for Standardization. *Information Technology – Text Communication – Standard Page Description Language (SPDL)*. ISO/IEC DIS 10180, ISO Geneva, 1991.
- [12] International Organization for Standardization. *Information Technology – Font information interchange (three parts)*. ISO/IEC 9541-1,2,3, ISO Geneva, 1991 and 1993.
- [13] International Organization for Standardization. *Information Technology – Hypermedia/Time-based Structuring Language (Hytime)*. ISO/IEC 10744:1992, ISO Geneva, 1992.
- [14] International Organization for Standardization. *Information processing – Text and office systems – Document Style Semantics and Specification Language (DSSSL)*. ISO/IEC DIS 10179.2, ISO Geneva, 1994.
- [15] International Organization or Standardization. *Information processing – Text and office systems – Standard Generalized Markup Language (SGML)*. ISO 8879:1986(E), ISO Geneva, 1986.
- [16] J. Karney. SGML and tag masters. *PC Magazine*, 14(3):144–162, 1995.
- [17] J. Karney. SGML: It's still à la carte. *PC Magazine*, 14(3):168–171, 1995.
- [18] P. Ores. Hypertext publishing – edit trial. *PC Magazine*, 14(3):132–143, 1995.
- [19] Eric van Herwijnen. *Practical SGML (Second Edition)*. Wolters-Kluwer Academic Publishers, Boston, 1994.
- [20] Dominique Vignaud. Éditions du Cercle de la Librairie, Paris, 1990.

## Appendix A: The DTD of the HTML2 language

```

1 <!-- html.dtd
2
3 Document Type Definition for the HyperText Markup Language
4 (HTML DTD)
5
6 $Id: html.dtd,v 1.25 1995/03/29 18:53:13 connolly Exp $
7
8 Author: Daniel W. Connolly <connolly@w3.org>
9 See Also: html.decl, html-0.dtd, html-1.dtd

```

```

10      http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html
11  -->
12
13  <!ENTITY % HTML.Version
14      "-//IETF//DTD HTML 2.0//EN"
15
16      -- Typical usage:
17
18          <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
19          <html>
20              ...
21          </html>
22      --
23  >
24
25
26  <!--===== Feature Test Entities =====>
27
28  <!ENTITY % HTML.Recommended "IGNORE"
29      -- Certain features of the language are necessary for
30      compatibility with widespread usage, but they may
31      compromise the structural integrity of a document.
32      This feature test entity enables a more prescriptive
33      document type definition that eliminates
34      those features.
35  -->
36
37  <![ %HTML.Recommended [
38      <!ENTITY % HTML.Deprecated "IGNORE">
39  ]]>
40
41  <!ENTITY % HTML.Deprecated "INCLUDE"
42      -- Certain features of the language are necessary for
43      compatibility with earlier versions of the specification,
44      but they tend to be used an implemented inconsistently,
45      and their use is deprecated. This feature test entity
46      enables a document type definition that eliminates
47      these features.
48  -->
49
50  <!ENTITY % HTML.Highlighting "INCLUDE"
51      -- Use this feature test entity to validate that a
52      document uses no highlighting tags, which may be
53      ignored on minimal implementations.
54  -->
55
56  <!ENTITY % HTML.Forms "INCLUDE"
57      -- Use this feature test entity to validate that a document
58      contains no forms, which may not be supported in minimal
59      implementations
60  -->
61
62  <!--===== Imported Names =====>
63
64  <!ENTITY % Content-Type "CDATA"
65      -- meaning an internet media type
66      (aka MIME content type, as per RFC1521)
67  -->
68
69  <!ENTITY % HTTP-Method "GET | POST"
70      -- as per HTTP specification, in progress
71  -->
72
73  <!ENTITY % URI "CDATA"

```

```

74      -- The term URI means a CDATA attribute
75      whose value is a Uniform Resource Identifier,
76      as defined by
77      "Universal Resource Identifiers" by Tim Berners-Lee
78      aka RFC 1630
79
80      Note that CDATA attributes are limited by the LITLEN
81      capacity (1024 in the current version of html.decl),
82      so that URIs in HTML have a bounded length.
83
84      -->
85
86
87 <!--==== DTD "Macros" =====>
88
89 <!ENTITY % heading "H1|H2|H3|H4|H5|H6">
90
91 <!ENTITY % list " UL | OL | DIR | MENU " >
92
93
94 <!--==== Character mnemonic entities =====>
95
96
97 <!ENTITY % ISOLat1 PUBLIC
98      "-//IETF//ENTITIES Added Latin 1 for HTML//EN" "iso-lat1.gml">
99
100 %ISOLat1;
101
102 <!ENTITY amp CDATA "&#38;"      -- ampersand      -->
103 <!ENTITY gt CDATA "&#62;"      -- greater than -->
104 <!ENTITY lt CDATA "&#60;"      -- less than    -->
105 <!ENTITY quot CDATA "&#34;"    -- double quote -->
106
107
108 <!--==== SGML Document Access (SDA) Parameter Entities =====>
109
110 <!-- HTML 2.0 contains SGML Document Access (SDA) fixed attributes
111 in support of easy transformation to the International Committee
112 for Accessible Document Design (ICADD) DTD
113      "-//EC-USA-CDA/ICADD//DTD ICADD22//EN".
114 ICADD applications are designed to support usable access to
115 structured information by print-impaired individuals through
116 Braille, large print and voice synthesis. For more information on
117 SDA & ICADD:
118     - ISO 12083:1993, Annex A.8, Facilities for Braille,
119     large print and computer voice
120     - ICADD ListServ
121     <ICADD%ASUACAD.BITNET@ARIZVM1.ccit.arizona.edu>
122     - Usenet news group bit.listserv.easi
123     - Recording for the Blind, +1 800 221 4792
124 -->
125
126 <!ENTITY % SDAFORM "SDAFORM CDATA #FIXED"
127      -- one to one mapping -->
128 <!ENTITY % SDARULE "SDARULE CDATA #FIXED"
129      -- context-sensitive mapping -->
130 <!ENTITY % SDAPREF "SDAPREF CDATA #FIXED"
131      -- generated text prefix -->
132 <!ENTITY % SDASUFF "SDASUFF CDATA #FIXED"
133      -- generated text suffix -->
134 <!ENTITY % SDASUSP "SDASUSP NAME #FIXED"
135      -- suspend transform process -->
136
137

```

```

138 <!--==== Text Markup =====>
139
140 <![ %HTML.Highlighting [
141
142 <!ENTITY % font " TT | B | I ">
143
144 <!ENTITY % phrase "EM | STRONG | CODE | SAMP | KBD | VAR | CITE ">
145
146 <!ENTITY % text "#PCDATA | A | IMG | BR | %phrase | %font">
147
148 <!ELEMENT (%font;%phrase) - - (%text)*>
149 <!ATTLIST ( TT | CODE | SAMP | KBD | VAR )
150     %SDAFORM; "Lit"
151     >
152 <!ATTLIST ( B | STRONG )
153     %SDAFORM; "B"
154     >
155 <!ATTLIST ( I | EM | CITE )
156     %SDAFORM; "It"
157     >
158
159 <!-- <TT>           Typewriter text           -->
160 <!-- <B>           Bold text                   -->
161 <!-- <I>           Italic text                 -->
162
163 <!-- <EM>           Emphasized phrase          -->
164 <!-- <STRONG>      Strong emphasais           -->
165 <!-- <CODE>        Source code phrase         -->
166 <!-- <SAMP>        Sample text or characters  -->
167 <!-- <KBD>         Keyboard phrase, e.g. user input -->
168 <!-- <VAR>         Variable phrase or substituable -->
169 <!-- <CITE>        Name or title of cited work -->
170
171 <!ENTITY % pre.content "#PCDATA | A | HR | BR | %font | %phrase">
172
173 ]]>
174
175 <!ENTITY % text "#PCDATA | A | IMG | BR">
176
177 <!ELEMENT BR      - 0 EMPTY>
178 <!ATTLIST BR
179     %SDAPREF; "&#RE;"
180     >
181
182 <!-- <BR>          Line break                 -->
183
184
185 <!--==== Link Markup =====>
186
187 <![ %HTML.Recommended [
188     <!ENTITY % linkName "ID">
189 ]]>
190
191 <!ENTITY % linkName "CDATA">
192
193 <!ENTITY % linkType "NAME"
194     -- a list of these will be specified at a later date -->
195
196 <!ENTITY % linkExtraAttributes
197     "REL %linkType #IMPLIED
198     REV %linkType #IMPLIED
199     URN CDATA #IMPLIED
200     TITLE CDATA #IMPLIED
201     METHODS NAMES #IMPLIED

```

```

202     ">
203
204 <![ %HTML.Recommended [
205     <!ENTITY % A.content "(%text)*"
206     -- <H1><a name="xxx">Heading</a></H1>
207         is preferred to
208     <a name="xxx"><H1>Heading</H1></a>
209     -->
210 ]]>
211
212 <!ENTITY % A.content "(%heading|%text)*">
213
214 <!ELEMENT A - - %A.content -(A)>
215 <!ATTLIST A
216     HREF %URI #IMPLIED
217     NAME %linkName #IMPLIED
218     %linkExtraAttributes;
219     %SDAPREF; "<Anchor: #AttList>"
220     >
221 <!-- <A> Anchor; source/destination of link -->
222 <!-- <A NAME="..."> Name of this anchor -->
223 <!-- <A HREF="..."> Address of link destination -->
224 <!-- <A URN="..."> Permanent address of destination -->
225 <!-- <A REL=...> Relationship to destination -->
226 <!-- <A REV=...> Relationship of destination to this -->
227 <!-- <A TITLE="..."> Title of destination (advisory) -->
228 <!-- <A METHODS="..."> Operations on destination (advisory) -->
229
230
231 <!--===== Images =====>
232
233 <!ELEMENT IMG - 0 EMPTY>
234 <!ATTLIST IMG
235     SRC %URI; #REQUIRED
236     ALT CDATA #IMPLIED
237     ALIGN (top|middle|bottom) #IMPLIED
238     ISMAP (ISMAP) #IMPLIED
239     %SDAPREF; "<Fig><?SDATrans Img: #AttList>#AttVal(Alt)</Fig>"
240     >
241
242 <!-- <IMG> Image; icon, glyph or illustration -->
243 <!-- <IMG SRC="..."> Address of image object -->
244 <!-- <IMG ALT="..."> Textual alternative -->
245 <!-- <IMG ALIGN=...> Position relative to text -->
246 <!-- <IMG ISMAP> Each pixel can be a link -->
247
248 <!--===== Paragraphs=====>
249
250 <!ELEMENT P - 0 (%text)*>
251 <!ATTLIST P
252     %SDAFORM; "Para"
253     >
254
255 <!-- <P> Paragraph -->
256
257
258 <!--===== Headings, Titles, Sections =====>
259
260 <!ELEMENT HR - 0 EMPTY>
261 <!ATTLIST HR
262     %SDAPREF; "&#RE;&#RE;"
263     >
264
265 <!-- <HR> Horizontal rule -->

```

```

266
267 <!ELEMENT ( %heading ) - - (%text;)*>
268 <!ATTLIST H1
269     %SDAFORM; "H1"
270     >
271 <!ATTLIST H2
272     %SDAFORM; "H2"
273     >
274 <!ATTLIST H3
275     %SDAFORM; "H3"
276     >
277 <!ATTLIST H4
278     %SDAFORM; "H4"
279     >
280 <!ATTLIST H5
281     %SDAFORM; "H5"
282     >
283 <!ATTLIST H6
284     %SDAFORM; "H6"
285     >
286
287 <!-- <H1>      Heading, level 1 -->
288 <!-- <H2>      Heading, level 2 -->
289 <!-- <H3>      Heading, level 3 -->
290 <!-- <H4>      Heading, level 4 -->
291 <!-- <H5>      Heading, level 5 -->
292 <!-- <H6>      Heading, level 6 -->
293
294
295 <!--===== Text Flows =====>
296
297 <![ %HTML.Forms [
298     <!ENTITY % block.forms "BLOCKQUOTE | FORM | ISINDEX">
299 ]]>
300
301 <!ENTITY % block.forms "BLOCKQUOTE">
302
303 <![ %HTML.Deprecated [
304     <!ENTITY % preformatted "PRE | XMP | LISTING">
305 ]]>
306
307 <!ENTITY % preformatted "PRE">
308
309 <!ENTITY % block "P | %list | DL
310     | %preformatted
311     | %block.forms">
312
313 <!ENTITY % flow "(%text|%block)*">
314
315 <!ENTITY % pre.content "#PCDATA | A | HR | BR">
316 <!ELEMENT PRE - - (%pre.content)*>
317 <!ATTLIST PRE
318     WIDTH NUMBER #IMPLIED
319     %SDAFORM; "Lit"
320     >
321
322 <!-- <PRE>      Preformatted text      -->
323 <!-- <PRE WIDTH=...>  Maximum characters per line  -->
324
325 <![ %HTML.Deprecated [
326
327 <!ENTITY % literal "CDATA"
328     -- historical, non-conforming parsing mode where
329     the only markup signal is the end tag

```

```

330         in full
331     -->
332
333 <!ELEMENT (XMP|LISTING) - - %literal>
334 <!ATTLIST XMP
335     %SDAFORM; "Lit"
336     %SDAPREF; "Example:&#RE;"
337     >
338 <!ATTLIST LISTING
339     %SDAFORM; "Lit"
340     %SDAPREF; "Listing:&#RE;"
341     >
342
343 <!-- <XMP>           Example section      -->
344 <!-- <LISTING>      Computer listing     -->
345
346 <!ELEMENT PLAINTEXT - 0 %literal>
347 <!-- <PLAINTEXT>   Plain text passage   -->
348
349 <!ATTLIST PLAINTEXT
350     %SDAFORM; "Lit"
351     >
352 ]]>
353
354
355 <!--===== Lists =====>
356
357 <!ELEMENT DL      - - (DT | DD)+>
358 <!ATTLIST DL
359     COMPACT (COMPACT) #IMPLIED
360     %SDAFORM; "List"
361     %SDAPREF; "Definition List:"
362     >
363
364 <!ELEMENT DT      - 0 (%text)*>
365 <!ATTLIST DT
366     %SDAFORM; "Term"
367     >
368
369 <!ELEMENT DD      - 0 %flow>
370 <!ATTLIST DD
371     %SDAFORM; "Litem"
372     >
373
374 <!-- <DL>           Definition list, or glossary  -->
375 <!-- <DL COMPACT>  Compact style list           -->
376 <!-- <DT>          Term in definition list      -->
377 <!-- <DD>          Definition of term           -->
378
379 <!ELEMENT (OL|UL) - - (LI)+>
380 <!ATTLIST OL
381     COMPACT (COMPACT) #IMPLIED
382     %SDAFORM; "List"
383     >
384 <!ATTLIST UL
385     COMPACT (COMPACT) #IMPLIED
386     %SDAFORM; "List"
387     >
388 <!-- <UL>          Unordered list               -->
389 <!-- <UL COMPACT>  Compact list style           -->
390 <!-- <OL>          Ordered, or numbered list    -->
391 <!-- <OL COMPACT>  Compact list style           -->
392
393

```

```

394 <!ELEMENT (DIR|MENU) - - (LI)+ -(%block)>
395 <!ATTLIST DIR
396     COMPACT (COMPACT) #IMPLIED
397     %SDAFORM; "List"
398     %SDAPREF; "<LHead>Directory</LHead>"
399     >
400 <!ATTLIST MENU
401     COMPACT (COMPACT) #IMPLIED
402     %SDAFORM; "List"
403     %SDAPREF; "<LHead>Menu</LHead>"
404     >
405
406 <!-- <DIR>           Directory list           -->
407 <!-- <DIR COMPACT>   Compact list style       -->
408 <!-- <MENU>          Menu list                 -->
409 <!-- <MENU COMPACT> Compact list style       -->
410
411 <!ELEMENT LI - 0 %flow>
412 <!ATTLIST LI
413     %SDAFORM; "LItem"
414     >
415
416 <!-- <LI>           List item                 -->
417
418 <!--===== Document Body =====>
419
420 <![ %HTML.Recommended [
421     <!ENTITY % body.content "(%heading|%block|HR|ADDRESS|IMG)*"
422     -- <h1>Heading</h1>
423     <p>Text ...
424         is preferred to
425     <h1>Heading</h1>
426     Text ...
427     -->
428 ]]>
429
430 <!ENTITY % body.content "(%heading | %text | %block |
431     HR | ADDRESS)*">
432
433 <!ELEMENT BODY 0 0 %body.content>
434
435 <!-- <BODY>         Document body           -->
436
437 <!ELEMENT BLOCKQUOTE - - %body.content>
438 <!ATTLIST BLOCKQUOTE
439     %SDAFORM; "BQ"
440     >
441
442 <!-- <BLOCKQUOTE>   Quoted passage         -->
443
444 <!ELEMENT ADDRESS - - (%text|P)*>
445 <!ATTLIST ADDRESS
446     %SDAFORM; "Lit"
447     %SDAPREF; "Address:&#RE;"
448     >
449
450 <!-- <ADDRESS>      Address, signature, or byline -->
451
452
453 <!--===== Forms =====>
454
455 <![ %HTML.Forms [
456
457 <!ELEMENT FORM - - %body.content -(FORM) +(INPUT|SELECT|TEXTAREA)>

```

```

458 <!ATTLIST FORM
459     ACTION %URI #IMPLIED
460     METHOD (%HTTP-Method) GET
461     ENCTYPE %Content-Type; "application/x-www-form-urlencoded"
462     %SDAPREF; "<Para>Form:</Para>"
463     %SDASUFF; "<Para>Form End.</Para>"
464     >
465
466 <!-- <FORM>                Fill-out or data-entry form    -->
467 <!-- <FORM ACTION=...>    Address for completed form   -->
468 <!-- <FORM METHOD=...>    Method of submitting form     -->
469 <!-- <FORM ENCTYPE=...>  Representation of form data -->
470
471 <!ENTITY % InputType "(TEXT | PASSWORD | CHECKBOX |
472                    RADIO | SUBMIT | RESET |
473                    IMAGE | HIDDEN )">
474 <!ELEMENT INPUT - 0 EMPTY>
475 <!ATTLIST INPUT
476     TYPE %InputType TEXT
477     NAME CDATA #IMPLIED
478     VALUE CDATA #IMPLIED
479     SRC %URI #IMPLIED
480     CHECKED (CHECKED) #IMPLIED
481     SIZE CDATA #IMPLIED
482     MAXLENGTH NUMBER #IMPLIED
483     ALIGN (top|middle|bottom) #IMPLIED
484     %SDAPREF; "Input: "
485     >
486
487 <!-- <INPUT>                Form input datum            -->
488 <!-- <INPUT TYPE=...>      Type of input interaction   -->
489 <!-- <INPUT NAME=...>     Name of form datum        -->
490 <!-- <INPUT VALUE=...>    Default/initial/selected value -->
491 <!-- <INPUT SRC=...>      Address of image          -->
492 <!-- <INPUT CHECKED>      Initial state is "on"     -->
493 <!-- <INPUT SIZE=...>     Field size hint            -->
494 <!-- <INPUT MAXLENGTH=...> Data length maximum       -->
495 <!-- <INPUT ALIGN=...>   Image alignment          -->
496
497 <!ELEMENT SELECT - - (OPTION+) -(INPUT|SELECT|TEXTAREA)>
498 <!ATTLIST SELECT
499     NAME CDATA #REQUIRED
500     SIZE NUMBER #IMPLIED
501     MULTIPLE (MULTIPLE) #IMPLIED
502     %SDAFORM; "List"
503     %SDAPREF;
504     "<LHead>Select #AttVal(Multiple)</LHead>"
505     >
506
507 <!-- <SELECT>                Selection of option(s)    -->
508 <!-- <SELECT NAME=...>     Name of form datum        -->
509 <!-- <SELECT SIZE=...>    Options displayed at a time -->
510 <!-- <SELECT MULTIPLE>    Multiple selections allowed -->
511
512 <!ELEMENT OPTION - 0 (#PCDATA)*>
513 <!ATTLIST OPTION
514     SELECTED (SELECTED) #IMPLIED
515     VALUE CDATA #IMPLIED
516     %SDAFORM; "Litem"
517     %SDAPREF;
518     "Option: #AttVal(Value) #AttVal(Selected)"
519     >
520
521 <!-- <OPTION>                A selection option            -->

```

```

522 <!-- <OPTION SELECTED>           Initial state           -->
523 <!-- <OPTION VALUE="...">      Form datum value for this option-->
524
525 <!ELEMENT TEXTAREA - - (#PCDATA)* -(INPUT|SELECT|TEXTAREA)>
526 <!ATTLIST TEXTAREA
527     NAME CDATA #REQUIRED
528     ROWS NUMBER #REQUIRED
529     COLS NUMBER #REQUIRED
530     %SDAFORM; "Para"
531     %SDAPREF; "Input Text -- #AttVal(Name): "
532     >
533
534 <!-- <TEXTAREA>                   An area for text input           -->
535 <!-- <TEXTAREA NAME=...>          Name of form datum             -->
536 <!-- <TEXTAREA ROWS=...>         Height of area                 -->
537 <!-- <TEXTAREA COLS=...>        Width of area                  -->
538
539 ]]>
540
541
542 <!--===== Document Head =====>
543
544 <![ %HTML.Recommended [
545     <!ENTITY % head.extra "META* & LINK*">
546 ]]>
547
548 <!ENTITY % head.extra "NEXTID? & META* & LINK*">
549
550 <!ENTITY % head.content "TITLE & ISINDEX? & BASE? &
551     (%head.extra)">
552
553 <!ELEMENT HEAD 0 0 (%head.content)>
554
555 <!-- <HEAD>           Document head -->
556
557 <!ELEMENT TITLE - - (#PCDATA)*>
558 <!ATTLIST TITLE
559     %SDAFORM; "Ti"    >
560
561 <!-- <TITLE>         Title of document -->
562
563 <!ELEMENT LINK - 0 EMPTY>
564 <!ATTLIST LINK
565     HREF %URI #REQUIRED
566     %linkExtraAttributes;
567     %SDAPREF; "Linked to : #AttVal (TITLE) (URN) (HREF)">    >
568
569 <!-- <LINK>          Link from this document           -->
570 <!-- <LINK HREF="..."> Address of link destination     -->
571 <!-- <LINK URN="..."> Lasting name of destination      -->
572 <!-- <LINK REL=...> Relationship to destination         -->
573 <!-- <LINK REV=...> Relationship of destination to this -->
574 <!-- <LINK TITLE="..."> Title of destination (advisory) -->
575 <!-- <LINK METHODS="..."> Operations allowed (advisory) -->
576
577 <!ELEMENT ISINDEX - 0 EMPTY>
578 <!ATTLIST ISINDEX
579     %SDAPREF;
580     "<Para>[Document is indexed/searchable.]</Para>">
581
582 <!-- <ISINDEX>      Document is a searchable index     -->
583
584 <!ELEMENT BASE - 0 EMPTY>
585 <!ATTLIST BASE

```

```

586         HREF %URI; #REQUIRED      >
587
588 <!-- <BASE>           Base context document      -->
589 <!-- <BASE HREF="..."> Address for this document  -->
590
591 <!ELEMENT NEXTID - O EMPTY>
592 <!ATTLIST NEXTID
593         N %linkName #REQUIRED      >
594
595 <!-- <NEXTID>           Next ID to use for link name      -->
596 <!-- <NEXTID N=...>     Next ID to use for link name      -->
597
598 <!ELEMENT META - O EMPTY>
599 <!ATTLIST META
600         HTTP-EQUIV NAME #IMPLIED
601         NAME NAME #IMPLIED
602         CONTENT CDATA #REQUIRED  >
603
604 <!-- <META>           Generic Metainformation      -->
605 <!-- <META HTTP-EQUIV=...> HTTP response header name  -->
606 <!-- <META NAME=...>     Metainformation name      -->
607 <!-- <META CONTENT="..."> Associated information  -->
608
609 <!--===== Document Structure =====>
610
611 <![ %HTML.Deprecated [
612         <!ENTITY % html.content "HEAD, BODY, PLAINTEXT?">
613 ]]>
614 <!ENTITY % html.content "HEAD, BODY">
615
616 <!ELEMENT HTML O O (%html.content)>
617 <!ENTITY % version.attr "VERSION CDATA #FIXED '%HTML.Version;'">
618
619 <!ATTLIST HTML
620         %version.attr;
621         %SDAFORM; "Book"
622         >
623
624 <!-- <HTML>           HTML Document      -->

```

## Appendix B: The HTML2 SGML declaration

```

1 <!SGML "ISO 8879:1986"
2 --
3         SGML Declaration for HyperText Markup Language (HTML).
4
5 --
6
7 CHARSET
8         BASESET "ISO 646:1983//CHARSET
9                 International Reference Version
10                (IRV)//ESC 2/5 4/0"
11         DESCSET 0 9 UNUSED
12                9 2 9
13                11 2 UNUSED
14                13 1 13
15                14 18 UNUSED
16                32 95 32
17                127 1 UNUSED
18         BASESET "ISO Registration Number 100//CHARSET
19                 ECMA-94 Right Part of
20                 Latin Alphabet Nr. 1//ESC 2/13 4/1"

```

```

21
22         DESCSET 128 32  UNUSED
23             160 96   32
24
25 CAPACITY      SGMLREF
26             TOTALCAP      150000
27             GRPCAP        150000
28
29 SCOPE  DOCUMENT
30 SYNTAX
31     SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
32             17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 127
33     BASESET  "ISO 646:1983//CHARSET
34             International Reference Version
35             (IRV)//ESC 2/5 4/0"
36     DESCSET 0 128 0
37     FUNCTION
38         RE      13
39         RS      10
40         SPACE   32
41         TAB SEPCHAR 9
42
43
44     NAMING  LCNMSTRT ""
45            UCNMSTRT ""
46            LCNMCHAR ".-"
47            UCNMCHAR ".-"
48            NAMECASE GENERAL YES
49            ENTITY NO
50     DELIM  GENERAL SGMLREF
51            SHORTREF SGMLREF
52     NAMES  SGMLREF
53     QUANTITY SGMLREF
54            ATTSPLEN 2100
55            LITLEN 1024
56            NAMELEN 72  -- somewhat arbitrary; taken from
57                       internet line length conventions --
58            PILEN 1024
59            TAGLEN 2100
60            GRPGTCNT 150
61            GRPCNT 64
62
63 FEATURES
64     MINIMIZE
65         DATATAG NO
66         OMITTAG YES
67         RANK NO
68         SHORTTAG YES
69     LINK
70         SIMPLE NO
71         IMPLICIT NO
72         EXPLICIT NO
73     OTHER
74         CONCUR NO
75         SUBDOC NO
76         FORMAL YES
77     APPINFO "SDA" -- conforming SGML Document Access application
78             --
79 >
80 <!--
81     $Id: html.decl,v 1.14 1995/02/10 22:20:05 connolly Exp $
82
83     Author: Daniel W. Connolly <connolly@hal.com>
84

```

```

85      See also: http://www.hal.com/%7Econnolly/html-spec
86      http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html
87  -->

```

## Appendix C: The SGML open HTML catalog file

SGML Open is an industry consortium dedicated to encouraging the adoption of SGML as a standard for document and data interchange. It proposes a standard way for mapping entity and other external references in a DTD to file names via a "catalog" file. Below is an example of such a catalog file for HTML.

## Appendix D: The ISO-Latin1 entity set

To have an idea of how character entity sets are defined in practice, below is shown the file corresponding to Latin1 (standard ISO/IEC 8859-1), available as SGML public entity set ISO1at1 with ISO 8879.

```

1  <!-- (C) International Organization for Standardization 1986
2      Permission to copy in any form is granted for use with
3      conforming SGML systems and applications as defined in
4      ISO 8879, provided this notice is included in all copies.
5  -->
6  <!-- Character entity set. Typical invocation:
7      <!ENTITY % ISO1at1 PUBLIC
8          "ISO 8879-1986/ENTITIES Added Latin 1//EN">
9      %ISO1at1;
10 -->
11 <!ENTITY aacute SDATA "[aacute]"--=small a, acute accent-->
12 <!ENTITY Aacute SDATA "[Aacute]"--=capital A, acute accent-->
13 <!ENTITY acirc SDATA "[acirc]"--=small a, circumflex accent-->
14 <!ENTITY Acirc SDATA "[Acirc]"--=capital A, circumflex accent-->
15 <!ENTITY agrave SDATA "[agrave]"--=small a, grave accent-->
16 <!ENTITY Agrave SDATA "[Agrave]"--=capital A, grave accent-->
17 <!ENTITY aring SDATA "[aring]"--=small a, ring-->
18 <!ENTITY Aring SDATA "[Aring]"--=capital A, ring-->
19 <!ENTITY atilde SDATA "[atilde]"--=small a, tilde-->
20 <!ENTITY Atilde SDATA "[Atilde]"--=capital A, tilde-->
21 <!ENTITY auml SDATA "[auml]"--=small a, dieresis or umlaut mark-->
22 <!ENTITY Auml SDATA "[Auml]"--=capital A, dieresis or umlaut mark-->
23 <!ENTITY aelig SDATA "[aelig]"--=small ae diphthong (ligature)-->
24 <!ENTITY AElig SDATA "[AElig]"--=capital AE diphthong (ligature)-->
25 <!ENTITY ccedil SDATA "[ccedil]"--=small c, cedilla-->
26 <!ENTITY Ccedil SDATA "[Ccedil]"--=capital C, cedilla-->
27 <!ENTITY eth SDATA "[eth]"--=small eth, Icelandic-->
28 <!ENTITY ETH SDATA "[ETH]"--=capital Eth, Icelandic-->
29 <!ENTITY eacute SDATA "[eacute]"--=small e, acute accent-->
30 <!ENTITY Eacute SDATA "[Eacute]"--=capital E, acute accent-->
31 <!ENTITY ecirc SDATA "[ecirc]"--=small e, circumflex accent-->
32 <!ENTITY Ecirc SDATA "[Ecirc]"--=capital E, circumflex accent-->
33 <!ENTITY egrave SDATA "[egrave]"--=small e, grave accent-->
34 <!ENTITY Egrave SDATA "[Egrave]"--=capital E, grave accent-->
35 <!ENTITY euml SDATA "[euml]"--=small e, dieresis or umlaut mark-->
36 <!ENTITY Euml SDATA "[Euml]"--=capital E, dieresis or umlaut mark-->
37 <!ENTITY iacute SDATA "[iacute]"--=small i, acute accent-->

```

```

38 <!ENTITY Iacute SDATA "[Iacute]"--=capital I, acute accent-->
39 <!ENTITY icirc SDATA "[icirc]"--=small i, circumflex accent-->
40 <!ENTITY Icirc SDATA "[Icirc]"--=capital I, circumflex accent-->
41 <!ENTITY igrave SDATA "[igrave]"--=small i, grave accent-->
42 <!ENTITY Igrave SDATA "[Igrave]"--=capital I, grave accent-->
43 <!ENTITY iuml SDATA "[iuml]"--=small i, dieresis or umlaut mark-->
44 <!ENTITY Iuml SDATA "[Iuml]"--=capital I, dieresis or umlaut mark-->
45 <!ENTITY ntilde SDATA "[ntilde]"--=small n, tilde-->
46 <!ENTITY Ntilde SDATA "[Ntilde]"--=capital N, tilde-->
47 <!ENTITY oacute SDATA "[oacute]"--=small o, acute accent-->
48 <!ENTITY Oacute SDATA "[Oacute]"--=capital O, acute accent-->
49 <!ENTITY ocirc SDATA "[ocirc]"--=small o, circumflex accent-->
50 <!ENTITY Ocirc SDATA "[Ocirc]"--=capital O, circumflex accent-->
51 <!ENTITY ograve SDATA "[ograve]"--=small o, grave accent-->
52 <!ENTITY Ograve SDATA "[Ograve]"--=capital O, grave accent-->
53 <!ENTITY oslash SDATA "[oslash]"--=small o, slash-->
54 <!ENTITY Oslash SDATA "[Oslash]"--=capital O, slash-->
55 <!ENTITY otilde SDATA "[otilde]"--=small o, tilde-->
56 <!ENTITY Otilde SDATA "[Otilde]"--=capital O, tilde-->
57 <!ENTITY ouml SDATA "[ouml]"--=small o, dieresis or umlaut mark-->
58 <!ENTITY Ouml SDATA "[Ouml]"--=capital O, dieresis or umlaut mark-->
59 <!ENTITY szlig SDATA "[szlig]"--=small sharp s, German (sz ligature)-->
60 <!ENTITY thorn SDATA "[thorn]"--=small thorn, Icelandic-->
61 <!ENTITY THORN SDATA "[THORN]"--=capital THORN, Icelandic-->
62 <!ENTITY uacute SDATA "[uacute]"--=small u, acute accent-->
63 <!ENTITY Uacute SDATA "[Uacute]"--=capital U, acute accent-->
64 <!ENTITY ucirc SDATA "[ucirc]"--=small u, circumflex accent-->
65 <!ENTITY Ucirc SDATA "[Ucirc]"--=capital U, circumflex accent-->
66 <!ENTITY ugrave SDATA "[ugrave]"--=small u, grave accent-->
67 <!ENTITY Ugrave SDATA "[Ugrave]"--=capital U, grave accent-->
68 <!ENTITY uuml SDATA "[uuml]"--=small u, dieresis or umlaut mark-->
69 <!ENTITY Uuml SDATA "[Uuml]"--=capital U, dieresis or umlaut mark-->
70 <!ENTITY yacute SDATA "[yacute]"--=small y, acute accent-->
71 <!ENTITY Yacute SDATA "[Yacute]"--=capital Y, acute accent-->
72 <!ENTITY yuml SDATA "[yuml]"--=small y, dieresis or umlaut mark-->
73

```

## Appendix E: The HTML3 DTD – Tables and mathematics parts

This appendix shows those parts of the HTML3 DTD that relate to tables and mathematics.

```

1 <!--===== Captions =====>
2
3 <!ELEMENT CAPTION - - (%text;)+ -- table or figure caption -->
4 <!ATTLIST CAPTION
5   %attrs;
6   align (top|bottom|left|right) #IMPLIED
7   >
8 <!--===== Tables =====>
9
10 <!--
11   Tables and figures can be aligned in several ways:
12
13   bleedleft  flush left with the left (window) border
14   left       flush left with the left text margin
15   center     centered (text flow is disabled for this mode)
16   right      flush right with the right text margin
17   bleedright flush right with the right (window) border
18   justify    when applicable the table/figure should stretch
19             to fill space between the text margins

```

```

20
21 Note: text will flow around the table or figure if the browser
22 judges there is enough room and the alignment is not centered
23 or justified. The table or figure may itself be part of the
24 text flow around some earlier figure. You can in this case use
25 the clear or needs attributes to move the new table or figure
26 down the page beyond the obstructing earlier figure. Similarly,
27 you can use the clear or needs attributes with other elements
28 such as headers and lists to move them further down the page.
29 -->
30
31 <!ENTITY % block.align
32     "align (bleedleft|left|center|right|bleedright|justify) center">
33
34 <!--
35     The HTML 3.0 table model has been chosen for its simplicity
36     and the ease in writing filters from common DTP packages.
37
38     By default the table is automatically sized according to the
39     cell contents and the current window size. Specifying the columns
40     widths using the colspec attribute allows browsers to start
41     displaying the table without having to wait for last row.
42
43     The colspec attribute is a list of column widths and alignment
44     specifications. The columns are listed from left to right with
45     a capital letter followed by a number, e.g. COLSPEC="L20 C8 L40".
46     The letter is L for left, C for center, R for right alignment of
47     cell contents. J is for justification, when feasible, otherwise
48     this is treated in the same way as L for left alignment.
49     Column entries are delimited by one or more space characters.
50
51     The number specifies the width in em's, pixels or as a
52     fractional value of the table width, as according to the
53     associated units attribute. This approach is more compact
54     than used with most SGML table models and chosen to simplify
55     hand entry. The width attribute allows you to specify the
56     width of the table in pixels, em units or as a percentage
57     of the space between the current left and right margins.
58
59     To assist with rendering to speech, row and column headers
60     can be given short names using the AXIS attribute. The AXES
61     attribute is used to explicitly specify the row and column
62     names for use with each cell. Otherwise browsers can follow
63     up columns and left along rows (right for some languages)
64     to find the corresponding header cells.
65
66     Table content model: Braille limits the width of tables,
67     placing severe limits on column widths. User agents need
68     to render big cells by moving the content to a note placed
69     before the table. The cell is then rendered as a link to
70     the corresponding note.
71
72     To assist with formatting tables to paged media, authors
73     can differentiate leading and trailing rows that are to
74     be duplicated when splitting tables across page boundaries.
75     The recommended way is to subclass rows with the CLASS attribute
76     For example: <TR CLASS=Header>, <TR CLASS=Footer> are used for
77     header and footer rows. Paged browsers insert footer rows at
78     the bottom of the current page and header rows at the top of
79     the new page, followed by the remaining body rows.
80 -->
81
82 <!ELEMENT TABLE - - (CAPTION?, TR*) -- mixed headers and data -->
83 <!ATTLIST TABLE

```

```

84     %attrs;
85     %needs; -- for control of text flow --
86     border (border) #IMPLIED -- draw borders --
87     colspec CDATA #IMPLIED -- column widths and alignment --
88     units (em|pixels|relative) em -- units for column widths --
89     width NUMBER #IMPLIED -- absolute or percentage width --
90     %block.align; -- horizontal alignment --
91     nowrap (nowrap) #IMPLIED -- don't wrap words --
92     >
93
94 <!ENTITY % cell "TH | TD">
95 <!ENTITY % vertical.align "top|middle|bottom|baseline">
96
97 <!--
98     Browsers should tolerate an omission of the first <TR>
99     tag as it is implied by the context. Missing trailing
100    <TR>s implied by rowspans should be ignored.
101
102    The alignment attributes act as defaults for rows
103    overriding the colspec attribute and being in turn
104    overridden by alignment attributes on cell elements.
105    Use valign=baseline when you want to ensure that text
106    in different cells on the same row is aligned on the
107    same baseline regardless of fonts. It only applies
108    when the cells contain a single line of text.
109 -->
110
111 <!ELEMENT TR - 0 (%cell)* -- row container -->
112 <!ATTLIST TR
113     %attrs;
114     align (left|center|right|justify) #IMPLIED
115     valign (%vertical.align) top -- vertical alignment --
116     nowrap (nowrap) #IMPLIED -- don't wrap words --
117     >
118
119 <!--
120     Note that table cells can include nested tables.
121     Missing cells are considered to be empty, while
122     missing rows should be ignored, i.e. if a cell
123     spans a row and there are no further TR elements
124     then the implied row should be ignored.
125 -->
126
127 <!ELEMENT (%cell) - 0 %body.content>
128 <!ATTLIST (%cell)
129     %attrs;
130     colspan NUMBER 1 -- columns spanned --
131     rowspan NUMBER 1 -- rows spanned --
132     align (left|center|right|justify) #IMPLIED
133     valign (%vertical.align) top -- vertical alignment --
134     nowrap (nowrap) #IMPLIED -- don't wrap words --
135     axis CDATA #IMPLIED -- axis name, defaults to element content --
136     axes CDATA #IMPLIED -- comma separated list of axis names --
137     >
138
139 <!--===== Entities for math symbols =====>
140
141 <!-- ISO subset chosen for use with the widely available Adobe math font -->
142
143 <!ENTITY % HTMLmath PUBLIC
144     "-//IETF//ENTITIES Math and Greek for HTML//EN">
145 %HTMLmath;
146
147 <!--===== Math =====>

```

```

148
149 <!-- Use &thinsp; &emsp; etc for greater control of spacing. -->
150
151 <!-- Subscripts and Superscripts
152
153 <SUB> and <SUP> are used for subscripts and superscripts.
154
155
156 X <SUP>i</SUP>Y<SUP>j</SUP> is X Y
157
158 i.e. the space following the X disambiguates the binding.
159 The align attribute can be used for horizontal alignment,
160 e.g. to explicitly place an index above an element:
161
162 X<sup align=center>i</sup> produces X
163
164 Short references are defined for superscripts, subscripts and boxes
165 to save typing when manually editing HTML math, e.g.
166
167 x^2^ is mapped to x<sup>2</sup>
168 y_z_ is mapped to y<sub>z</sub>
169 {a+b} is mapped to <box>a + b</box>
170
171 Note that these only apply within the MATH element and can't be
172 used in normal text!
173 -->
174 <!ENTITY REF1 STARTTAG "SUP">
175 <!ENTITY REF2 ENDTAG "SUP">
176 <!ENTITY REF3 STARTTAG "SUB">
177 <!ENTITY REF4 ENDTAG "SUB">
178 <!ENTITY REF5 STARTTAG "BOX">
179 <!ENTITY REF6 ENDTAG "BOX">
180
181 <!USEMAP MAP1 MATH>
182 <!USEMAP MAP2 SUP>
183 <!USEMAP MAP3 SUB>
184 <!USEMAP MAP4 BOX>
185
186 <!SHORTREF MAP1 "^" REF1
187 " " REF3
188 "{" REF5 >
189
190 <!SHORTREF MAP2 "^" REF2
191 " " REF3
192 "{" REF5 >
193
194 <!SHORTREF MAP3 "_" REF4
195 "^" REF1
196 "{" REF5 >
197
198 <!SHORTREF MAP4 "]" REF6
199 "^" REF1
200 " " REF3
201 "{" REF5 >
202
203 <!--
204 The inclusion of %math and exclusion of %notmath is used here
205 to alter the content model for the B, SUB and SUP elements,
206 to limit them to formulae rather than general text elements.
207 -->
208
209 <!ENTITY % mathvec "VEC|BAR|DOT|DDOT|HAT|TILDE" -- common accents -->
210 <!ENTITY % mathface "B|T|BT" -- control of font face -->
211 <!ENTITY % math "BOX|ABOVE|BELOW|&mathvec|ROOT|SQRT|ARRAY|SUB|SUP|&mathface">

```

```

212 <!ENTITY % formula "#PCDATA|%/math">
213
214 <!ELEMENT MATH -- ((#PCDATA)* -(%notmath) +(%math))>
215 <!ATTLIST MATH
216     id ID #IMPLIED
217     model CDATA #IMPLIED>
218
219 <!-- The BOX element acts as brackets. Delimiters are optional and
220 stretch to match the height of the box. The OVER element is used
221 when you want a line between numerator and denominator. This line
222 is suppressed with the alternative ATOP element. CHOOSE acts like
223 ATOP but adds enclosing round brackets as a convenience for binomial
224 coefficients. Note the use of { and } as shorthand for <BOX> and
225 </BOX> respectively:
226
227     1 + X
228 {1 + X<OVER>Y} is -----
229                      Y
230
231     a + b
232 {a + b<ATOP>c - d} is
233                      c - d
234
235 The delimiters are represented using the LEFT and RIGHT
236 elements as in:
237
238 {[<LEFT>x + y<RIGHT>]} is [ x + y ]
239 {(<LEFT>a<RIGHT>)} is (a)
240 {||<LEFT>a<RIGHT>||} is || a ||
241
242 Use &lbrace; and &rbrace; for "{" and "}" respectively as
243 these symbols are used as shorthand for BOX, e.g.
244
245 {&lbrace;<LEFT>a+b<RIGHT>&rbrace;} is {a+b}
246
247 You can stretch definite integrals to match the integrand, e.g.
248
249 {&int;<SUB>a</SUB><SUP>b</SUP><LEFT>{f(x)<over>1+x} dx}
250
251     b
252     / f(x)
253     | ---- dx
254     / 1 + x
255     a
256
257 Note the complex content model for BOX is a work around
258 for the absence of support for infix operators in SGML.
259
260 You can get oversized delimiters with the SIZE attribute,
261 for example <BOX SIZE=large>(<LEFT>...<RIGHT>)</BOX>
262
263 Note that the names of common functions are recognized
264 by the parser without the need to use "&" and ";" around
265 them, e.g. int, sum, sin, cos, tan, ...
266 -->
267
268 <!ELEMENT BOX -- ((%formula)*, (LEFT, (%formula)*)?,
269 ((OVER|ATOP|CHOOSE), (%formula)*)?,
270 (RIGHT, (%formula)*)?)>
271 <!ATTLIST BOX
272     size (normal|medium|large|huge) normal -- oversized delims -->
273
274 <!ELEMENT (OVER|ATOP|CHOOSE|LEFT|RIGHT) - O EMPTY>
275

```

```

276 <!-- Horizontal line drawn ABOVE contents
277     The symbol attribute allows authors to supply
278     an entity name for an accent, arrow symbol etc.
279     Generalisation of LaTeX's overline command.
280 -->
281
282 <!ELEMENT ABOVE - - (%formula)+>
283 <!ATTLIST ABOVE symbol ENTITY #IMPLIED>
284
285 <!-- Horizontal line drawn BELOW contents
286     The symbol attribute allows authors to
287     supply an entity name for an arrow symbol etc.
288     Generalisation of LaTeX's underline command.
289 -->
290
291 <!ELEMENT BELOW - - (%formula)+>
292 <!ATTLIST BELOW symbol ENTITY #IMPLIED>
293
294 <!-- Convenience tags for common accents:
295     vec, bar, dot, ddot, hat and tilde
296 -->
297
298 <!ELEMENT (%mathvec) - - (%formula)+>
299
300 <!--
301     T and BT are used to designate terms which should
302     be rendered in an upright font (& bold face for BT)
303 -->
304
305 <!ELEMENT (T|BT) - - (%formula)+>
306 <!ATTLIST (T|BT) class NAMES #IMPLIED>
307
308 <!-- Roots e.g. <ROOT>3<OF>1+x</ROOT> -->
309
310 <!ELEMENT ROOT - - ((%formula)+, OF, (%formula)+)>
311 <!ELEMENT OF - O (%formula)* -- what the root applies to -->
312
313 <!ELEMENT SQRT - - (%formula)* -- square root convenience tag -->
314
315 <!-- LaTeX like arrays. The COLDEF attribute specifies
316     a single capital letter for each column determining
317     how the column should be aligned, e.g. coldef="CCC"
318
319     "L"    left
320     "C"    center
321     "R"    right
322
323     An optional separator letter can occur between columns
324     and should be one of + - or =, e.g. "C+C+C=C".
325     Whitespace within coldef is ignored. By default, the
326     columns are all centered.
327
328     The ALIGN attribute alters the vertical position of the
329     array as compared with preceding and following expressions.
330
331     Use LDELIM and RDELIM attributes for delimiter entities.
332     When the LABELS attribute is present, the array is
333     displayed with the first row and the first column as
334     labels displaced from the other elements. In this case,
335     the first element of the first row should normally be
336     left blank.
337
338     Use &vdots; &cdots; and &ddots; for vertical, horizontal
339     and diagonal ellipsis dots. Use &dotfill; to fill an array

```

```

340     cell with horizontal dots (e.g. for a full row).
341     Note &ldots; places the dots on the baseline, while &cdots;
342     places them higher up.
343 -->
344
345 <!ELEMENT ARRAY - - (ROW)+>
346 <!ATTLIST ARRAY
347     align (top|middle|bottom) middle -- vertical alignment --
348     coldef CDATA #IMPLIED -- column alignment and separator --
349     ldelim NAMES #IMPLIED -- stretchy left delimiter --
350     rdelim NAMES #IMPLIED -- stretchy right delimiter --
351     labels (labels) #IMPLIED -- TeX's \bordermatrix style -->
352
353 <!ELEMENT ROW - 0 (ITEM)*>
354 <!ELEMENT ITEM - 0 (%formula)*>
355 <!ATTLIST ITEM
356     align CDATA #IMPLIED -- override coldef alignment --
357     colspan NUMBER 1 -- merge columns as per TABLE --
358     rowspan NUMBER 1 -- merge rows as per TABLE -->

```

## Appendix F: The ISO-12083 mathematics DTD

This appendix shows the mathematics DTD math.dtd of the ISO 12083 DTD.

```

1  <!-- This is the ISO12083:1994 document type definition for Mathematics -->
2
3  <!-- Copyright: (C) International Organization for Standardization 1994.
4  Permission to copy in any form is granted for use with conforming SGML
5  systems and applications as defined in ISO 8879:1986, provided this notice
6  is included in all copies. -->
7
8  <!-- ===== -->
9  <!-- PUBLIC DOCUMENT TYPE DEFINITION SUBSET -->
10 <!-- ===== -->
11
12 <!--
13 This DTD is included by the Book and Article DTDs of ISO12083:1994.
14 As it is a separate entity it may also be included by other DTDs.
15
16 Since there is no consensus on how to describe the semantics of formulas,
17 it only describes their presentational or visual structure. Since, however,
18 there is a strong need for such description (especially within the
19 print-disabled community), it is recommended that the following
20 declaration be added where there is a requirement for a consistent,
21 standardized mechanism to carry semantic meanings for the SGML
22 elements declared throughout this part of this International Standard:
23
24 <!ENTITY % SDAMAP "SDAMAP NAME #IMPLIED" >
25
26 and that the attribute represented by %SDAMAP; be made available for
27 all elements which may require a semantic association, or, in the simpler
28 case, be added to all elements in this DTD. -->
29
30
31
32 <!-- ===== -->
33 <!-- Parameter entities describing the possible contents of formulas. -->
34 <!-- ===== -->
35
36 <!ENTITY % p.trans "bold|italic|sansser|typewrit|smallcap|roman"
37 -- character transformations -->
38 <!ENTITY % m.math "fraction|subform|sup|inf|top|bottom|middle|fence|mark|

```

```

39 post|box|overline|undrlne|radical|array|hspace|vspace|break|markref|
40 #PCDATA" -- mathematical formula elements -->
41
42
43
44 <!-- ===== -->
45 <!-- Accessible Document and other Parameter Entities
46 If this DTD is not imbedded by a ISO12083:1994 Book or Article,
47 the comment delimiters should be removed. -->
48 <!-- ===== -->
49
50 <!--ENTITY % SDAFORM "SDAFORM CDATA #FIXED" -->
51 <!--ENTITY % SDARULE "SDARULE CDATA #FIXED" -->
52 <!--ENTITY % SDAPREF "SDAPREF CDATA #FIXED" -->
53 <!--ENTITY % SDASUFF "SDASUFF CDATA #FIXED" -->
54 <!--ENTITY % SDASUSP "SDASUSP NAME #FIXED" -->
55
56
57
58 <!-- ===== -->
59 <!-- This entity is for an attribute to indicate which alphabet is
60 used in the element (formula, dformula). You may change this to
61 a notation attribute, where the notation could describe a
62 keyboard mapping. Please modify the set as necessary.
63 If this DTD is not imbedded by a ISO12083:1994 Book or Article,
64 the comment delimiters should be removed. -->
65 <!-- ===== -->
66
67 <!-- ENTITY % a.types "(latin|greek|cyrillic|hebrew|kanji) latin" -->
68
69
70 <!-- ===== -->
71 <!-- character transformations -->
72 <!-- ===== -->
73
74 <!-- ELEMENT MIN CONTENT EXPLANATIONS -->
75 <!ELEMENT bold - - (%p.trans;|#PCDATA)* -- bold -->
76 <!ELEMENT italic - - (%p.trans;|#PCDATA)* -- italic -->
77 <!ELEMENT sansser - - (%p.trans;|#PCDATA)* -- sans serif -->
78 <!ELEMENT typewrit - - (%p.trans;|#PCDATA)* -- typewriter -->
79 <!ELEMENT smallcap - - (%p.trans;|#PCDATA)* -- small caps -->
80 <!ELEMENT roman - - (%p.trans;|#PCDATA)* -- roman -->
81
82
83 <!-- ===== -->
84 <!-- Fractions -->
85 <!-- ===== -->
86
87 <!-- ELEMENT MIN CONTENT EXPLANATIONS -->
88 <!ELEMENT fraction - - (num, den) -- fraction -->
89 <!ELEMENT num - - (%p.trans;|#m.math;)* -- numerator -->
90 <!ELEMENT den - - (%p.trans;|#m.math;)* -- denominator -->
91 <!-- ELEMENT NAME VALUE DEFAULT -->
92 <!ATTLIST fraction shape (built|case) #IMPLIED -->
93 align (left|center|right) -->
94 center -->
95 style (single|double|triple|dash|dot|bold|blank|none) -->
96 single -->
97
98
99
100 <!-- ===== -->
101 <!-- Superiors, inferiors, accents, over and under -->
102 <!-- ===== -->

```

```

103
104 <!-- ELEMENT MIN CONTENT EXPLANATIONS -->
105 <!ELEMENT sup - - (%p.trans;|%m.math;)* -- superior -->
106 <!ELEMENT inf - - (%p.trans;|%m.math;)* -- inferior -->
107 <!-- ELEMENT NAME VALUE DEFAULT -->
108 <!ATTLIST sup location (pre|post) post
109 arrange (compact|stagger)
110 compact >
111 <!ATTLIST inf location (pre|post) post
112 arrange (compact|stagger) compact >
113
114
115 <!-- ===== -->
116 <!-- Embellishments -->
117 <!-- ===== -->
118
119 <!-- ELEMENT MIN CONTENT EXPLANATIONS -->
120 <!ELEMENT top - - (%p.trans;|%m.math;)*
121 -- top embellishment -->
122 <!ELEMENT middle - - (%p.trans;|%m.math;)*
123 -- middle, or "through" -->
124 <!ELEMENT bottom - - (%p.trans;|%m.math;)*
125 -- bottom embellishment -->
126 <!-- ELEMENT NAME VALUE DEFAULT -->
127 <!ATTLIST top align (left|center|right)
128 center
129 sizeid ID #IMPLIED
130 -- to pass on the height -->
131 <!ATTLIST middle align (left|center|right)
132 center
133 sizeid ID #IMPLIED
134 -- to pass on the height -->
135 <!ATTLIST bottom align (left|center|right)
136 center
137 sizeid ID #IMPLIED
138 -- to pass on the height -->
139
140
141 <!-- The subform element is defined later -->
142
143
144
145 <!-- ===== -->
146 <!-- Fences, boxes, overlines and underlines -->
147 <!-- ===== -->
148
149 <!-- ELEMENT MIN CONTENT EXPLANATIONS -->
150 <!ELEMENT mark - 0 EMPTY >
151 <!ELEMENT fence - - (%p.trans;|%m.math;)* -- fence -->
152 <!ELEMENT post - 0 EMPTY -- post -->
153 <!ELEMENT box - - (%p.trans;|%m.math;)* -- box -->
154 <!ELEMENT overline - - (%p.trans;|%m.math;)* -- overline -->
155 <!ELEMENT underline - - (%p.trans;|%m.math;)* -- underline -->
156 <!-- ELEMENT NAME VALUE DEFAULT -->
157 <!ATTLIST mark id ID #REQUIRED >
158 <!ATTLIST fence lpost CDATA "|" -- left post --
159 rpost CDATA "|" -- right post --
160 style (single|double|triple|dash|dot|bold|blank|none)
161 single
162 sizeid ID #IMPLIED
163 -- to pass on the height --
164 sizeref IDREF #IMPLIED
165 -- to pick up a height -->
166 <!ATTLIST post post CDATA "|"
```



```

231
232 <!-- ===== -->
233 <!-- Spacing -->
234 <!-- ===== -->
235
236 <!-- ELEMENT          MIN  CONTENT          EXPLANATIONS  -->
237 <!ELEMENT hspace     - 0 EMPTY          -- horizontal spacing -->
238 <!ELEMENT vspace     - 0 EMPTY          -- vertical spacing  -->
239 <!ELEMENT break      - 0 EMPTY          -- turn line, break  -->
240 <!ELEMENT markref    - 0 EMPTY          -- hmark reference   -->
241
242 <!-- ELEMENT  NAME    VALUE      DEFAULT      -->
243 <!ATTLIST hspace  space  CDATA      "1 mm"       -->
244              -- units as required -->
245 <!ATTLIST vspace  space  CDATA      "1 mm"       -->
246              -- units as required -->
247 <!ATTLIST markref refid  IDREF      #REQUIRED    -->
248              direct  (hor|ver)  hor         -->
249              -- horizontal or vertical -->
250
251
252 <!-- ===== -->
253 <!-- the formula elements -->
254 <!-- ===== -->
255
256 <!-- ELEMENT          MIN  CONTENT          EXPLANATIONS  -->
257 <!ELEMENT formula     - - (%p.trans;|%m.math;)* -->
258              -- in-line formula -->
259 <!ELEMENT dformula    - - (%p.trans;|%m.math;)* -->
260              -- display formula -->
261 <!ELEMENT dformgrp    - - (formula|dformula)+  -->
262              -- display-formula group -->
263
264 <!-- ELEMENT  NAME    VALUE      DEFAULT      -->
265 <!ATTLIST formula  id      ID          #IMPLIED    -->
266              alphabet %a.types; -->
267              %SDAPREF;  "<?SDATRANS>Inline formula" -->
268              %SDASUSP;  "SUSPEND"  -->
269 >
270 <!ATTLIST dformula  id      ID          #IMPLIED    -->
271              num      CDATA      #IMPLIED    -->
272              align    (left|center|right) -->
273              center   -->
274              alphabet %a.types; -->
275              %SDAPREF;  "<?SDATRANS>Display formula" -->
276              %SDASUSP;  "SUSPEND"  -->
277 >
278 <!ATTLIST dformgrp  id      ID          #IMPLIED    -->
279              num      CDATA      #IMPLIED    -->
280              align    (left|center|right) -->
281              center   -->
282              %SDAPREF;  "<?SDATRANS>Display formula group" -->
283
284 >

```

## Appendix G: Example of a conversion of the DocBook DTD to HTML3

### G.1 The original document marked up in the DocBook DTD

The listing below is part of the manual describing the DocBook DTD and is tagged according to that same DocBook DTD (V2.2.1).

```
<sect1><title>How to Get the DocBook DTD Online</title>

<para>
You can find the DocBook DTD and its documentation online in
the Davenport archive (<filename>/pub/davenport/docbook</filename>)
at <filename>ftp.ora.com</filename> (198.112.208.13).
</para>

<para>
This sample session shows how to retrieve the DTD and its documentation:

<screen>
<!-- could mark up the prompt in next line with computeroutput -->
<systemitem class="prompt">%</><userinput>ftp ftp.ora.com</>
<computeroutput>Connected to amber.ora.com.</>
<computeroutput>220 amber FTP server (Version wu-2.4(1) Fri Apr 15 14:14:30 EDT 1994) ready.</>
<computeroutput>Name (ftp.ora.com:terry): </><userinput>anonymous</>
<computeroutput>331 Guest login ok, send your complete e-mail address as password.</>
<computeroutput>Password: </><lineannotation>&larr; type e-mail address</>
<systemitem class="prompt">ftp></><userinput>cd pub/davenport/docbook</>
</screen>

The DocBook DTD and related ASCII files are in a file named
<filename>docbook.N.shar</>, where <emphasis>N</>
is the current revision number:

<screen>
<systemitem class="prompt">ftp></><userinput>get docbook.2.2.1.shar</>
</screen>

Most of these files also exist separately and may be ftp'd individually.
</para>

<para>
The <command>get</> command will put this ASCII shar file
on your system. You must later unpack it on your system:
<screen>
<userinput>sh docbook.2.2.1.shar</>
</screen>
</para>
```

### G.2 ESIS representation of the source document

The following is the ESIS representation of the same document produced by nsgmls.

AID IMPLIED	APAGENUM IMPLIED	(PARA
ALANG IMPLIED	(TITLE	-You can find the DocBook DTD
AREMAP IMPLIED	-How to Get the DocBook DTD	and its documentation \nline
AROLE IMPLIED	Online	in the Davenport archive \n(
AXREFLABEL IMPLIED	)TITLE	
ALABEL IMPLIED	AID IMPLIED	AID IMPLIED
ARENDERAS IMPLIED	ALANG IMPLIED	ALANG IMPLIED
(SECT1	AREMAP IMPLIED	AREMAP IMPLIED
AID IMPLIED	AROLE IMPLIED	AROLE IMPLIED
ALANG IMPLIED	AXREFLABEL IMPLIED	AXREFLABEL IMPLIED
AREMAP IMPLIED	AMOREINFO TOKEN NONE	AMOREINFO TOKEN NONE
AROLE IMPLIED	(FILENAME	
AXREFLABEL IMPLIED		

<pre> -/pub/davenport/docbook )FILENAME -) at \n AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AMOREINFO TOKEN NONE (FILENAME -ftp.ora.com )FILENAME - (198.112.208.13). )PARA AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED (PARA -This sample session shows how to retrieve the DTD\nand its documentation:\n AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED sline ends and leading white space must be preserved in output NLINESPECIFIC AFORMAT NOTATION LINESPECIFIC AWIDTH IMPLIED (SCREEN AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AClass TOKEN PROMPT AMOREINFO TOKEN NONE )SYSTEMITEM -% )SYSTEMITEM AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AMOREINFO TOKEN NONE </pre>	<pre> (USERINPUT -ftp ftp.ora.com )USERINPUT -\n AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AMOREINFO TOKEN NONE (COMPUTEROUTPUT -Connected to amber.ora.com. )COMPUTEROUTPUT -\n AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AMOREINFO TOKEN NONE (COMPUTEROUTPUT -220 amber FTP server (Version wu-2.4(1) Fri Apr 15 14:14:30 EDT 1994) ready. )COMPUTEROUTPUT -\n AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AMOREINFO TOKEN NONE (COMPUTEROUTPUT -Name (ftp.ora.com:terry): )COMPUTEROUTPUT AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AMOREINFO TOKEN NONE (USERINPUT -anonymous )USERINPUT -\n AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED </pre>	<pre> AMOREINFO TOKEN NONE (COMPUTEROUTPUT -331 Guest login ok, send your complete e-mail address as password. )COMPUTEROUTPUT -\n AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AMOREINFO TOKEN NONE (COMPUTEROUTPUT -Password: )COMPUTEROUTPUT AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED (LINEANNOTATION -\ [larr ]\ type e-mail address )LINEANNOTATION -\n AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AClass TOKEN PROMPT AMOREINFO TOKEN NONE )SYSTEMITEM -ftp\ [gt ]\  )SYSTEMITEM AID IMPLIED ALANG IMPLIED AREMAP IMPLIED AROLE IMPLIED AXREFLABEL IMPLIED AMOREINFO TOKEN NONE (USERINPUT -cd pub/davenport/docbook )USERINPUT )SCREEN -\nThe DocBook DTD and related ASCII files are in\na file named </pre>
---	--	---

AID IMPLIED	AXREFLABEL IMPLIED	AROLE IMPLIED
ALANG IMPLIED	AClass TOKEN PROMPT	AXREFLABEL IMPLIED
AREMAP IMPLIED	AMOREINFO TOKEN NONE	AMOREINFO TOKEN NONE
AROLE IMPLIED	(SYSTEMITEM	(COMMAND
AXREFLABEL IMPLIED	-ftp\ gt ]\	-get
AMOREINFO TOKEN NONE	)SYSTEMITEM	)COMMAND
(FILENAME	AID IMPLIED	- command will put this ASCII
-docbook.N.shar	ALANG IMPLIED	shar \nfile on your system.
)FILENAME	AREMAP IMPLIED	You must later unpack \nit on
-, where	AROLE IMPLIED	your system:\n
AID IMPLIED	AXREFLABEL IMPLIED	AID IMPLIED
ALANG IMPLIED	AMOREINFO TOKEN NONE	ALANG IMPLIED
AREMAP IMPLIED	(USERINPUT	AREMAP IMPLIED
AROLE IMPLIED	-get docbook.2.2.1.shar	AROLE IMPLIED
AXREFLABEL IMPLIED	)USERINPUT	AXREFLABEL IMPLIED
(EMPHASIS	)SCREEN	AFORMAT NOTATION LINESPECIFIC
-N	-\nMost of these files\nalso	AWIDTH IMPLIED
)EMPHASIS	exist separately and may be	(SCREEN
-\nis the current revision	ftp'd individually.	AID IMPLIED
number:\n	)PARA	ALANG IMPLIED
AID IMPLIED	AID IMPLIED	AREMAP IMPLIED
ALANG IMPLIED	ALANG IMPLIED	AROLE IMPLIED
AREMAP IMPLIED	AREMAP IMPLIED	AXREFLABEL IMPLIED
AROLE IMPLIED	AROLE IMPLIED	AMOREINFO TOKEN NONE
AXREFLABEL IMPLIED	AXREFLABEL IMPLIED	(USERINPUT
AFORMAT NOTATION LINESPECIFIC	(PARA	-sh docbook.2.2.1.shar
AWIDTH IMPLIED	-The	)USERINPUT
(SCREEN	AID IMPLIED	)SCREEN
AID IMPLIED	ALANG IMPLIED	)PARA
ALANG IMPLIED	AREMAP IMPLIED	
AREMAP IMPLIED	AROLE IMPLIED	
AROLE IMPLIED		

### G.3 HTML3 output

The following presents the final HTML3 output resulting from the translation process.

```

<HTML>
<HEAD>
<TITLE>How to Get the DocBook DTD Online</TITLE>
</HEAD>
<BODY>
<H1>How to Get the DocBook DTD Online</H1>
You can find the DocBook DTD and its documentation online in the
Davenport archive (/pub/davenport/docbook) at ftp.ora.com
(198.112.208.13).<P>This sample session shows how to retrieve
the DTD and its documentation:
<pre>
%<i>ftp ftp.ora.com</i>
Connected to amber.ora.com.
220 amber FTP server (Version wu-2.4(1) Fri Apr 15 14:14:30 EDT 1994) ready.
Name (ftp.ora.com:terry): <i>anonymous</i>
331 Guest login ok, send your complete e-mail address as password.
Password: type e-mail address
ftp>&t;<i>cd pub/davenport/docbook</i>
</pre>
The DocBook DTD and related ASCII files are in a file named docbook.N.shar,
where <STRONG>N</STRONG> is the current revision number:
<pre>
ftp>&t;<i>get docbook.2.2.1.shar</i>

```

```
</pre>
Most of these files also exist separately and may be ftp'd individually.
<P>
The get command will put this ASCII shar file on your system.
You must later unpack it on your system:
<pre>
<i>sh docbook.2.2.1.shar</i>
</pre>
</BODY>
</HTML>
```



# From L<sup>A</sup>T<sub>E</sub>X to HTML and back

Michel Goossens and Janne Saarela

CERN, CN Division  
CH-1211 Geneva 23  
Switzerland  
goossens@cern.ch, saarela@cern.ch

## Abstract

Both L<sup>A</sup>T<sub>E</sub>X and HTML are languages that can express the structure of a document, and similarities between these two systems are shown. A detailed study is made of the L<sup>A</sup>-T<sub>E</sub>X2HTML program, written by Nikos Drakos, that is today the most complete utility for translating L<sup>A</sup>T<sub>E</sub>X code into HTML, providing a quasi-automatic translation for most elements. A discussion of a few other tools for translating between HTML and L<sup>A</sup>T<sub>E</sub>X concludes the article.

## 1 Similarities between L<sup>A</sup>T<sub>E</sub>X and HTML

HTML and L<sup>A</sup>T<sub>E</sub>X are both generic markup systems, and a comparison between tags for structural elements in both cases is shown in Table 1. In most cases the differences are trivial, seeming to indicate that, at first approximation, translating between these two systems should not prove too difficult.

The translation programs described in this article use these similarities, but in order to exploit the richness of the L<sup>A</sup>T<sub>E</sub>X language as compared to HTML (especially HTML2, which has no support for tables or mathematics), an *ad hoc* approach has to be adopted. To handle correctly L<sup>A</sup>T<sub>E</sub>X commands that have no equivalent in HTML, such elements can either be transformed into bitmap or PostScript pictures (an approach taken by L<sup>A</sup>T<sub>E</sub>X2HTML), or the user can specify how the given element should be handled in the target language.

Description	HTML	L <sup>A</sup> T <sub>E</sub> X
<b>Sectioning commands</b>		
level 1	<H1>	\chapter or \section
level 2	<H2>	\section or \subsection
level 3	<H3>	\subsection or \subsubsection
level 4	<H4>	\subsubsection or \paragraph
level 5	<H5>	\paragraph or \subparagraph
level 6	<H5>	\subparagraph
new paragraph	<P>	\par
<b>Lists</b>		
numbered list	<OL>	\begin{enumerate}
unnumbered list	<UL>	\begin{itemize}
list element	<LI>	\item
description list	<DL>	\begin{description}
term	<DT>	\item
definition	<DD>	<i>text</i>
<b>Highlighting text</b>		
emphasis	<EM>text</EM>	\emph{text}
italic	<I>text</I>	\textit{text}
bold	<B>text</B>	\textbf{text}
fixed with	<TT>text</TT>	\texttt{text}

Table 1: Comparison of structural elements in HTML and L<sup>A</sup>T<sub>E</sub>X

## 2 Converting L<sup>A</sup>T<sub>E</sub>X into HTML

Before discussing the `LaTeX2HTML` program, we want to mention a few other programs. First there is `l2x`<sup>1</sup>, written by Henning Schulzrinne (Berlin, Germany), which translates L<sup>A</sup>T<sub>E</sub>X into various other formats. This program is written in C and calls a Tcl function [4] for each L<sup>A</sup>T<sub>E</sub>X command.

A converter `html.tcl` is available for translating L<sup>A</sup>T<sub>E</sub>X files into HTML, by writing, for instance:

```
l2x -p html.tcl article.tex
```

Presently, only a sub-set of all L<sup>A</sup>T<sub>E</sub>X commands are handled (no mathematical formulae, tables, verbatim texts, etc.), yet it is not too difficult to augment the code of the converter `html.tcl` by introducing new Tcl commands.

1. See the URL <http://info.cern.ch/hypertext/WWW/Tools/l2x.html>.

Schwarzkopf has developed `Hyperlatex`<sup>2</sup>, a package written in the GNU Emacs Lisp language to translate documents marked up in (a subset of) L<sup>A</sup>T<sub>E</sub>X into HTML.

Another interesting tool is `tex2RTF`<sup>3</sup>, a utility to convert from L<sup>A</sup>T<sub>E</sub>X to four other formats, including HTML. It does a relatively good job for a sub-set of L<sup>A</sup>T<sub>E</sub>X commands, but, as with the `tc1` approach of `l2x`, it cannot handle more complex structures, such as mathematical expressions and tables.

Finally, although not directly relevant to L<sup>A</sup>T<sub>E</sub>X, `texihtml`<sup>4</sup> translates `texinfo` sources<sup>5</sup> into HTML.

### 3 The L<sup>A</sup>T<sub>E</sub>X2HTML converter – Generalities

L<sup>A</sup>T<sub>E</sub>X2HTML is a program written in the `perl` programming language<sup>6</sup> [7, 8, 9] by Nikos Drakos.<sup>7</sup> It transforms a L<sup>A</sup>T<sub>E</sub>X document into a series of HTML files linked in a way that reflects the structure of the original document.

#### 3.1 What L<sup>A</sup>T<sub>E</sub>X2HTML is and What it is Not

L<sup>A</sup>T<sub>E</sub>X2HTML is a conversion tool that allows documents written in L<sup>A</sup>T<sub>E</sub>X to become part of the World Wide Web. In addition, it offers an easy migration path towards authoring complex hypermedia documents using familiar word-processing concepts.

L<sup>A</sup>T<sub>E</sub>X2HTML replicates the basic structure of a L<sup>A</sup>T<sub>E</sub>X document as a set of interconnected HTML files which can be explored using automatically generated navigation panels. The cross-references, citations, footnotes, the table of contents and the lists of figures and tables are also translated into hypertext links. Formatting information which has equivalent “tags” in HTML (lists, quotes, paragraph breaks, type styles, etc.) is also converted appropriately. The remaining heavily formatted items such as mathematical equations, pictures or tables are converted to images placed automatically at the correct positions in the final HTML document.

L<sup>A</sup>T<sub>E</sub>X2HTML extends L<sup>A</sup>T<sub>E</sub>X by supporting arbitrary hypertext links and symbolic cross-references between evolving remote documents. It also allows the specification

---

2. The documentation is available at the URL <http://www.cs.ruu.nl/people/otfried/html/Hyperlatex/hyperlatex.html>. Otfried Schwarzkopf, who works at the University of Utrecht, can be reached via email at [otfried@cs.ruu.nl](mailto:otfried@cs.ruu.nl).

3. Written by Julian Smart (Edinburgh, Great Britain). For more information see the URL <http://www.aiai.ed.ac.uk/~jacs/tex2rtf.html>.

4. Written in `perl` by Lionel Cons (CERN, Geneva). For more information see the URL <http://asis01.cern.ch/infohtml/texi2html.html>.

5. `texinfo` is a T<sub>E</sub>X based markup language used for all gnu project related documentation.

6. More information can be found in the UF/NA `perl` archive at the URL <http://www.cis.ufl.edu/perl/>.

7. The documentation is at the URL <http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html>. One can also join the L<sup>A</sup>T<sub>E</sub>X2HTML mailing list by sending a message to [latex2html-request@mcs.anl.gov](mailto:latex2html-request@mcs.anl.gov) with as only contents line: `subscribe`.

of *conditional text* and the inclusion of raw HTML commands. These hypermedia extensions to L<sup>A</sup>T<sub>E</sub>X are available as new commands and environments from within a L<sup>A</sup>T<sub>E</sub>X document.

### 3.2 Overview

The main characteristics of the L<sup>A</sup>T<sub>E</sub>X2HTML translator are summarized in this section.

- a document is broken into one or more components as specified by the user;
- optional, customizable navigation panels can be added to each generated page to link other parts of the document, or external documents;
- inline and displayed equations are handled as images;
- tables and figures, and all other arbitrary environments are passed on to L<sup>A</sup>T<sub>E</sub>X and included as images; these images are included inline or made available via hypertext links;
- figures or tables can be arbitrarily scaled and shown either as inlined images or “thumbnails”;
- output can be generated to cope with the possibilities of various kind of browsers (for example, line browsers);
- definitions of new commands, environments, and theorems are given even when they are in external files;
- footnotes, tables of contents, lists of figures and tables, bibliographies, and the index are handled correctly;
- L<sup>A</sup>T<sub>E</sub>X cross-references and citations are transformed into hyperlinks, which work just as well inside a (sub)document as between several documents (located anywhere on the Internet);
- most L<sup>A</sup>T<sub>E</sub>X accented national characters are translated into their ISO-Latin-1 equivalent;
- hypertext links to arbitrary Internet services are recognized;
- programs running arbitrary scripts can be invoked (at the L<sup>A</sup>T<sub>E</sub>X level);
- a conditional text mechanism allows material to be included in the HTML or printed (dvi) versions only;
- similarly raw HTML material can be present in the L<sup>A</sup>T<sub>E</sub>X document (such as for specifying interactive forms);
- *common* L<sup>A</sup>T<sub>E</sub>X commands (i.e., those defined in the L<sup>A</sup>T<sub>E</sub>X Reference manual [3]) are handled gracefully;
- the user can define (in `per1`) functions to translate (un)known L<sup>A</sup>T<sub>E</sub>X commands in a customized way.

### 3.3 Using L<sup>A</sup>T<sub>E</sub>X2HTML

To use L<sup>A</sup>T<sub>E</sub>X2HTML, simply type

```
latex2html options-list file.tex
```

By default a new directory “file” will be created to contain the generated HTML files, some log files and possibly some images.

The output from LaTeX2HTML can be customized using a number of command line options, as described below.

The command line options *options-list* allow one to change the default behavior of LaTeX2HTML. Alternatively, the corresponding perl variables in the initialization file `.latex2html-init` may be changed, in order to achieve the same result (see Section 3.5).

`-split num` The default is 8.

Stop splitting sections into separate files at this depth. A value of 0 will put the document into a single HTML file.

`-link num` The default is 4.

Stop revealing child nodes at each node at this depth. (A node is characterized by the sequence part-chapter-section-subsection-...). A value of 0 will show no links to child nodes, a value of 1 will show only the immediate descendents, etc. A value at least as big as that of the `-split` option will produce a table of contents for the tree structure, rooted at each given node.

`-external_images`

Instead of including any generated images inside the document, leave them outside the document and provide hypertext links to them.

`-ascii_mode`

Use only ASCII characters and do not include any images in the final output. In ASCII mode, the output of the translator can be used on character-based browsers that do not support inlined images (the `<IMG>` tag).

`-t top-page-title`

Use the string *top-page-title* for the title of the document.

`-dir output-dir`

Redirect the output to the *output-dir* directory.

`-no_subdir`

Place the generated HTML files in the current directory. By default another file directory is created (or reused).

`-ps_images`

Use links to external PostScript pictures rather than inlined GIF (Graphics Interchange Format) images.

`-address author-address`

The address *author-address* will be used to sign each page.

`-no_navigation`

Do not put navigation links in each page.

`-top_navigation`

Put navigation links at the top of each page.

- bottom\_navigation  
Put navigation links at the bottom of each page *as well* as at the top.
- auto\_navigation  
Put navigation links at the top of each page. If the page has more words than `$WORDS_IN_PAGE` (the default is 450) then put one at the bottom of the page also.
- index\_in\_navigation  
When an index exists, put a link to the index page in the navigation panel.
- contents\_in\_navigation  
When a table of contents exists, put a link to that table in the navigation panel.
- next\_page\_in\_navigation  
Put a link to the next logical page in the navigation panel.
- previous\_page\_in\_navigation  
Put a link to the previous logical page in the navigation panel.
- info *string*  
Generate a new section *About this document ...* containing information about the document being translated. The default is for generating such a section with information on the original document, the date, the user and the translator. If *string* is empty (or has the value 0), this section is not created. If *string* is non-empty, it will replace the default information in the contents of the *About this document ...* section.
- dont\_include *file1 file2 ...*  
Do not include the specified file(s) *file1*, *file2*, etc. Such files can be package files that contain raw  $\TeX$  commands that the translator cannot handle.
- reuse  
Images generated during a previous translation process should be reused as far as possible. This option disables the initial interactive session where the user is asked whether to reuse the old directory, delete its contents or quit. Images which depend on context (for example, numbered tables or equations) cannot be reused and are always regenerated.
- no\_reuse  
Do *not* reuse images generated during a previous translation. This enables the initial interactive session during which the user is asked whether to reuse the old directory, delete its contents or quit.
- init\_file *file*  
Load the perl initialization script *file*. It will be loaded after the file (if it exists) `$HOME/.latex2html-init`. It can be used to change default options.
- no\_images  
Do not produce inlined images. If needed, the missing images can be generated "off-line" by restarting  $\text{LaTeX2HTML}$  with the `-images_only` option.

**-images\_only**

Try and convert any inlined images that were left over from previous runs of LaTeX2HTML. The advantage of using the latter two options is that the translation can be allowed to finish even when there are problems with image conversion. In addition, it may be possible to fix manually any image conversion problems and then run LaTeX2HTML again just to integrate the new images without having to translate the rest of the text.

**-show\_section\_numbers**

Instruct LaTeX2HTML to show section numbers. By default, section numbers are not shown, in order to allow individual sections to be used as stand-alone documents.

**-h** Print the list of options.

**3.4 Simple use of LaTeX2HTML**

To show the procedure for translating a L<sup>A</sup>T<sub>E</sub>X document into HTML, let us first look at a simple example, namely the file shown in Figure 1.<sup>8</sup> After running this file through L<sup>A</sup>T<sub>E</sub>X (twice, to resolve the cross-references) one obtains the output shown in Figure 2.

This same L<sup>A</sup>T<sub>E</sub>X source document is now run through LaTeX2HTML with the command

```
> latex2html -init_file french.pl babel.tex
```

where the default options have been used apart from the fact that we want titles in French. That is why we use the option `-init_file` to load the file `french.pl`, which merely contains

```
$TITLES_LANGUAGE = "french";
1;
```

as explained in Section 3.9.

The log messages generated by LaTeX2HTML are shown below.

```
This is LaTeX2HTML Version 95.1 (Fri Jan 20 1995)
  by Nikos Drakos,
  Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/usr/g/goossens/babel.tex

Loading /usr/local/lib/latex2html/styles/makeidx.perl
....
Reading ...
Processing macros ....+.....
Reading babel.aux .....
Translating ...0/8.....1/8....2/8....3/8
....4/8.....5/8.....6/8.....
```

---

8. This one-page example is chosen because it is discussed in detail in Chapter 9 of [1] and at the same time shows how LaTeX2HTML handles non-English documents.

```

\documentclass{article}
\usepackage{makeidx}
\usepackage[dvips]{graphicx}
\usepackage[french]{babel}
\makeindex
\begin{document}
\begin{center}\Large
  Exemple d'un article en fran\c{c}ais\l[2mm]\today
\end{center}
\tableofcontents
\listoffigures
\listoftables
\section{Une figure EPS}
\index{section}
Cette section montre comment inclure une figure
PostScript\cite{bib-PS} dans un document \LaTeX. La
figure\ref{Fpsfig} est ins\er\ee dans le texte \a
l'aide de la commande \verb!\includegraphics{colorcir.eps}!.
\index{figure}\index{PostScript}

\begin{figure}
\centering
\begin{tabular}{c@{\quad}c}
\includegraphics[width=3cm]{colorcir.eps} &
\includegraphics[width=3cm]{tac2dim.eps} \\
\end{tabular}
\caption{Deux images EPS}\label{Fpsfig}
\end{figure}
\section{Exemple d'un tableau}
Le tableau\ref{tab:exa} \a la page \pageref{tab:exa}
montre l'utilisation de l'environnement \texttt{table}.
\begin{table}
\centering
\begin{tabular}{cccccc}
\Lcs{primo} \primo & \Lcs{secundo} \secundo & \Lcs{tertio} \tertio &
\Lcs{quatro} \quatro & 2\Lcs{ieme} \ 2\ieme \\
\end{tabular}
\caption{Quelques commandes de l'option \texttt{french}
de \texttt{babel}}\label{tab:exa}\index{tableau}
\end{table}
\begin{thebibliography}{99}
\index{r\ef\erences}
\bibitem{bib-PS}
Adobe Inc. \emph{PostScript, manuel de r\ef\erence
(2i\eme \ 'edition)} Inter\ 'Editions (France), 1992
\end{thebibliography}
\printindex
\index{index}
\end{document}

```

Figure 1: Example of a L<sup>A</sup>T<sub>E</sub>X document

Exemple d'un article en français  
7 décembre 1994

### Table des matières

<b>1</b>	Une figure EPS	<b>1</b>
<b>2</b>	Exemple d'un tableau	<b>1</b>

### Liste des figures

1	Deux images EPS	1
---	-----------------	---

### Liste des tableaux

1	Quelques commandes de l'option french de babel	1
---	--	---

## 1 Une figure EPS

Cette section montre comment inclure une figure PostScript[1] dans un document L<sup>A</sup>T<sub>E</sub>X. La figure 1 est insérée dans le texte à l'aide de la commande `\includegraphics{colorcir.eps}`.

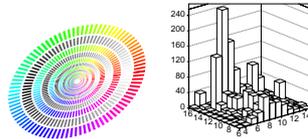


Figure 1: Deux images EPS

## 2 Exemple d'un tableau

Le tableau 1 à la page 1 montre l'utilisation de l'environnement `table`.

```
\primo 1° \secundo 2° \tertio 3° \quatro 4° 2\ieme 2°
```

Tableau 1: Quelques commandes de l'option french de babel

### Références

[1] Adobe Inc. *PostScript, manuel de référence (2ième édition)* InterÉditions (France), 1992

### Index

figure, 1	PostScript, 1	section, 1
index, 1	références, 1	tableau, 1

```

7/8.....8/8.....
Writing image file ...

This is TeX, Version 3.1415 (C version 6.1)
(images.tex
LaTeX2e <1994/12/01>

Generating postscript images using dvips ...
This is dvipsk 5.58e Copyright 1986, 1994
      Radical Eye Software
' TeX output 1995.05.11:0844' -> 14024_image
(-> 14024_image001) <tex.pro><special.pro>
[1<colorcir.eps><tac2dim.eps>]
(-> 14024_image002) <tex.pro><special.pro>[2]
Writing 14024_image002.ppm
Writing img2.gif
Writing 14024_image001.ppm
Writing img1.gif

Doing section links .....
Doing table of contents .....
Doing the index ....
Done.

```

The results are shown in Figure 3. The main document is shown in the middle at the top. Numbered arrows indicate the secondary documents that are produced and to which point in the main document they are linked. The document also contains a table of contents that is not shown explicitly, since its contents are almost identical to that of the main document. Note the navigation buttons at the top of each “page”. This navigation panel corresponds to the (default) option “-top\_navigation”. The navigation panel contains five push buttons:

- Next** to go to the *next* document,  
default option `-next_page_in_navigation`;
- Up** to go *up* one level;
- Previous** to move to the *previous* document,  
default option `-previous_page_in_navigation`;
- Contents** to jump directly to *Table of Contents*,  
default option `-contents_in_navigation`;
- Index** to jump straight to the *Index*,  
default option `-index_in_navigation`.

Each of the default values can be modified by redefining the corresponding perl variables in the initialization file `.latex2html.init`, as described in Section 3.5.

A detailed explanation of the meaning of the various numbers in Figure 3 is given below.

- ❶ the list of figures, containing a hyperlink pointing to document ❸ (containing the figure in question);
- ❷ the list of tables, containing a hyperlink pointing to document ❹ (containing the table in question);
- ❸ the first section, containing some text, a figure, and a hyperlink ([1]) pointing to an entry in the bibliography (document ❺);
- ❹ the second section, also containing some text and a table;
- ❺ the bibliographic references;
- ❻ the index, containing keywords that provide hyperlinks pointing to entry points in the various documents;
- ❼ an explanatory note detailing the procedure by which the document was translated into HTML. This text can be customized with the help of the option `-desc` (see Section 3.6).

### 3.5 Extending and customizing the translator

As the translator only partially covers the set of L<sup>A</sup>T<sub>E</sub>X commands and because new L<sup>A</sup>T<sub>E</sub>X commands can be defined arbitrarily using low level T<sub>E</sub>X commands, the translator should be flexible enough to allow end users to specify how they want particular commands to be translated.

#### *Adding support for packages*

LaTeX2HTML provides a mechanism to automatically load files containing code to translate specific packages. For instance, when in a L<sup>A</sup>T<sub>E</sub>X document, the command `\includegraphics{xxxx}` is found, a file called `LATEX2HTMLDIR/styles/xxxx.perl` is looked for. If such a file exists, it will be loaded into the main script.

This mechanism helps keep the core script smaller and modular and also makes it easier for others to contribute perl code to translate specific packages. The current distribution includes the files `german.perl`, `french.perl`, `makeidx.perl`, and for the hypertext extensions `html.perl`. Note, however, that writing such extensions requires an understanding of perl and of the way LaTeX2HTML is organized. Some more details will be given in Appendix C.

Presently, the user can ask that particular commands and their arguments be ignored or passed on to L<sup>A</sup>T<sub>E</sub>X for processing (the default behavior for unrecognized commands is for their arguments remains in the HTML text). Commands passed to L<sup>A</sup>T<sub>E</sub>X are converted to images that are either “inlined” in the main document or are accessible via hypertext links. Simple extensions using the commands below may be included in the system initialization file `LATEX2HTMLDIR/latextohtml.config`, or in the customization initialization file `.latex2html-init` in the user’s home directory or in the directory where the files to be converted reside.

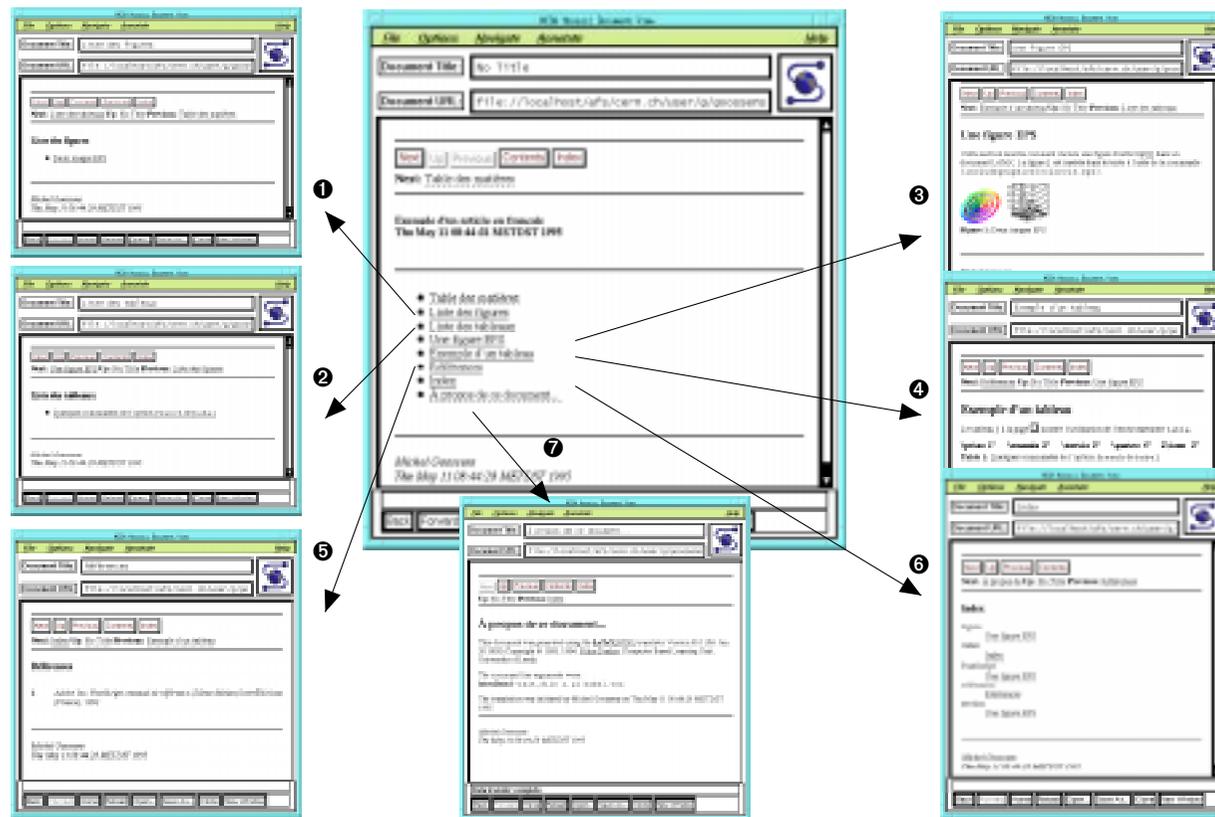


Figure 3: The HTML structure generated from the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  source of Figure 1 as visualized with the Mosaic browser

*Directing the translator to ignore commands*

Commands that should be ignored may be specified in the `.latex2html-init` file as input to the `ignore_commands` subroutine. Each command which is to be ignored should be on a separate line followed by compulsory or optional argument markers separated by #'s, for example:<sup>9</sup>

```
<cmd_name>#{ }# [ ]# { }# [ ] ...
```

{}'s mark compulsory arguments and []'s optional ones.

Some commands may have arguments which should be left as text, even though the command should be ignored (`\mbox`, `\center`, etc.). In these cases the arguments should be left unspecified.

Here is an example of how this mechanism may be used:

```
&ignore_commands( <<_IGNORED_CMDS_>);
documentclass # [ ] # { }
linebreak# [ ]
pagebreak# [ ]
center
<add your commands here>
_IGNORED_CMDS_
```

*Asking the translator to pass commands to L<sup>A</sup>T<sub>E</sub>X*

Commands that should be passed on to L<sup>A</sup>T<sub>E</sub>X for processing because there is no direct translation to HTML may be specified in the `.latex2html-init` file as input to the `process_commands_in_tex` subroutine. The format is the same as that for specifying commands to be ignored. Here is an example:

```
&process_commands_in_tex ( <<_RAW_ARG_CMDS_>);
fbox # { }
framebox # [ ] # [ ] # { }
<add your commands here>
_RAW_ARG_CMDS_
```

*Customizing L<sup>A</sup>T<sub>E</sub>X2HTML*

Besides honoring the options specified on the command line, L<sup>A</sup>T<sub>E</sub>X2HTML reads two standard files that can be used to customize its behavior. The first file, `latextohtml.config`, is a system-wide file (usually in the directory `/usr/local/lib/latex2html`). It contains the definitions for a complete installation, i.e., those common for all users, and specifies where certain external utility programs needed by L<sup>A</sup>T<sub>E</sub>X2HTML are to be found on the system (such as L<sup>A</sup>T<sub>E</sub>X, `dvips`, `gs`, `pmbplus`). Moreover, in this file important `perl` variables are initialized to their default values. At the end of the file one has the

---

9. It is possible to add arbitrary `perl` code between any of the argument markers that will be executed when the command is processed. For this, however, a basic understanding of how the translator works and, of course, `perl` is required.

possibility of specifying those L<sup>A</sup>T<sub>E</sub>X commands or environments that should be ignored, and those that should be passed on to L<sup>A</sup>T<sub>E</sub>X to be transformed into images for inclusion in the HTML file.

The second file, `.latex2html-init`, allows the user to customize L<sup>A</sup>T<sub>E</sub>X2HTML on an individual level. L<sup>A</sup>T<sub>E</sub>X2HTML will normally look for this file in the user's home directory (variable `$HOME` on Unix). This file can contain the same information as the global configuration file `latex2html.config` and is thus the ideal place to overwrite default values or to specify in the `perl` language how certain specific L<sup>A</sup>T<sub>E</sub>X commands should be handled. It should be noted that the L<sup>A</sup>T<sub>E</sub>X2HTML distribution L<sup>A</sup>T<sub>E</sub>X2HTML already contains a few files with definitions for translations of supplementary L<sup>A</sup>T<sub>E</sub>X commands introduced by certain extension packages, such as `german.perl`, `french.perl`, `html.perl` and `makeidx.perl`. To help the user, the distribution comes with an example file `dot.latex2html-init` that can serve as a model for writing one's own `.latex2html-init`.

#### *Creating a customization file .latex2html-init*

Before discussing examples of commands that can be put in the `.latex2html-init` customization file, it should be emphasized once more that this file, as well as all other files that are part of the L<sup>A</sup>T<sub>E</sub>X2HTML system, contain only `perl` instructions, and that one should thus have at least a basic understanding of this language before trying to edit any of these files.

Figures 4 and 5 show an example initialization file `dot.latex2html-init`. Its first parts initialize most of the `perl` variables used by the L<sup>A</sup>T<sub>E</sub>X2HTML system by setting them equal to their default values (as defined in the system-wide initialization file `latex2html.config`. Values that need not to be changed can be deleted from the file. When studying the various system variables, note the correspondence between the `perl` variables and the options of the `latex2html` described in Section 3.3).

#### *Examples*

We want to leave most of the values at their defaults as shown in Figures 4 and 5. However, we specify the format of the address fields explicitly and make a few more modifications; in particular, we do not want images to be included inside the HTML documents. Thus we should write something like:

```
$ADDRESS = "<I>Michel Goossens<BR>" .
           "CN Division<BR>"           .
           "Tel. 3363<BR>"           .
           "\n$address_data[1]</I>";
$MAX_SPLIT_DEPTH = 2; # stop at subsection
$MAX_LINK_DEPTH = 1; # child nodes to sections
$EXTERNAL_IMAGES = 1; # images outside document
# draw a nice rainbow-colored line (gif file)
```

```

#LaTeX2HTML Version 95.1 : dot.latex2html-init
#
### Command Line Argument Defaults #####

$MAX_SPLIT_DEPTH = 8;          # Stop making separate files at this depth
$MAX_LINK_DEPTH = 4;          # Stop showing child nodes at this depth
$NOLATEX = 0;                  # 1 = do not pass unknown environments to Latex
$EXTERNAL_IMAGES = 0;         # 1 = leave the images outside the document
$ASCII_MODE = 0;              # 1 = do not use any icons or internal images
$PS_IMAGES = 0;               # 1 = use links to external postscript
                                # images rather than inlined GIF's.
$TITLE = $default_title;      # The default is "No Title"
$DESTDIR = '.';               # Put the result in this directory
$NO_SUBDIR = 0;               # 0 = create (reuse) file subdirectory
                                # 1 = put generated HTML files in current dir.
# Supply your own string if you don't like the default <Name> <Date>
$ADDRESS = "<I>$address_data[0] <BR>\n$address_data[1]</I>";
$NO_NAVIGATION = 0;           # 1 = no navigation panel at top of each page
$AUTO_NAVIGATION = 1;         # 1 = put navigation links at top of page
$WORDS_IN_PAGE = 300;         # if nb. words on page > $WORDS_IN_PAGE put
                                # navigation panel at bottom of page.
$INDEX_IN_NAVIGATION = 1;     # put link to index page in navigation panel
$CONTENTS_IN_NAVIGATION = 1;  # put link to table of contents " " "
$NEXT_PAGE_IN_NAVIGATION = 1; # put link to next logical page " " "
$PREVIOUS_PAGE_IN_NAVIGATION = 1; # put link to prev. " " " " "
$INFO = 1;                    # 0 = do not make "About this document..." section
$REUSE = 1;                   # Reuse images generated during previous runs

# Do not try to translate these package files.
# Complex LaTeX packages may cause the translator to hang.
# If this occurs add the package's filename here.
$DONT_INCLUDE = "memo:psfig:pictex:revtex";

# When this is 1, the section numbers are shown. The section numbers should
# then match those that would have been produced by LaTeX.
# The correct section numbers are obtained from the $FILE.aux file generated
# by LaTeX.
# Hiding the section numbers encourages use of particular sections
# as standalone documents. In this case the cross reference to a section
# is shown using the default symbol rather than the section number.
$SHOW_SECTION_NUMBERS = 0;

### Other global variables #####
$CHILDLINE = "<BR> <HR>\n";

# This is the line width measured in pixels and it is used to right justify
# equations and equation arrays;
$LINE_WIDTH = 500;

# Affects ONLY the way accents are processed
$default_language = 'english';

# This number will determine the size of the equations, special characters,
# and anything which will be converted into an inlined image
# *except* "image generating environments" such as "figure", "table"
# or "minipage".
# Effective values are those greater than 0.
# Sensible values are between 0.1 - 4.
$MATH_SCALE_FACTOR = 1.6;

# This number will determine the size of
# image generating environments such as "figure", "table" or "minipage".
# Effective values are those greater than 0.
# Sensible values are between 0.1 - 4.
$FIGURE_SCALE_FACTOR = 0;

```

Figure 4: dot.latex2html-init file (part 1)

```

# If this is set then intermediate files are left for later inspection.
# This includes $$_images.tex and $$_images.log created during image
# conversion.
# Caution: Intermediate files can be *enormous*.
$DEBUG = 0;

# The value of this variable determines how many words to use in each
# title that is added to the navigation panel (see below)
#
$WORDS_IN_NAVIGATION_PANEL_TITLES = 4;

# If both of the following two variables are set then the "Up" button
# of the navigation panel in the first node/page of a converted document
# will point to $EXTERNAL_UP_LINK. $EXTERNAL_UP_TITLE should be set
# to some text which describes this external link.
$EXTERNAL_UP_LINK = "";
$EXTERNAL_UP_TITLE = "";

# If this is set then the resulting HTML will look marginally better if viewed
# with Netscape.
$NETSCAPE_HTML = 0;

# Valid paper sizes are "letter", "legal", "a4","a3","a2" and "a0"
# Paper sizes has no effect other than in the time it takes to create inlined
# images and in whether large images can be created at all ie
# - larger paper sizes *MAY* help with large image problems
# - smaller paper sizes are quicker to handle
$PAPERSIZE = "a4";

# Replace "english" with another language in order to tell LaTeX2HTML that you
# want some generated section titles (eg "Table of Contents" or "References")
# to appear in a different language. Currently only "english" and "french"
# is supported but it is very easy to add your own. See the example in the
# file "latex2html.config"
$TITLES_LANGUAGE = "english";

### Navigation Panel #####
# The navigation panel is constructed out of buttons and section titles.
# These can be configured in any combination with arbitrary text and
# HTML tags interspersed between them.
# The buttons available are:
# $PREVIOUS - points to the previous section
# $UP - points up to the "parent" section
# $NEXT - points to the next section
# $NEXT_GROUP - points to the next "group" section
# $PREVIOUS_GROUP - points to the previous "group" section
# $CONTENTS - points to the contents page if there is one
# $INDEX - points to the index page if there is one
#
# If the corresponding section exists the button will contain an
# active link to that section. If the corresponding section does
# not exist the button will be inactive.
#
# Also for each of the $PREVIOUS $UP $NEXT $NEXT_GROUP and $PREVIOUS_GROUP
# buttons there are equivalent $PREVIOUS_TITLE, $UP_TITLE, etc variables
# which contain the titles of their corresponding sections.
# Each title is empty if there is no corresponding section.
#
# The subroutine below constructs the navigation panel in each page.
# Feel free to mix and match buttons, titles, your own text, your logos,
# and arbitrary HTML (the "." is the Perl concatenation operator).
sub navigation_panel {...}
1; # This must be the last line

```

Figure 5: dot.latex2html-init file (part 2). The navigation panel perl code is shown in Figure 10.

```
# instead of the default simple line (<HR>)
$CHILDLINE = "<BR><IMG " .
              "SRC=rainbow_line.gif><BR>"
```

Normally, LaTeX2HTML will read all package and class files and interpret all the commands that are defined in those files. This can lead to problems, so it is wise to exclude some files. Also, one may want to define a translation into perl for the commands in one or more files, so they should also not be read. The list of files to be excluded, is specified as follows:

```
$DONT_INCLUDE = "memo:psfig:times:revtex:" .
                "aps:float:harvard:tabs";
```

Special symbols and inline equations are generally transformed into inlined (bitmap) images that are placed inside the HTML text on the same line when viewing the document with a browser. On the other hand, display environments, such as tables, figures, minipages, and multi-line equations are transformed into images that will also be shown on a line by themselves after starting a new paragraph. The scale factor for the two kinds of images (inline and displayed) can be specified by the following parameters:

```
$MATH_SCALE_FACTOR = 1.6;# inline images
$FIGURE_SCALE_FACTOR = 0;# display images
# = 0, original dimension
```

Finally, we specify – as described in Sections 3.5 and 3.5 – a list of commands to be ignored and passed to L<sup>A</sup>T<sub>E</sub>X.

```
### Commands to ignore #####
&ignore_commands( <<_IGNORED_CMDS_);
documentclass # [] # {}
usepackage # [] # {}
mbox
makebox# [] # []
_IGNORED_CMDS_
### Commands to pass on to LaTeX{} ##
&process_commands_in_tex (<<_RAW_ARG_CMDS_);
includegraphics # [] # [] # {}
rotatebox # {} # {}
_RAW_ARG_CMDS_

1; # This MUST be the last line
```

We notice that the mandatory argument of the `\mbox` and `\makebox` commands is not specified, so that it will end up in the text, while the optional arguments of the

`\makebox` command will disappear. In the case of the framed box commands `\fbox` and `\framebox`, both mandatory and optional arguments are passed on to  $\text{\LaTeX}$ .

It is important to note that the last line of the file *must* be the one shown in the example above.

### 3.6 Hypertext extensions

These commands are defined in the `html.sty` package file that is part of the distribution. They are defined as  $\text{\LaTeX}$  commands that are (mostly) ignored during the  $\text{\LaTeX}$  run but are activated in the HTML version. To use them the `html` package must be included with a `\usepackage` command.

#### *Hyperlinks in $\text{\LaTeX}$*

With the `\htmladdnormallink` and `\htmladding` commands one can build arbitrary hypertext references.

```
\htmladdnormallink{text}{\langle URL \rangle}
```

When processed by  $\text{\LaTeX}$  the URL part will be ignored, but `LaTeX2HTML` will transform it into an active hypertext link that can give access to sound, images, other documents, etc., for instance,

```
\htmladdnormallink{The  $\Omega$  Project}
{http://www.ens.fr/omega/}
```

```
\htmladdnormallinkfoot{text}{\langle URL \rangle}
```

This command takes the same two arguments and has the same effect when generating HTML as the command `\htmladdnormallink`, but when processed by  $\text{\LaTeX}$  it shows the URL as a footnote.

```
\htmladding{\langle URL \rangle}
```

In a similar way, the argument of the `\htmladding` command should be a URL pointing to an image. This URL is ignored in the  $\text{\LaTeX}$  hard copy output.

#### *Cross-references between living documents*

In this case we want to use a mechanism for establishing cross-references between two or more documents via symbolic labels independent of the physical realisation of these documents. The documents involved may reside in remote locations and may be spread across one or more HTML files.

The mechanism is an extension of the simple `\label`-`\ref` pairs that can be used only within a single document. A symbolic label defined with a `\label` command can be referred to using a `\ref` command. When processed by  $\text{\LaTeX}$ , each `\ref` command

is replaced by the section number at which the corresponding `\label` occurred. When processed by L<sup>A</sup>T<sub>E</sub>X2HTML each `\ref` is replaced by a hypertext link to the place where the corresponding `\label` occurred. The new commands, detailed below, show how it is possible to refer to symbolic labels defined with `\label` in other external documents. Such references to external symbolic labels are then translated into hyperlinks pointing to the external document.

Cross-references between documents are established with the commands:

```
\externallabels
  {\URL directory external document}
  {\external document labels.pl file}
\externalref{\label in remote document}
```

The first argument to `\externallabels` should be a URL to the directory containing the external document. The second argument should be the full pathname to the `labels.pl` file belonging to the external document. The file `labels.pl` associates symbolic labels with physical files and is generated automatically for each translated document. For *remote* external documents it is necessary to copy the `labels.pl` file locally so that it can be read when processing a local document that uses it. The command `\externallabels` should be used once for each external document in order to import the external labels into the current document. The argument to `\externalref` can be any symbolic label defined in any of the external documents in the same way that the `\ref` command refers to labels defined internally.

After modifications in an external document, such as addition or deletion of sectioning levels, or a segmentation into different physical parts, a new file, `labels.pl`, will be generated. If in another document the `\externallabels` command contains the correct address to the `labels.pl` file, then cross-references will be realigned correctly. A warning will be given if `labels.pl` cannot be found.

#### *Example of a composite document*

In this section we show how to handle a set of composite documents taking advantage of the hypertext extensions described in Section 3.6.

We start with the L<sup>A</sup>T<sub>E</sub>X source document shown in Figure 1 and divide it, for demonstration purposes, into four sub-documents, shown in Figure 6, namely a main file (`ex20.tex`) and three secondary files (`ex21.tex`, `ex22.tex` and `ex2bib.tex`). We must first run all these files through L<sup>A</sup>T<sub>E</sub>X and then *in the correct order* through L<sup>A</sup>T<sub>E</sub>X2HTML. Indeed, as we use cross-references to refer to document elements in external documents (with the commands `\externalref` and `\externallabels`) we should first treat the secondary files `ex21.tex`, `ex22.tex`, and `ex2bib.tex`, before tackling the master file `ex20.tex`.

By default, L<sup>A</sup>T<sub>E</sub>X2HTML writes the files that it creates into a subdirectory with the same name as the original file, for example, after execution of the command:

```

\documentclass{article}
\usepackage{html}
\usepackage[dvips]{graphicx}
\usepackage[french]{babel}
\begin{document}
\begin{center}
\Large Exemple d'un document compos\`e
\end{center}

\htmladdnormallink{Les Images}%
    {../ex21/ex21.html}
\externallabels{../ex21}%
    {../ex21/labels.pl}
R\`eference \`a une figure
externe~\externalref{Fpsfig}.

\htmladdnormallink{Les tableaux}%
    {../ex22/ex22.html}
\externallabels{../ex22}%
    {../ex22/labels.pl}
R\`eference \`a un tableau
externe~\externalref{tab-exa}.

\htmladdnormallink{La bibliographie}%
    {../ex2bib/ex2bib.html}
\end{document}

```

Master file (ex2.tex)

```

\documentclass{article}
\usepackage{html}
\usepackage[dvips]{graphicx}
\usepackage[french]{babel}
\makeindex
\begin{document}
\section{Une figure EPS}\label{sc-figure}
Cette section montre comment inclure une
\externallabels{../ex2bib}%
    {../ex2bib/labels.pl}%
figure PostScript\externalref{bibPS}
dans un document \LaTeX. La
\hyperref{figure}{figure }{}{Fpsfig}
est ins\`er\`ee dans le texte \`a l'aide
de la commande
\verb!\includegraphics{colorcir.eps}!.
\begin{figure}\centering
\htmlimage{thumbnail=0.2}
\begin{tabular}{c@{\quad}c}
\includegraphics[width=6cm]{colorcir.eps}&
\includegraphics[width=6cm]{tac2dim.eps}
\end{tabular}
\caption{Deux images EPS}\label{Fpsfig}
\end{figure}
\end{document}

```

File containing images (ex21.tex)

```

\documentclass{article}
\usepackage{html}
\usepackage[french]{babel}
\newcommand{\Lcs}[1]{%
    \texttt{\symbol{'134}#1}\enspace}
\begin{document}
\section{Exemple d'un tableau}
\label{sec-tableau}
Le \hyperref{tableau}{tableau }{}{tab-exa}
montre l'utilisation de l'environnement
\texttt{table}.
\begin{table}\centering
\begin{tabular}{ccccc}
\Lcs{primo} \primo & & & & \\
\Lcs{secundo} \secundo & & & & \\
\Lcs{tertio} \tertio & & & & \\
\Lcs{quatro} \quatro & & & & \\
2\Lcs{ieme} \ 2\ieme & & & & \\
\end{tabular}
\caption{Quelques commandes de l'option
\texttt{french} de
\texttt{babel}}\label{tab-exa}
\end{table}
\end{document}

```

File containing the table (ex22.tex)

```

\documentclass{article}
\usepackage{html}
\usepackage[french]{babel}
\makeindex
\begin{document}
\begin{thebibliography}{99}
\bibitem{bib-PS}\label{bibPS}
Adobe Inc. \emph{PostScript, manuel de
r\`eference (2i\`eme \`edition)}
Inter\`Editions (France), 1992
\end{thebibliography}
\end{document}

```

File with the bibliography (ex2bib.tex)

Figure 6: Example of a composite document (L<sup>A</sup>T<sub>E</sub>X files)

```
> latex2html ex20.tex
```

one ends up with a directory `ex20` containing all files associated with the translation of the input file `ex20.tex`. Figure 7 shows the structure of the four sub-directories created.

To guide LaTeX2HTML in translating these documents we also used a customization file, `myinit.pl`, containing some redefinitions of perl constants.

```
# File myinit.pl
# Customization of latex2html
$ADDRESS = "<I>Michel Goossens<BR>" .
           "Division CN<BR>" .
           "Tél. 3363<BR>" .
           "\n$address_data[1]</I>";
$MAX_SPLIT_DEPTH = 0; # do not split document
$MAX_LINK_DEPTH = 0; # no down links needed
$NO_NAVIGATION = 1;  # no navigation panel

1; # Mandatory last line
```

When executing LaTeX2HTML on the files we then issued the following command:

```
> latex2html -init_file myinit.pl \
>           -t "Image" \
>           -info "Test du 2/12/94" \
>           ex21.tex
```

Apart from loading our customization file `moninit.pl` (option `-init_file`), we also specify a title for the document (option `-t`), and add a description about the document (option `-info`). The result can be seen in the upper left corner of Figure 8.

Shown below are the informative messages generated by LaTeX2HTML when executing the above command. At first the file `html.perl` associated with the hypertext extensions described in Section 3.6 is loaded (thanks to the `\usepackage{html}` command as seen in the source in Figure 6). The auxiliary file `ex21.aux` is also read, thus reminding us that the documents should be treated by L<sup>A</sup>T<sub>E</sub>X before LaTeX2HTML is run. After reading the complete L<sup>A</sup>T<sub>E</sub>X input file, LaTeX2HTML generates the file `image.tex` which contains the L<sup>A</sup>T<sub>E</sub>X code corresponding to all environments for which no translation rules were available. After running L<sup>A</sup>T<sub>E</sub>X on `images.tex` the dvi file is transformed by the `dvips` program into PostScript. Then another program, `ghostview`, interprets this PostScript and transforms it into the GIF format (via an intermediate stage using the `ppm` format). It is these GIF images that are used by most browsers to show the images on screen. At the end, LaTeX2HTML reads the file(s) containing the labels of the external documents in order to resolve possible cross-references by including the necessary `<URL>` addresses.

This is LaTeX2HTML Version 0.6.4 (Tues Aug 30 1994)

```

Top directory (TeX source files)
=====
569 ex20.tex
721 ex21.tex
627 ex22.tex
322 ex2bib.tex

Subdirectories (generated HTML files)
=====
    ex20
    ----
1187 ex20.html
109  images.pl
 93  labels.pl

    ex21
    ----
1755 T_18854_figure15.gif
12118 _18854_figure15.gif
122  _18854_tex2html_wrap57.gif
1345 ex21.html
 539 images.pl
 589 images.tex
 190 labels.pl

    ex22
    ----
 624 _15561_table12.gif
1047 ex22.html
 512 images.pl
 687 images.tex
 191 labels.pl

    ex2bib
    -----
 844 ex2bib.html
 109  images.pl
 141  labels.pl

```

Note the presence of the files `labels.pl` that contain information associating the symbolic names used on the `\label` commands in the original `LATEX` source documents with the physical documents. The other files are one or more HTML files that were created by the translation process. GIF images are generated for all environments that `LATEX2HTML` cannot translate gracefully into HTML. In this case the relevant part of the `LATEX` code is first copied into a file `images.tex` that is run through `LATEX`, which places each such environment on a separate page, so that the `dvips` program can produce a PostScript picture for each that is then (in principle) translated into GIF by using the `Ghostscript` program (see Section A.1 for more information about all these programs)

Figure 7: Subdirectory structure after translation of the four documents shown in Figure 6

```

                by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/usr/goossens/html/ex21.tex

Loading /usr/local/lib/latex2html/styles/html.perl...

Reading ...
Reading ex21.aux .....
Translating ...0/2.....1/2.....2/2.....
Generating images using LaTeX ...
This is TeX, Version 3.1415 (C version 6.1)
(18854_images.tex
LaTeX2e <1994/06/01> patch level 3
Hyphenation patterns for english, loaded.

Generating postscript images using dvips ...
This is dvipsk 5.58c Copyright 1986, 1994
                Radical Eye Software
' TeX output 1994.12.02:1830' -> 18854_image
(-> 18854_image001) <tex.pro><special.pro>[1]
(-> 18854_image002) <tex.pro>
<special.pro>[2<colorcir.eps><tac2dim.eps>]
GS>GS>Writing 18854_image001.ppm
GS>Writing _18854_tex2html_wrap57.gif
GS>GS>Writing 18854_image002.ppm
GS>Writing _18854_figure15.gif
GS>GS>Writing 18854_image002.ppm
GS>Writing T_18854_figure15.gif

Doing section links ....
Done.

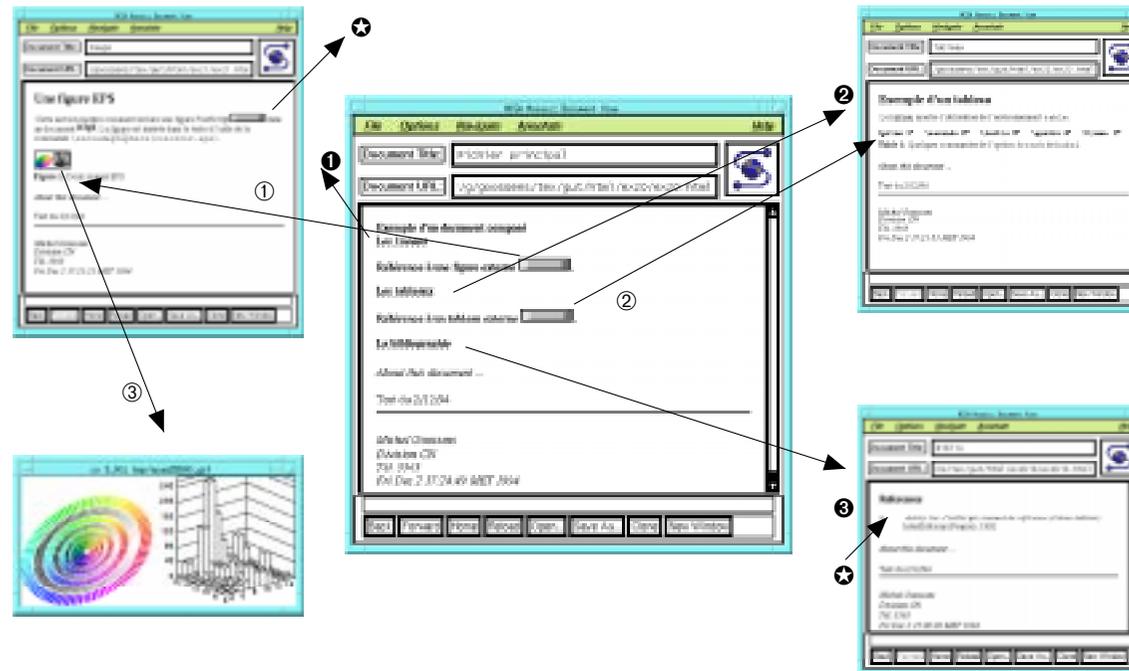
```

The result of all our efforts is shown in Figure 8.

### 3.7 Including arbitrary HTML markup

Raw HTML tags and text may be included using the `rawtext` environment. An interesting use of this feature is in the creation of interactive electronic forms. from within a L<sup>A</sup>T<sub>E</sub>X document. When producing the paper (DVI) version of a document the `rawhtml` environment is ignored.

Here is an example:



As requested, there are no navigation panels, the titles and the information part *About this document ...* have been customized as indicated in the file `myinit.pl`. The arrows carrying the numbers ①, ②, and ③ correspond to hyperlinks pointing to an HTML document using the `\htmladdnormallink` command in the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  source. The arrows numbered ① and ② are cross-references constructed with the commands `\externalref`, that make use of symbolic names specified as the argument of `\label` commands in the target documents. The arrow numbered ③ corresponds to a hyperlink connecting the thumbnail in the text with the real-size image available as a separate external `gif` file. Finally, the start and end points of the bibliographic reference link are indicated by the symbol  $\star$ .

Figure 8: The HTML file structure obtained from the composite document and its sub-documents (Figure 6) as viewed by the Mosaic browser.

```

\begin{rawhtml}
<HTML>
<HEAD>
<TITLE>Example of simple form</TITLE>
</HEAD>
<BODY>
<FORM
  ACTION="http://www.server.ch/form.cgi"
  METHOD="POST">
  Radio buttons:
<UL>
<LI> <INPUT TYPE="radio" NAME="mode"
      VALUE="FM"> Frequency modulation.
<LI> <INPUT TYPE="radio" NAME="mode"
      VALUE="SW" CHECKED> Short waves.
</UL>

</FORM>
</BODY>
</HTML>
\end{rawhtml}

```

The result of running this electronic form with Mosaic would yield Figure 9

### *Conditional text*

Conditional text can be specified using the environments `latexonly` and `htmlonly`. These allow the writing of parts of a document intended only for electronic delivery or only for paper-based delivery.

This would be useful in, for example, adding a long description of a multi-media resource to the paper version of a document. Such a description would be redundant in the electronic version, as the user can have direct access to this resource.

Using L<sup>A</sup>T<sub>E</sub>X commands involving counters (for example, numbered figures or equations) in conditional texts may cause problems with the values of the counters in the electronic version.

### *Cross-references shown as "hyperized" text*

In printed documents cross-references are shown by *numerical or symbolic indirection*, such as "see equation 3.1a" (numeric indirection), or "see chapter "Hypertext" (symbolic indirection). In a hypertext document, however, cross-references can be shown without any indirection by using highlighting of a relevant piece of text. This can contribute to making a document more readable by removing unnecessary visual information.

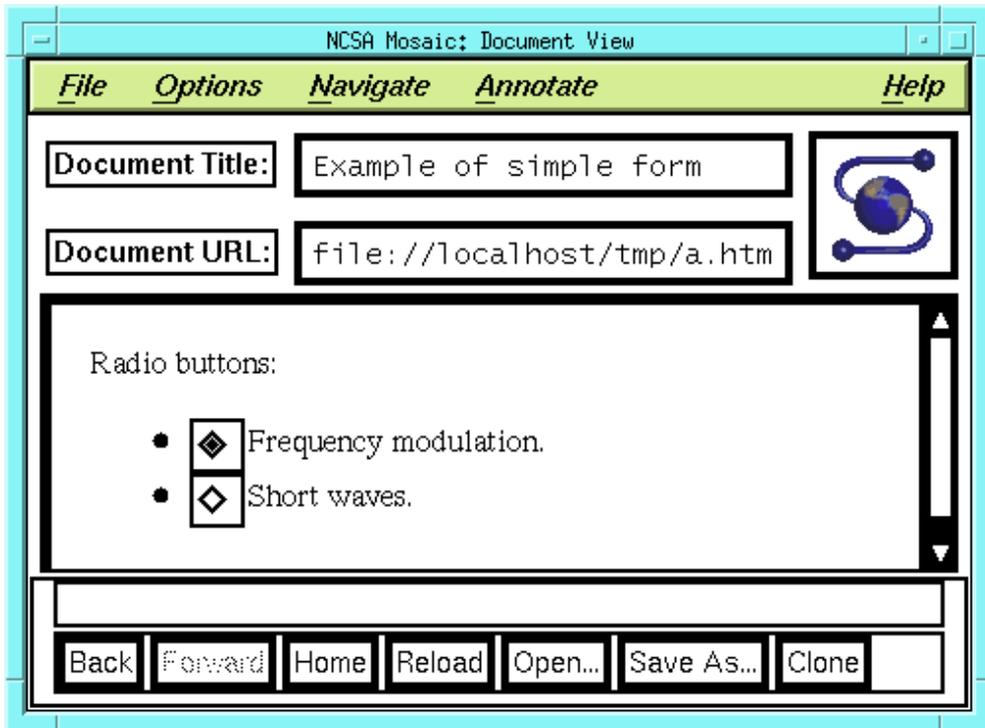


Figure 9: Including arbitrary HTML Markup

With LaTeX2HTML one can use the `\hyperref` command to specify how a cross-reference should appear in the printed and hypertext versions of a document.

```
\hyperref{text-h}{text-d1}{text-d2}{label}
```

The meaning of the four arguments is as follows:

- text-h* text to be highlighted in the hypertext version;
- text-d1* text to be shown in the printed version followed by a numeric reference;
- text-d2* text following the reference text;
- label* the label defining the cross-reference.

```
Example of the use of hyperref, with a
\hyperref
{reference to conditional text}
{reference to conditional text}
```

```
(see Section }
{ for more information)}
{sec:latexonly}
as an example.
```

Here is how it will be printed:

Example of the use of hyperref, with a reference to conditional text (see Section 3.7 for more information) as an example.

In the hypertext version what would appear is:

Example of the use of hyperref, with a [reference to conditional text](#) as an example.

A simpler version of the above command but having the same effect for the HTML version:

```
\htmlref{text}{label}
```

In the HTML version the text will be “hyperized” pointing to the label, while in the printed version the text will be shown as it is and the label ignored.

### *Customizing the navigation panel*

The navigation panel is the strip containing “buttons” and text that appears at the top and possibly at the bottom of each generated page and that provides hypertext links to other sections of a document. Some of the options and variables that control whether and where it should appear have already been mentioned in Section 3.3.

A simple mechanism for appending customized buttons to the navigation panel is provided by the command, `\htmladdtonavigation`. This takes one argument, which L<sup>A</sup>T<sub>E</sub>X2HTML appends to the navigation panel:

```
\htmladdtonavigation
{\htmladdnormallink
 {\htmladding{http://server/mybutton.gif}}
 {http://server/link}}
```

For example, the above will add an active button `mybutton.gif` pointing to the specified location.

It is also possible to completely specify what is to appear in the navigation panel and its order of appearance. As each section is processed, L<sup>A</sup>T<sub>E</sub>X2HTML assigns relevant information to a number of global variables. These variables are used by the subroutine `navigation_panel` where the navigation panel is constructed as a string consisting of these variables and some formatting information.

This subroutine can be redefined in the system and/or user configuration files `HOME/.latex2html-init` and `LATEX2HTMLDIR/latex2html.config`.

The list below contains the names of control panel variables that relate to navigation icons and explains where they point to.

CONTENTS	contents page (if it exists);
INDEX	index page (if it exists).
NEXT	next section;
PREVIOUS	previous section;
UP	“parent” section;
NEXT_GROUP	next “group” section;
PREVIOUS_GROUP	previous “group” section.

The list below contains the names of textual links that point to the title information associated with the control panel variables described above.

NEXT_TITLE	next section;
PREVIOUS_TITLE	previous section;
UP_TITLE	“parent” section;
NEXT_GROUP_TITLE	next “group” section;
PREVIOUS_GROUP_TITLE	previous “group” section.

If the corresponding section exists, each iconic button will contain an active link to that section. If the corresponding section does not exist, the button will be inactive. If the section corresponding to a textual link does not exist then the link will be empty.

The variable `WORDS_IN_NAVIGATION_PANEL_TITLES` controls the number of words in each textual link. It may be changed in the configuration files. Figure 10 shows an example of a navigation panel.

### 3.8 Image conversion

LaTeX2HTML converts equations, special accents, external PostScript files, and L<sup>A</sup>T<sub>E</sub>X environments it cannot directly translate into inlined images. It is possible to control the final appearance of such images, both for inline and display-type constructs.

The size of all “inline” images depends on a configuration variable which specifies how much to enlarge or reduce them in relation to their original size in the printed version of the document (`MATH_SCALE_FACTOR`), i.e., scale factors of 0.5 or 2.0 make all images half or twice as large as the original. A value of 0.0 means that no scaling factor has to be applied.

On the other hand, display-type images (such as those generated by the environments `table`, `figure`, `equation`, or `minipage`) are controlled by the variable `FIGURE_SCALE_FACTOR`. The default value for both of these variables is 1.6.

Moreover, several parameters can affect the conversion of a single “figure” with the `\htmlimage` command:

```

sub navigation_panel {

    # Start with a horizontal rule (3-d dividing line)
    "<HR> ".

    # Now add few buttons with a space between them
    "$NEXT $UP $PREVIOUS $CONTENTS $INDEX $CUSTOM_BUTTONS" .

    "<BR>\n" . # Line break

    # If ‘‘next’’ section exists, add its title to the navigation panel
    ($NEXT_TITLE ? "<B> Next:</B> $NEXT_TITLE\n" : undef) .

    # Similarly with the ‘‘up’’ title ...
    ($UP_TITLE ? "<B>Up:</B> $UP_TITLE\n" : undef) .

    # ... and the ‘‘previous’’ title
    ($PREVIOUS_TITLE ? "<B> Previous:</B> $PREVIOUS_TITLE\n" : undef) .

    # Horizontal rule (3-d dividing line) and new paragraph
    "<HR> <P>\n"
}

```

Figure 10: Example definition of a navigation panel. (Note that “.” is the `perl` string concatenation operator and “#” signifies a comment).

`\htmlimage{options}`

This command can be used inside every environment that is normally translated into a display-type image. To be effective the `\htmlimage` command (and its options) must be placed inside the environment on which it has to operate. The argument *options* specifies how the image in question will be handled; it contains a comma-separated list of keyword-value pairs.

*scale=scale-factor*

the scale factor for the final image;

*external*

the image does not have to be included in the document, but a hyperlink whose URL points to it has to be inserted to access it;

*thumbnail=reduction-factor*

a small inlined image will be generated and placed in the caption; its size depends

on the specification *reduction-factor* that downsizes the image by that amount. Note that this option implies `external`.

`map=image-map-URL`  
turns the inlined image into an active image map.<sup>10</sup>

An example is the following L<sup>A</sup>T<sub>E</sub>X code:

```
\begin{figure}
  \htmlimage{thumbnail=0.25}
  \includegraphics{myfig.eps}
  \caption{description of my figure}
  \label{fig-myfig}
\end{figure}
```

`\htmlimage` can also be used to locally cancel out the effect of the configuration variable `FIGURE_SCALE_FACTOR`. For instance, if one does not want to resize a given image, then the command `\htmlimage{scale=0}` will do the trick.

### 3.9 Internationalization

A special variable, `TITLES_LANGUAGE`, in the initialization or configuration files determines the language in which some section titles will appear. For example, by setting it to

```
$TITLES_LANGUAGE = "french";
```

LaTeX2HTML will produce “Table des matières” instead of “Table of Contents”.

Currently, “french” and “english” are the only languages supported. It is not difficult, however, to add support for other languages in the file `latex2html.config`. As an example, below is shown the entry for French titles:

```
sub french_titles {
  $toc_title = "Table des matières";
  $lof_title = "Liste des figures";
  $lot_title = "Liste des tableaux";
  $idx_title = "Index";
  $bib_title = "Références";
  $info_title =
    "À propos de ce document...";
}
```

In order to provide full support for another language you may also want to replace the navigation buttons which come with LaTeX2HTML (which are by default in English) with your own. As long as the new buttons have the same filenames as the old ones there should not be a problem.

---

10. More information on active image maps is at the URL <http://wintermute.ncsa.uiuc.edu:8080/map-tutorial/image-maps.html>.

### 3.10 Known problems

Users of L<sup>A</sup>T<sub>E</sub>X2HTML should take note of the following shortcomings of the translator.

- *Unrecognized commands and environments.*  
Unrecognized *commands* are ignored and any arguments are left in the text. Unrecognized *environments* are passed to L<sup>A</sup>T<sub>E</sub>X and the result is included in the document as one or more inlined images. Users can take care of this by providing information to L<sup>A</sup>T<sub>E</sub>X2HTML on how to handle such cases, either by deciding to ignore them (see Section 3.5 on page 117), or by defining a perl procedure (see Appendix C).
- *Cross-references.*  
References in environments that are passed to L<sup>A</sup>T<sub>E</sub>X for processing (such as `\cite` or `\ref` commands), are not processed correctly. On the other hand, `\label` commands are handled satisfactorily.
- *Order-sensitive commands.*  
Commands affecting global parameters during the translation that are sensitive to the order in which they are processed may cause problems. In particular, counter manipulation with commands such as `\newcounter`, `\setcounter`, `\stepcounter` should be watched.
- *Index.*  
L<sup>A</sup>T<sub>E</sub>X2HTML generates its own index by saving the arguments of the `\index` command. The contents of the `\theindex` environment are ignored.
- *New definitions.*  
New definitions (with the commands: `\def`, `\newcommand`, `\newenvironment`, `\newtheorem`) will not work as expected if they are defined more than once. Indeed, only the last definition will be used throughout the document.
- *Scope of declarations and environments.*  
L<sup>A</sup>T<sub>E</sub>X2HTML processes sections as independent units. Thus, when the scope of a declaration or environment crosses section boundaries, the output may not be as expected.

## 4 HTML3 extensions to L<sup>A</sup>T<sub>E</sub>X2HTML

### 4.1 The MATH2HTML program

The simple notation for even complex mathematics and the diversity of the symbols and characters sets available makes L<sup>A</sup>T<sub>E</sub>X the typesetting system of choice in many of the scientific fields. Tens of thousands of articles, theses, and reports have been written in L<sup>A</sup>T<sub>E</sub>X and most publishing houses that deal with scientific papers use L<sup>A</sup>T<sub>E</sub>X for handling, storing and archiving their documents. Therefore it is to be expected that all these parties wish to protect their investment and prefer not to have to recode their mathematics formulae for hypertext purposes only.

The L<sup>A</sup>T<sub>E</sub>X2HTML translator solves the problem of presenting mathematics in HTML by converting each mathematical sentence into a bitmap image. Although simple and

straightforward, this approach seems a little unreasonable in general, since in many cases an article of a few pages can generate many hundreds of bitmap images, which have to be stored with the document, kept up to date, and transmitted with the document over the Internet, thus wasting an enormous amount of bandwidth. Therefore, a clear need for a translator from L<sup>A</sup>T<sub>E</sub>X mathematics into HTML3's primitive mathematics was considered an important goal. Thanks to the increased displaying capabilities of HTML3-compliant browsers, most inline mathematics and a fair proportion of display equations can be translated into HTML3 source code and hence transmitted in textual format together with the rest of the document, doing away with well over 90% of the images that are created in the HTML2 case where only bitmap images are generated. In addition, mathematics text can be searched for keywords as the rest of the document, thus increasing the value of the HTML document.

The `math2html` program has been interfaced to the `LaTeX2HTML` program via a new option `-html3`. When this option is specified, `LaTeX2HTML` will first pass the L<sup>A</sup>T<sub>E</sub>X input source code through the `math2html` translator. In this case, native HTML3 code will be generated for mathematics and tables when `math2html` can handle the input. In case `math2html` cannot parse the given L<sup>A</sup>T<sub>E</sub>X input, it gives an error message and `LaTeX2HTML` creates an image as usual.

At CERN we have translated thousands of pages of manuals and hundreds of physics articles. We found that `math2html` successfully translates on average 95% of all mathematics present in the input files, thus reducing by a substantial amount the number of generated bitmap images.

### *A few examples*

The HTML3 extensions translate quite a large fraction of not-too-complex L<sup>A</sup>T<sub>E</sub>X math constructs (for as far as they can be handled by the HTML3 DTD, of course).

A first explicit example is the code representing the differential cross-section of  $\delta$ -ray production. The original L<sup>A</sup>T<sub>E</sub>X code and its result as typeset by L<sup>A</sup>T<sub>E</sub>X are shown in parts (a) and (b) of Figure 11, while the result of the translation by `math2html` of the L<sup>A</sup>T<sub>E</sub>X source in (a) into HTML3 is shown in (c), yielding the output with the `arena` browser shown in (d). Part of the tree constructed by `math2html` when parsing this L<sup>A</sup>T<sub>E</sub>X input is shown in Figure 18 on page 165.

Multi-line mathematical constructs, such as arrays (`array` and `eqnarray` environments), are also handled without too many problems, and the present limits of the translation are due more to shortcomings of the (only) HTML3 browser `arena` (which is, after all, merely a beta-test version), than to intrinsic limitations in the approach. In Figure 12 we show the L<sup>A</sup>T<sub>E</sub>X source and result as seen with `arena` of two multi-line environments.

(a) L<sup>A</sup>T<sub>E</sub>X source that has to be translated:

```
\frac{d\sigma}{d\epsilon}=\frac{2\pi Zr_0^2m}{\beta^2(E-m)}\left[\frac{(\gamma-1)^2}{\gamma^2}+\frac{1}{\epsilon}\left(\frac{1}{\epsilon}-\frac{2\gamma-1}{\gamma^2}\right)+\frac{1}{1-\epsilon}\left(\frac{1}{1-\epsilon}-\frac{2\gamma-1}{\gamma^2}\right)\right]
```

(b) Result of the above source as typeset with L<sup>A</sup>T<sub>E</sub>X:

$$\frac{d\sigma}{d\epsilon} = \frac{2\pi Zr_0^2 m}{\beta^2(E-m)} \left[ \frac{(\gamma-1)^2}{\gamma^2} + \frac{1}{\epsilon} \left( \frac{1}{\epsilon} - \frac{2\gamma-1}{\gamma^2} \right) + \frac{1}{1-\epsilon} \left( \frac{1}{1-\epsilon} - \frac{2\gamma-1}{\gamma^2} \right) \right]$$

(c) Result of the translation of the code in (a) into HTML3:

```
<math><box>d&sigma;</over>d&epsilon;</box>=<box>2&pi;Zr<sub>0</sub><sup>2</sup>m<over>&beta;<sup>2</sup></sup>(E-m)</box>[<box>(&gamma;-1)<sup>2</sup></over>&gamma;<sup>2</sup></box>+<box>1<over>&epsilon;</box>(<box>1<over>&epsilon;</box>-<box>2&gamma;-1<over>&gamma;<sup>2</sup></box>)+<box>1<over>1-&epsilon;</box>(<box>1<over>1-&epsilon;</box>-<box>2&gamma;-1<over>&gamma;<sup>2</sup></box>)]</math>
```

(d) Result of viewing of the HTML3 code of (c) with the arena browser:

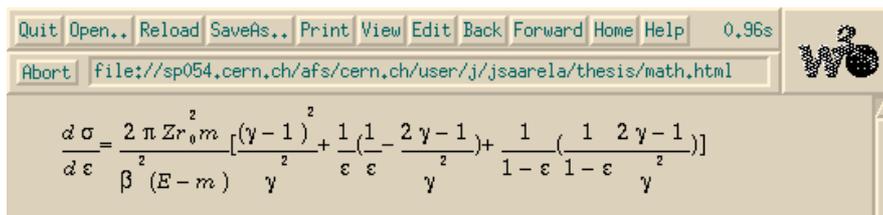


Figure 11: Example of transforming L<sup>A</sup>T<sub>E</sub>X code to HTML3 with math2html

### Writing convertible L<sup>A</sup>T<sub>E</sub>X

By following the rules below, one can expect the LaTeX2HTML translator enhanced with math2html to produce good output in terms of a low number of bitmap images.

- Do not write the base of a superscript or a subscript outside the mathematics markup, i.e.,  $a^2$  is not converted correctly but creates a bitmap image. The correct way is to write it  $a^2$  or  $\mathrm{a}^2$  depending, on whether or not one wants the letter “a” in math italic or in a roman font. When you leave the base outside of the math markup (the \$ signs) the text between the mathematics delimiters is passed to the math2html translator and the latter does not know where to place the mathematics start (<math>) tag.

```

\begin{eqnarray}
a & = & \sin \alpha_2 & \\
b & = & \cos \omega_3 & \\
\Gamma & = & \Phi + \Theta & \\
\end{eqnarray}

\[
\begin{array}{cccccc}
a_{11} & & & & & \\
a_{21} & & a_{22} & & & \\
a_{31} & & a_{32} & & a_{33} & \\
a_{41} & & a_{42} & & a_{43} & & a_{44} \\
a_{51} & & a_{52} & & a_{53} & & a_{54} & & a_{55} \\
a_{61} & & a_{62} & & a_{63} & & a_{64} & & a_{65} & & a_{66}
\end{array}
\]

```

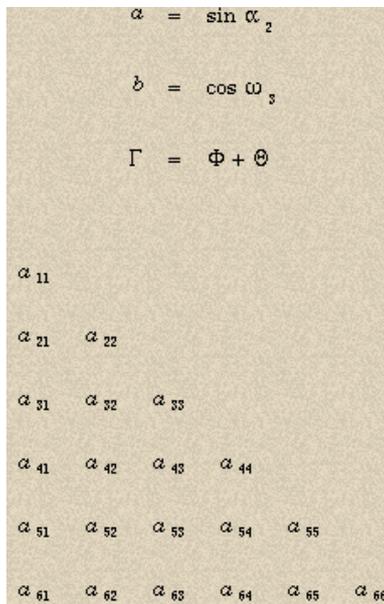


Figure 12: How `math2html` translates `LATEX` multi-line mathematics into HTML3

- Do not write nested `array`/`tabular` environments. The `math2html` translator cannot create an HTML3 counterpart for that markup since the HTML3 table model does not allow nested tables. Keeping the tables simple (not nested, for example) will improve their reusability.

#### 4.2 Tables to HTML3 conversion

Hennecke recently developed some code for treating `LATEX`'s `tabular` environment with `LaTeX2HTML` by translating it into HTML3-compliant tables. His patches<sup>11</sup> allow `LaTeX2HTML` to translate most `LATEX` tables reasonably well. There are a few things it cannot do, but mainly because HTML tables are not quite as powerful as `LATEX` tables. Most importantly, HTML tables are quite limited when it comes to borders, since they are not nearly as flexible in specifying borders as `LATEX` tables. In his implementation, when a `LATEX` table has a border anywhere, the resulting HTML table will have borders around all cells. `LATEX` commands inside cells are treated as they should and declarations are limited in scope to the cell in which they appear (just as in `LATEX` itself).

His additions can be placed in the `LaTeX2HTML` `perl` code itself, or in the customization files. In any case to leave open the possibility of generating tables with and without

11. Available from the URL <ftp://ftp.crc.ricoh.com/pub/www/12h/tables.tar.gz>. The author Marcus E. Hennecke can be reached by email at [marcush@crc.ricoh.com](mailto:marcush@crc.ricoh.com).

this feature turned on, a new command line option `-html_level` can be used to specify the level of HTML to be produced.

### Examples

First, we look at a simple table with different alignments:

```
\begin{tabular}{|l|c|r|} \hline
first column & second column & third column \\ \hline
111 111      & 22 22 22      & 3 3 3 3      \\ \hline
\end{tabular}
```

The result is seen at the top of Figure 13.

Math can also be handled (in this case it will be translated into images). With a little bit of “hand-work” it could be translated into native HTML3:

```
\begin{tabular}{|l|} \hline
 $10^{10}$  & a big number \\ \hline
 $10^{-999}$  & a small number \\ \hline
\end{tabular}
```

The result is seen in the second table from the top in Figure 13.

Modifications to text inside cells remain limited to that cell (as it should). In the present version only *one* `\multicolumn` command is recognized (when more than one such command is encountered inside a row, only the first one is taken into account):

```
\begin{tabular}{|l|}
\multicolumn{2}{c}{\bf PostScript type 1 fonts} \\
\em Courier      & \\
cour, courb, courbi, couri & \\
\em Charter      & \\
bchb, bchbi, bchr, bchri & \\
\em Nimbus      & \\
unmr, unmr      & \\
\em URW Antiqua & \\
uaqrrc          & \\
\em URW Grotesk & \\
ugqp            & \\
\em Utopia      & \\
putb, putbi, putr, putri & \\
\end{tabular}
```

The result is seen in the third table from the top of Figure 13. Note that, even though only vertical rules were specified in the `tabular`'s preamble, rules are drawn everywhere. This is because the `BORDER` attribute of the `<TABLE>` tag in HTML3 has only one value, i.e., borders are present or absent everywhere.

first column	second column	third column
111 111	22 22 22	3 3 3 3

$10^{10^{10}}$	a big number
$10^{-999}$	a small number

PostScript type 1 fonts	
<i>Courier</i>	cour, courb, courbi, couri
<i>Charter</i>	bchb, bchbi, bchr, bchri
<i>Nimbus</i>	unmr, unmrs
<i>URW Antiqua</i>	uaqrcc
<i>URW Grotesk</i>	ugqp
<i>Utopia</i>	putb, putbi, putr, putri

**global top title**

a11

a21 a22

a31 a32 a33

a41 a42 a43 a44

a51 a52 a53 a54 a55

a61 a62 a63 a64 a65 a66

*columns 1-6 bottom title*

Figure 13: Four examples of tabular environments translated automatically to HTML3 as viewed with the arena browser

Our final example has again a few `\multicolumn` commands, but also shows that non-specified cells are treated gracefully (this can be compared to the example in Section 4.1, where a similar table was built as an array inside math mode):

```
\begin{tabular}{cccccc}
\multicolumn{6}{c}{\bf global top title}\!
a11 \! \!
a21 & a22 \! \!
a31 & a32 & a33 \! \!
a41 & a42 & a43 & a44 \! \!
a51 & a52 & a53 & a54 & a55 \! \!
a61 & a62 & a63 & a64 & a65 & a66 \! \!
\multicolumn{6}{c}{\em columns 1-6 bottom title}
\end{tabular}
```

The result is seen as the bottom table of Figure 13. As no vertical nor horizontal rules were specified in the input, the resulting table has no borders.

## 5 Caml based L<sup>A</sup>T<sub>E</sub>X to HTML translation

Xavier Leroy (INRIA, France) developed a L<sup>A</sup>T<sub>E</sub>X to HTML translator based on the Caml language.<sup>12</sup>

What was needed was to translate a 200-page technical document (the reference manual and user's documentation for their implementation of the Caml Light programming language). This manual was written in L<sup>A</sup>T<sub>E</sub>X and contained some rather non-standard environments and macros written directly in T<sub>E</sub>X. Parts of the document were automatically generated: syntactic definitions (typeset from BNF descriptions) and descriptions of library functions (extracted from commented source code).

### 5.1 Why not just use L<sup>A</sup>T<sub>E</sub>X<sub>2</sub>HTML

When L<sup>A</sup>T<sub>E</sub>X<sub>2</sub>HTML finds a L<sup>A</sup>T<sub>E</sub>X construct that it does not know how to translate into HTML, it simply turns it into a bitmap. This approach was considered inappropriate by Leroy *et al.*, since

- information transformed into a bitmap is not searchable;
- bitmaps cannot easily be integrated into Macintosh or Windows online documentation systems;
- bitmaps are generally hard to read, since their resolution usually does not match that of the HTML viewer;
- as bitmaps can be quite large, transmission times increase and network bandwidth suffers.

12. More information can be found at the URL <http://pauillac.inria.fr/~xleroy/w4g.html>.

In order to minimize the generation of bitmaps and to allow the production of a better quality HTML source, the information in the  $\text{\LaTeX}$  source was tagged by  $\text{\LaTeX}$  macros to explicitly show its semantics meaning. Special care was taken to avoid inline mathematics constructs, since they result in bitmap images, for example,  $\text{\var{x}}$  was preferred to its typographic equivalent  $\$x\$$  (denoting a meta-variable), and  $\text{\nth{v}{n}}$  was used to mean the  $n$ -th component of  $v$ , rather than writing  $\$v_n\$$ . The same technique was also used to eliminate “low-level” typesetting constructs and environments such as `center` and `tabular`.

When typesetting the document with  $\text{\LaTeX}$  these new commands were simple translated into the needed typographic representation, but during the translation into HTML they were explicitly recognized and individually translated into a form that corresponds with the possibilities of HTML. For instance,  $\text{\nth{v}{n}}$  would become something like `&lt;i>v(n)&lt;/i>`, showing `<i>v(n)</i>`.

The programs that automatically generated BNF or program fraction for inclusion in the  $\text{\LaTeX}$  source were adapted so that its contents could now also be included without problems in the HTML source by “hiding” the generated material inside a `rawhtml` environment.

Finally, the few places where more complex mathematical constructs were needed were hand-translated into a form acceptable to HTML and stored inside a `rawhtml` environment, leaving the original mathematics expressions inside a `latexonly` environment. Thus both the  $\text{\LaTeX}$  and HTML views of the document were optimized. Although in principle such an approach can lead to synchronization problems between the  $\text{\LaTeX}$  and HTML parts of the information, it was found that, due to the care that was taken in using the generic markup approach outlined above, only about 0.2% of the source had to be manually translated.

Although Leroy and his collaborators originally planned to use `LaTeX2HTML` for translating their document into HTML, they found that some commands (especially those using verbatim-like constructs, most notably the `alltt` environment) cannot be defined in `perl` in an easy way using the interface of init files described earlier. Therefore modifications have to be made inside the body of the `LaTeX2HTML` program itself, and this is very complicated since the inner workings of `LaTeX2HTML` are undocumented and scarcely commented, so that the `perl` code is not always clear to follow. Also, the memory requirements of `LaTeX2HTML` (especially the pre-1995 versions, when the tests were done) can be huge, exhausting all the memory available on the machine and causing the program to crash (this should no longer be a problem with the current version of `LaTeX2HTML` if the  $\text{\LaTeX}$  source is divided into a set of smaller files). They therefore decided to write their own  $\text{\LaTeX}$ -to-HTML translator for the extended subset of  $\text{\LaTeX}$  commands they used.

This translator works in two main stages:

- The translator first reads the whole  $\text{\LaTeX}$  document and outputs one large HTML document. It is written in `Caml Light` and uses the lexical analyzer generator

`camllex` (the Caml equivalent of `lex` for C) heavily. Note that Caml is a modern, type-safe high-level programming language with good memory management, so that the translator has negligible memory requirements, runs quickly, is easy to extend, and took little time to develop.

- The output of the translator is then split into smaller parts (for instance at the `<H1>` or `<H2>` heading levels), and these parts are linked together using “Next” and “Previous” buttons. This linking is performed by two simple `perl` scripts.

In order to get a feeling of the result of the translation, one can look at a randomly chosen page from the manual that was converted. Figure 14 shows the result of L<sup>A</sup>T<sub>E</sub>X (viewed with `xdvi`) and Figure 15 is the result of the HTML conversion, as shown by Mosaic.<sup>13</sup> Appendix E takes a closer look at how the Caml system translates L<sup>A</sup>T<sub>E</sub>X commands into HTML.

## 5.2 Discussion

Based on their experience with writing and using their translator Leroy and collaborators draw the conclusions summarized in the next sections.

### *About the HTML language*

Despite its apparent simplicity, the HTML language is almost rich enough to format T<sub>E</sub>X-intensive technical documentation. The sole features that were badly missed were tables, subscripts, and superscripts. This is much less true today, since HTML3 already contains an interesting table model, and allows for super and subscripts. Moreover, the latest versions of Mosaic and Netscape support these functions.

### *About HTML viewers*

The quality of the typesetting performed by popular HTML viewers (such as Mosaic and Netscape) is very often insufficient. It seems especially difficult to ensure font consistency throughout a document.

The difficulty in finding good translators and adequate viewers probably has to do with the immaturity of the field. Leroy et. al. are convinced that the widespread use of `perl` for programming translation tools is partly responsible for this situation. They state that `perl` is inherently not suited to the parsing and transformation of structured languages, such as L<sup>A</sup>T<sub>E</sub>X and HTML, and go on to say that languages with high-level parsing capabilities, real data structures and clean semantics are much more suited for these tasks.

They also ask the question: what is the best markup language for preparing documentation so that it can be nicely printed but also easily transformed into HTML for publishing on the Web? They accept that, L<sup>A</sup>T<sub>E</sub>X presently being the *de facto* standard markup language used in computing and other science fields, it will be difficult in the

---

13. The complete manual – HTML and dvi files – are at the URLs <http://pauillac.inria.fr/~xleroy/man-caml/> and <ftp://ftp.inria.fr/lang/caml-light/Release7beta/cl7refman.dvi.gz>, respectively.

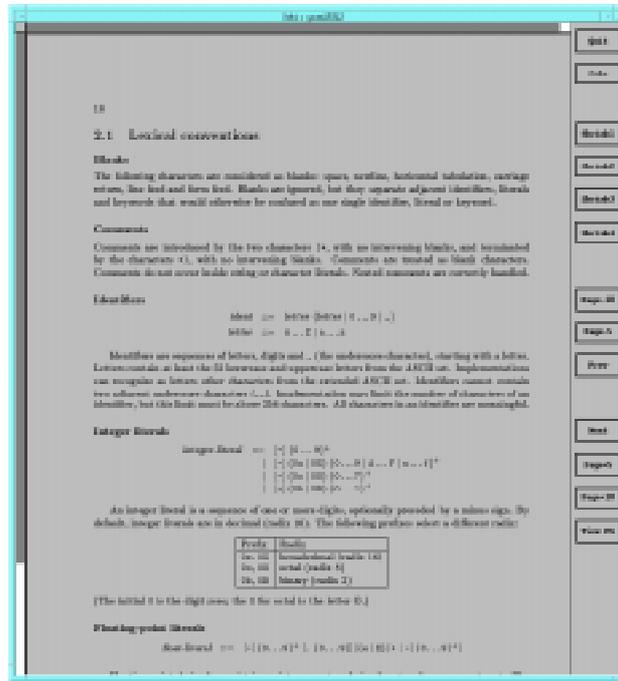


Figure 14: Example page of CamL manual (LATEX viewed with xdvi)

short term to propose a solution other than to invest more effort in developing cleverer and more comprehensive LATEX-to-HTML translators.

## 6 Converting HTML to LATEX

Although utilities for obtaining PostScript representations from HTML files are readily available, either using HTML browsers, such as Mosaic, that offer PostScript as one of their output formats, or directly (for example, [https](https://info.cern.ch/hypertext/WWW/Tools/https.html)<sup>14</sup>) the visual layout of these documents is often appalling, and the structuring of the information has been made almost completely invisible. Often one would like to obtain a nicely typeset document that presents the information marked up in HTML in a structured way, with all document elements clearly identifiable. A translation into LATEX allows one to combine the power of the T<sub>E</sub>X typesetting engine while at the same time exploiting the structural similarities between HTML and LATEX as explained in Section 1 and Table 1.

14. More information is at the URL <http://info.cern.ch/hypertext/WWW/Tools/https.html>.

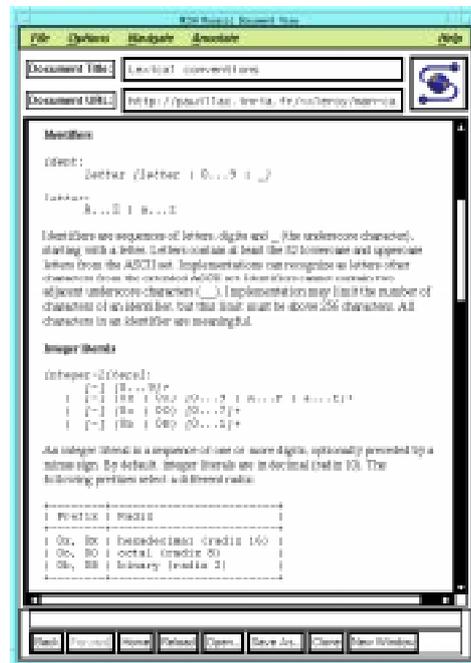


Figure 15: Example page of CamL manual (HTML converted with CamL based translator viewed with Mosaic)

A first program  $\text{HTML2L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  translates a large fraction of the HTML commands into  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ , while  $\text{SGML2T}_{\text{E}}\text{X}$  takes a more general approach and allows the transformation of an arbitrary SGML source into  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ .

### 6.1 $\text{HTML2L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ , an HTML-to- $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ converter

$\text{HTML2L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  is a C-program written by Nathan Torkington (New Zealand). Basically, the HTML parser of the NCSA Mosaic HTML browser is used for the translation. The calling sequence of the program is:

```
html2latex [options] [filenames]
```

For each input file specified,  $\text{HTML2L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  transforms the HTML markup in the source into the equivalent  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  markup. When no filenames are specified,  $\text{HTML2L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  will display a short description of how to use the program. If *filenames* is equal to  $-$ , then the input text is read on standard input `stdin`. For each input file an output file with the same name, but with the extension `.tex` instead of `.html` is generated.

### Options

HTML2LaTeX has a number of options that modify its way of operation. The more important are:

- n           number the sections;
- p           start a new page after the titlepage (if present) or the table of contents (if present);
- c           generate a table of contents;
- s           write output information on `stdout`;
- t *Title*    generate a titlepage containing the title *Title*;
- a *Author*   generate a titlepage containing the author(s) *Author*;
- h *Start-Text* introduce the text *Start-Text* immediately following the command `\begin{document}`;
- h *End-Text*   introduce the text *End-Text* immediately preceding the command `\end{document}`;
- o *options*   specify the options *options* on the `\documentclass` command.

### Examples

Let us consider the following command:

```
html2latex -n - < file.html | more
```

In this case the file `file.html` is transformed into  $\text{\LaTeX}$  and the result is shown on the screen. The option `-n` makes sure no section numbers are generated.

A more complex example is shown below:

```
html2latex -t 'HTML for Pedestrians' \
-a 'First Last' -p \
-c -o '[12pt,twoside]{article}' \
my-article.html
```

In this case file `my-article.html` will be read, and the output written to the file `my-article.tex`. A titlepage (using the text "HTML for Pedestrians" as title and "First Last" as author) will be output on a separate page (option `-p`). A Table of contents (option `-c`) followed by a new page (option `-p` again) will also be generated. Sections will be numbered (default behavior). The  $\text{\LaTeX}$  document will be typeset at 12 point using the document option `twoside`, to allow *two-sided* printing.

### Limitations

The present version of HTML2LaTeX recognizes the following HTML tags: `<TITLE>`, `<H1>` to `<H6>`, for lists `<OL>`, `<UL>`, `<DT>`, `<DD>` and `<LI>`, plus the presentation tags `<B>`, `<I>`, `<U>`, `<EM>`, `<CODE>`, `<SAMP>`, `<STRONG>`, `<KBD>`, `<VAR>`, `<DFN>`, `<CITE>`, and `<LISTING>`. Of the entities only `&amp;`, `&lt;` and `&gt;` are handled correctly. The content fields of the tags `<ADDRESS>`, `<DIR>` and `<MENU>` are not handled correctly. Moreover, the `COMPACT` attribute of the `<DL>` tag is not honored and the text of the

<TITLE> tag is ignored. Even worse, the body of the <PRE> elements are completely ignored.

Note that the complete HTML file is read into memory; this can lead to problems when handling large files on machines with limited memory capabilities.

## 6.2 SGML2TeX, a General-Purpose SGML to L<sup>A</sup>T<sub>E</sub>X Converter

SGML2TeX<sup>15</sup> is a program written by Peter Flynn (Cork, Ireland) that translates SGML tags into T<sub>E</sub>X instructions. At present the system is only implemented in PCL<sup>16</sup> running runs under ms-dos on a PC but the author has plans to rewrite it in a more portable programming language.

SGML2TeX does not verify the SGML source for correctness but accepts all SGML documents marked up using the reference concrete syntax. It is up to the user to define a L<sup>A</sup>T<sub>E</sub>X equivalent for each of the SGML document elements, their attributes, and the entities used in the source. A configuration file that contains a set of such predefined correspondences for certain elements, attributes, or entities, can be read by SGML2TeX, thus substantially alleviating the task of the user, who will only have to provide the missing definitions. By default, i.e., in the absence of an explicit translation, SGML elements are translated in a form acceptable to L<sup>A</sup>T<sub>E</sub>X by adopting the following conventions:

- start tags get the prefix `\start` and end tags the prefix `\finish` followed by the tag-name in upper case, followed by a pair of braces `{}`. This pair of braces corresponds to a *do-nothing* definition for each of the tags thus handled;
- SGML entities of the form `&ent;` are translated into `\ent{}` and written into the output file;
- attributes are handled in the same way, but their value is specified between curly braces like a L<sup>A</sup>T<sub>E</sub>X argument.

## Acknowledgments

We sincerely thank Nelson Beebe (Utah University, [beebe@math.utah.edu](mailto:beebe@math.utah.edu)) for e-mail discussions and for his detailed comments of the compuscript. His many suggestions improvements have without doubt substantially increased the readability and quality of the article. We also want to acknowledge Steven Kennedy (CERN) for proofreading the article.

## References

- [1] M. Goossens, F. Mittelbach, and A. Samarin. *The L<sup>A</sup>T<sub>E</sub>X Companion*. Addison-Wesley, Reading, 1994.

---

15. For more information see the URL <http://info.cern.ch/hypertext/WWW/Tools/SGML2TeX.html>.

16. PCL stands for Personal Computer Language, an interpreted language for dos on the \*86 chips. It is a very fast prototyping tool, not a production language since it cannot link executable images.

- [2] M. Goossens and E. van Herwijnen. The elementary particle entity notation (pen) scheme. *TUGboat*, 13(1):201–207, July 1992.
- [3] L. Lamport. *L<sup>A</sup>T<sub>E</sub>X, User's Guide and Reference Manual (2nd Edition)*. Addison-Wesley, Reading, 1994.
- [4] J.K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, 1994.
- [5] Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice Hall, Inc., Englewood Cliffs, N.J., 1991.
- [6] J. Schrod. Towards interactivity for T<sub>E</sub>X. *TUGboat*, 15(3):309–317, September 1994.
- [7] R.L. Schwartz. *Learning Perl*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1993.
- [8] D. Till. Reach yourself perl in 21 days. *Sams publishing*, 1995.
- [9] L. Wall and Schwartz R.L. *Programming Perl*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1991.

A complete and up-to-date list of titles of books on HTML and `per1` is maintained by Nelson Beebe (Utah University, [beebe@math.utah.edu](mailto:beebe@math.utah.edu)) and can be found in his BibT<sub>E</sub>X databases `sgml.bib` and `unix.bib`, respectively, in the directory with URL address `ftp://ftp.math.utah.edu/pub/tex/bib/`.

## Appendices

Appendices A and B present a few practical details that we found particularly relevant when installing or troubleshooting LaT<sub>E</sub>X2HTML.<sup>17</sup> Appendix C then provides some more information about the internal workings of the LaT<sub>E</sub>X2HTML program and how it can be extended by writing `per1` procedures. Finally, Appendix D contains technical information about the `math2html` extension to LaT<sub>E</sub>X2HTML, while Appendix E takes a closer look at Leroy's Caml-based L<sup>A</sup>T<sub>E</sub>X-to-HTML translator.

## Appendix A: L<sup>A</sup>T<sub>E</sub>X2HTML – Installation

### A.1 Requirements to run L<sup>A</sup>T<sub>E</sub>X2HTML

LaT<sub>E</sub>X2HTML uses several publicly available tools that can be readily found on most computer platforms, namely:

- L<sup>A</sup>T<sub>E</sub>X (of course).
- `per1` (version 4 from patch level 36 onward, or, even better, version 5).
- DBM or NDBM, the Unix DataBase Management system.
- `dvips` (version 5.516 or later) or `dvipsk`.

---

<sup>17</sup> These sections are adapted from the LaT<sub>E</sub>X2HTML manual that is available at the URL <http://cb1.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html>.

- `gs` (Ghostscript version 2.6.1 or later).
- The `pbmpplus` or better still the `netpbm` libraries; some of the filters in those libraries are used during the postscript to GIF conversion.
- For making transparent inlined images one needs `giftrans.c`<sup>18</sup> by A. Ley together with `pbmpplus`. Alternatively, `netpbm` will do the trick.

To reduce the memory requirements of the translation, L<sup>A</sup>T<sub>E</sub>X2HTML spawns off separate Unix processes to deal with each of the `input`'ed or `include`'d files. As each process terminates, all the space that it used is reclaimed. Asynchronous communication between processes takes place using the Unix DataBase Management system (DBM or NDBM) which should be present. To take advantage of these changes it is necessary to split the source text into multiple files that can be assembled using L<sup>A</sup>T<sub>E</sub>X's `\input` or `\include` commands.

When `gs` or the `pbmpplus` (`netpbm`) library are not available, one can still generate HTML output, but without images (using the `-no_images` option). Also, do not forget to include the `html` package with the `\usepackage` command if you want to include any of the hypertext extension commands described in Section 3.6.

## A.2 Installing L<sup>A</sup>T<sub>E</sub>X2HTML

Those intending to install L<sup>A</sup>T<sub>E</sub>X2HTML on their system should read the manual in detail. Below we describe only the main steps.

- *Specify where `perl` is on the system.*  
In the files `latex2html`, `texexpand`, `pstogif`, and `install-test` modify the first line saying where `perl` is on your system.
- *Specify where the external utilities are on the system.*  
In the file `latex2html.config` give the correct pathnames for some directories (the `latex2html` directory and the `pbmpplus` or `netpbm` library) and some executables (`latex`, `dvips`, `gs`).  
Note that L<sup>A</sup>T<sub>E</sub>X2HTML can be run even if one does not have some of these utilities.

One can also include the following supplementary customization:

- *Setting up different initialization files.*  
One can customize on a "per user" basis the initialization file. To this effect one should copy the file `dot.latex2html-init` into the home directory of any user who wants it, modify it according to the user's preferences and rename it to `.latex2html-init`.  
At runtime both `latex2html.config` and `$HOME/.latex2html-init` files will be loaded, but the latter will take precedence. Moreover, one can also set up a "per directory" initialization file by copying a version of the initialization file `.latex2html-init` into each directory where it should be effective. In this case

---

18. <ftp://ftp.rz.uni-karlsruhe.de/pub/net/www/tools/giftrans.c>.

an initialization file `/X/Y/Z/.latex2html-init` takes precedence over all other initialization files if `/X/Y/Z` is the “current directory” when `LaTeX2HTML` is invoked.

- *Make local copies of the LaTeX2HTML icons.*

The `icons` subdirectory should be copied to a place in the local WWW tree where it can be served by the local server. Therefore, in the file `latex2html.config` file the value of the variable `$ICONSERVER` should be changed accordingly.

## Appendix B: LaTeX2HTML – Troubleshooting

This section gives a few hints about how to solve problems with `LaTeX2HTML`. As a general rule, if one gets really lost, one can obtain a lot of information from the `perl` system by setting the environment variable `DEBUG` to `1`. In particular it will point out missing files or utilities. Below we present some often occurring problems and propose a way how to deal with them.

### LaTeX2HTML just stops without further warnings

Check the package files that are included, since they might contain raw `TeX` commands, which cannot be handled. In this case start `LaTeX2HTML` with the option `-dont_include` followed by the name of the package file in question. Alternatively, one can add the name of the package file to the variable `DONT_INCLUDE` in the `HOME/.latex2html-init` file, or create one in the current directory containing the following lines:

```
$DONT_INCLUDE = "$DONT_INCLUDE:<name-of-package-file>";
1; # This must be the last line
```

Similarly, when the `LaTeX` source file itself contains raw `TeX` command (`\let` is a common example!) `LaTeX2HTML` might also stop. Such commands should therefore be introduced inside a `latexonly` environment.

### LaTeX2HTML gives an “Out of memory” message and crashes

Divide the `LaTeX` source file into several files that can be input using `\include` commands. One can also try the `-no_images` option.

### The “tilde” (~) does not show.

The easiest solution is to use the command `\~{}`. Alternatively it is possible to write something like:

```
\htmladdnormallink{mylink}
\begin{rawhtml}
  {http://host/~me/path/file.html}
\end{rawhtml}
```

**Macro definitions do not work correctly**

As already mentioned, plain T<sub>E</sub>X definitions are converted. But there can be problems even when using L<sup>A</sup>T<sub>E</sub>X definitions (with the `\newcommand` and `\newenvironment` commands) if such definitions make use of *sectioning* or *verbatim* commands, since these are handled in a special way by LaTeX2HTML and cannot be used in macro definitions.

**L<sup>A</sup>T<sub>E</sub>X2HTML behaves differently when running on the same file**

When noticing strange side-effects due to files remaining from previous runs of LaTeX2HTML one can use the option `-no_reuse` and choose (d) when prompted. This deletes intermediate files generated during previous runs. One can also delete those files oneself by removing the complete subdirectory created by LaTeX2HTML for storing the translated files. Note that in this case the image reuse mechanism is disabled.

```
> latex2html -no_reuse myfile.tex
This is LaTeX2HTML Version 95.1 (Fri Jan 20 1995) by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.
OPENING /afs/cern.ch/user/goossens/myfile.tex
Cannot create directory ./myfile: File exists
(r) Reuse the images in the old directory OR
(d) *** DELETE *** ./myfile AND ITS CONTENTS OR
(q) Quit ?
:d
```

**Cannot convert PostScript images included in the L<sup>A</sup>T<sub>E</sub>X file**

It is likely that the macros used for including PostScript files (for example, `\epsffile` or `\includegraphics`) are not understood by LaTeX2HTML. To avoid this problem enclose them in an environment which will be passed to L<sup>A</sup>T<sub>E</sub>X anyway, for instance:

```
\begin{figure}
  \epsffile{<PostScript file name>}
\end{figure}
```

Another reason why this might happen is that the shell environment variable `TEXINPUTS` is undefined. This is not always fatal but if you have problems you can use full pathnames for included postscript files (even when the PostScript files are in the same directory as the L<sup>A</sup>T<sub>E</sub>X source file). Therefore it is important to check the setting of the `TEXINPUTS` environment variable and make sure that it ends in a colon ":", for example, `./:somedir:`.

**Some of the inlined images are in the wrong places**

This occurs when any one of the inlined images is more than a (paper) page long. This is sometimes the case with very large tables or large PostScript images. In this

case, one should specify a larger paper size (such as "a3", "a2", or even "a0") instead of the default ("a4") using the LaTeX2HTML variable PAPERSIZE in the file latex2html.config.

### The labels of figures, tables or equations are wrong

This can happen if inside figures, tables, equations or counters are used inside conditional text, i.e., inside a latexonly or a htmlonly environment.

### With Ghostscript 3.X inline images are no longer generated for equations, etc.

One can run the installation script install-test again, or else change the way gs is invoked in the file pstogif, using something like:

```
open (GS, "|$GS -q -sDEVICE=ppmraw -sOutputFile=$base.ppm $base.ps");
```

### Cannot get it to generate inlined images

Try a small test file for example,

```
% image-test.tex
\documentclass{article}
\begin{document}
Some text followed by \fbox{some more text in a box}.
\end{document}
```

One should get something like the following:

```
> latex2html image-test.tex
This is LaTeX2HTML Version 95.1
      (Fri Jan 20 1995) by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/usr/goossens/image-test.tex

Reading ...
Processing macros ...
Translating ...0/1.....1/1.....
Writing image file ...
This is TeX, Version 3.1415 (C version 6.1)
(images.tex
LaTeX2e <1994/12/01>
Generating postscript images using dvips ...
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.05.08:1958' -> 6666_image
(-> 6666_image001) <tex.pro>[1]
```

```

Writing 6666_image001.ppm
Writing img1.gif
Doing section links .....
Done.

```

Problems encountered during the conversion from PostScript to GIF can be located by doing the translation manually, as shown below for a generation using `gs 3.33`.

```

> latex image-test
This is TeX, Version 3.1415 (C version 6.1)
(image-test.tex
LaTeX2e <1994/12/01>
(/usr/local/lib/texmf/tex/latex/base/article.cls
Document Class: article 1994/12/09 v1.2x Standard LaTeX document class
(/usr/local/lib/texmf/tex/latex/base/size10.clo))
No file image-test.aux.
[1] (image-test.aux)
Output written on image-test.dvi (1 page, 348 bytes).
Transcript written on image-test.log.
> dvips -o image-test.ps image-test.dvi
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.05.08:2006' -> image-test.ps
<tex.pro>. [1]
cblelca% gs -dNODISPLAY pstoppm.ps
> gs -dNODISPLAY pstoppm.ps
Aladdin Ghostscript 3.33 (4/10/1995)
Copyright (C) 1995 Aladdin Enterprises, Menlo Park, CA. All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
Usage: (file) ppmNrun
    converts file.ps to file.ppm (single page),
    or file.1ppm, file.2ppm, ... (multi page).
    N is # of bits per pixel (1, 8, or 24).
Examples: (golfer) ppm1run ..or.. (escher) ppm8run
Optional commands you can give first:
    horiz_DPI vert_DPI ppmsetdensity
    horiz_inches vert_inches ppmsetpagesize
    (dirname/) ppmsetprefix
    page_num ppmsetfirstpagenumber
GS>(image-test) ppm1run
Writing image-test.ppm
GS>quit
> pnmcrop image-test.ppm >image-test.crop.ppm
pnmcrop: cropping 74 rows off the top
pnmcrop: cropping 139 rows off the bottom
pnmcrop: cropping 149 cols off the left
pnmcrop: cropping 249 cols off the right
> ppmtogif image-test.crop.ppm >image-test.gif
ppmtogif: computing colormap...
ppmtogif: 2 colors found

```

### Still no inlined images are obtained

When there have been no problems with the above file `image-test.tex` but some images have still not been successfully converted in some of the files then one should look in the directory with the generated HTML files for the two files `images.tex` and

images.log. In particular, one should check whether there is something unusual in these files. One can copy the source images.tex into the directory of the original L<sup>A</sup>T<sub>E</sub>X file, run L<sup>A</sup>T<sub>E</sub>X on images.tex and check for any errors in the log file images.log. If errors are found then one should fix images.tex, put it back into the subdirectory with the HTML files, and run LaTeX2HTML on the original document using the option `-images_only`.

If one gets into trouble, then one should rerun LaTeX2HTML with the options `-no_reuse` and `-no_images`, for example,

```
> latex2html -no_reuse -no_images image-test.tex
This is LaTeX2HTML Version 95.1 (Fri Jan 20 1995) by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/user/goossens/image-test.tex
Cannot create directory ./image-test: File exists
(r) Reuse the images in the old directory OR
(d) *** DELETE *** ./image-test AND ITS CONTENTS OR
(q) Quit ?
:d

Reading ...
Processing macros ...
Translating ...0/1.....1/1.....
Writing image file ...

This is TeX, Version 3.1415 (C version 6.1)
(images.tex
LaTeX2e <1994/12/01>

Doing section links .....

***** WARNINGS *****

If you are having problems displaying the correct images with Mosaic,
try selecting "Flush Image Cache" from "Options" in the menu-bar and
then reload the HTML file.

Done.
```

Now one should look into the file images.tex (as described above) and correct possible problems. Once everything seems alright, LaTeX2HTML should be run again with the option `-images_only`.

Some problems in displaying the correct inlined images may be due to the image-caching mechanisms of the browser. With some browsers, a simple "Reload Current Document" will be enough to refresh the images, but with others (including Mosaic) one may need to refresh the cache explicitly. With Mosaic one should select "Flush Image Cache" in the Options menu, then reload the HTML file.

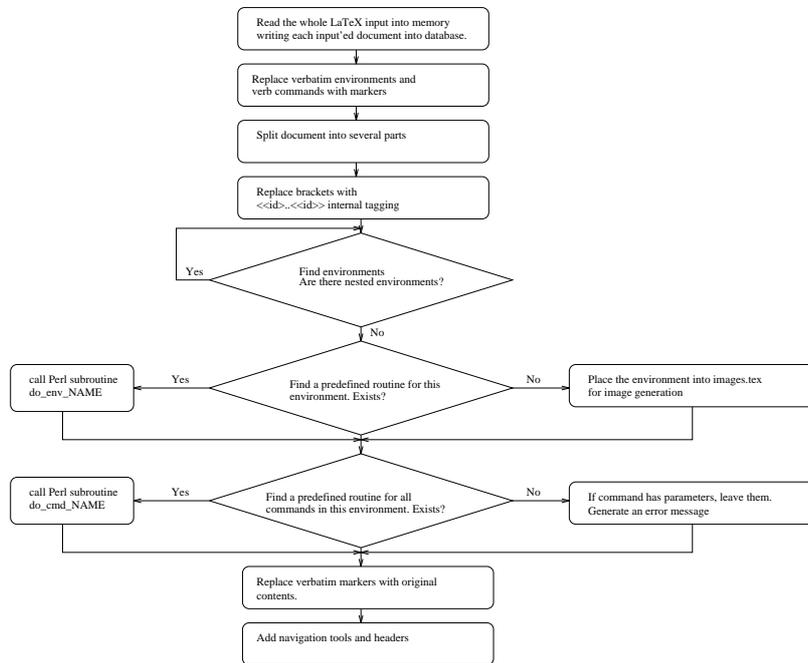


Figure 16: Flow diagram of the LaTeX2HTML system

## Appendix C: For PERL hackers only – Inside L<sup>A</sup>T<sub>E</sub>X2HTML

The basic principle of LaTeX2HTML is that it reads a L<sup>A</sup>T<sub>E</sub>X source code document, converts the parts it recognizes into HTML and passes unknown parts to L<sup>A</sup>T<sub>E</sub>X, which, in turn, creates pictures out of them. These pictures are then placed inside the final hypertext document.

As discussed in Section 3.3, the program is started by specifying the L<sup>A</sup>T<sub>E</sub>X source code document together with a set of parameters. The result is a number of HTML documents and images as GIF or PostScript files. An overall flow-diagram is shown in Figure 16

Unknown environments, tables, or pictures are also passed on to L<sup>A</sup>T<sub>E</sub>X and transformed into GIF or PostScript images, and kept inline or outside the hypertext documents.

### C.1 The translation process

Below are shown the various phases that a document goes through when translated from L<sup>A</sup>T<sub>E</sub>X into HTML. Let us first consider the original L<sup>A</sup>T<sub>E</sub>X source document:

```

\documentclass{article}
\begin{document}

```

```

\section{test}
This is a list of two items:
\begin{itemize}
  \item{First item}
  \item{Second item}
\end{itemize}
\begin{verbatim}
This section includes some special characters such as $, <, >, _ .
\end{verbatim}
\end{document}

```

This L<sup>A</sup>T<sub>E</sub>X source is first preprocessed by removing parts which have a special meaning in L<sup>A</sup>T<sub>E</sub>X, such as the `verbatim` and `\verb` constructs. In this example the `verbatim` part is stored in a separate file for later reference and a marker is placed inside the document together with a unique identification number “<id>” that will later be used to find the original text.

```

\documentclass{article}
\begin{document}
\section{test}

This is a list of two items:

\begin{itemize}
  \item{First item}
  \item{Second item}
\end{itemize}

<tex2html_verbatim_mark>verbatim1

\end{document}

```

At the end of preprocessing in the `mark_string` procedure, all the bracketed areas are replaced by <<id><<id>> tags where “id” is identical at both ends of the originally bracketed text.

```

\documentclass<<1>>article<<1>>
\begin<<2>>document<<2>>
\section<<3>>test<<3>>

This is a list of two items:

\begin<<4>>itemize<<4>>

```

```

\item<<5>>First item<<5>>
\item<<6>>Second item<<6>>
\end<<7>>itemize<<7>>

<tex2html_verbatim_mark>verbatim1

\end<<8>>document<<8>>

```

Next, the document is split into sections. The L<sup>A</sup>T<sub>E</sub>X sectioning commands `\chapter`, `\section`, `\subsection`, etc. work as search-patterns used to split the document into items in an perl array. In our example, the conversion is configured to create a single document (i.e., no splitting).

For each section, the conversion rules are applied. These rules are implemented as procedures that have names like `do_env_X` or `do_cmd_X`, depending on whether one is dealing with a L<sup>A</sup>T<sub>E</sub>X environment or command, where `X` stands for either the environment or command name. For instance, our example document includes an `itemize` environment, and `LaTeX2HTML` will thus call the perl procedure `do_env_itemize`, that will receive as its parameter the contents of the environment, and will then parse that information.

Similarly a procedure `do_cmd_chapter` exists for converting a chapter command, and so on for the other sectioning commands. The resulting document after applying these conversion rules looks as follows.

```

<H1><A NAME=SECTION00100000000000000000> test</A></H1>
This is a list of two items:
<UL>
  <LI><#5#>First item<#5#>
  <LI><#6#>Second item<#6#>
</UL>
<tex2html_verbatim_mark>verbatim1

```

After this each document is enhanced with headers and navigation tools.

```

<!DOCTYPE HTML PUBLIC "-//W30//DTD W3 HTML 2.0//EN">
<!Converted with LaTeX2HTML 95.1 (Fri Jan 20 1995) by Nikos
Drakos (nikos@cbl.leeds.ac.uk), CBLU, University of Leeds >
<HEAD>
<TITLE> test</TITLE>
</HEAD>
<BODY>
<meta name="description" value=" test">
<meta name="keywords" value="example">
<meta name="resource-type" value="document">
<meta name="distribution" value="global">
<BR>
<HR>
<A NAME=tex2html13 HREF="node2.html">
  <tex2html_next_page_visible_mark></A>

```

```

<A NAME=tex2html11 HREF="example.html">
  <tex2html_up_visible_mark></A>
<A NAME=tex2html5 HREF="example.html">
  <tex2html_previous_page_visible_mark></A>
<BR>
<B> Next:</B> <A NAME=tex2html14 HREF="node2.html">About this document</A>
<B>Up:</B> <A NAME=tex2html12 HREF="example.html">No Title</A>
<B> Previous:</B><A NAME=tex2html6 HREF="example.html">No Title</A>
<BR>
<HR>
<P>
<H1><A NAME=SECTION00100000000000000000> test</A></H1>
This is a list of two items:
<UL><LI><#5#>First item<#5#>
  <LI><#6#>Second item<#6#>
</UL>
<tex2html_verbatim_mark>verbatim1
<BR>
<HR>

```

Finally, the markers are replaced with the contents to which they point. Extraneous tags are removed and the address of the author is appended to the file.

```

<!DOCTYPE HTML PUBLIC "-//W3O//DTD W3 HTML 2.0//EN">
<!Converted with LaTeX2HTML 95.1 (Fri Jan 20 1995) by Nikos
Drakos (nikos@cbl.leeds.ac.uk), CBLU, University of Leeds >
<HEAD>
<TITLE> test</TITLE>
</HEAD>
<BODY>
<meta name="description" value=" test">
<meta name="keywords" value="example">
<meta name="resource-type" value="document">
<meta name="distribution" value="global">
<P>
<BR>
<HR>
<A NAME=tex2html13 HREF="node2.html">
  <IMG ALIGN=BOTTOM ALT="next"
    SRC="http://asdwww.cern.ch/icons/next_motif.gif"></A>
<A NAME=tex2html11 HREF="example.html">
  <IMG ALIGN=BOTTOM ALT="up"
    SRC="http://asdwww.cern.ch/icons/up_motif.gif"></A>
<A NAME=tex2html5 HREF="example.html">
  <IMG ALIGN=BOTTOM ALT="previous"
    SRC="http://asdwww.cern.ch/icons/previous_motif.gif"></A>
<BR>
<B> Next:</B> <A NAME=tex2html14 HREF="node2.html">About this document</A>
<B>Up:</B> <A NAME=tex2html12 HREF="example.html">No Title</A>
<B> Previous:</B> <A NAME=tex2html6 HREF="example.html">No Title</A>
<BR>
<HR>
<P>
<H1><A NAME=SECTION00100000000000000000> test</A></H1>
<P>
This is a list of two items:
<UL><LI>First item

```

```

    <LI>Second item
  </UL>
  <P>
  <PRE>This section includes some special
  characters such as $, &lt;, &gt;, _ .
  </PRE>
  <P>
  <BR> <HR>

```

## C.2 Enhancing the translator

From the previous section it is evident that the way to handle user commands and environments is to add perl code into the system or personal configuration files, as also discussed in Section 3.5. One can include as well a file with new definitions on the command line using the `-init_file` option.

To give a taste of how commands and environments are handled by L<sup>A</sup>T<sub>E</sub>X2HTML, we provide a few simple examples that nevertheless clearly show the powerful techniques used to generate HTML documents that preserve the information present in the original L<sup>A</sup>T<sub>E</sub>X document.

Let us first consider a L<sup>A</sup>T<sub>E</sub>X command (`\Ucom`) used to tag commands that have to be typed by the user on the keyboard. A possible definition using the HTML tag `<KBD>` for keyboard input is:

```

sub do_cmd_Ucom {
    local($_) = @_;
    s/$next_pair_pr_rx//o;
    join(' ', qq+<KBD>$$&</KBD>+, $_);
}

```

The perl variable `$next_pair_pr_rx` contains the substitution pattern that extracts the string of characters surrounded by the following pair of delimiters. The string of characters and the delimiters are eliminated and the string is then copied between the HTML `<KBD>` and `</KBD>` appended to the output stream.

Similarly, one can translate the argument of a `\URL` command (containing a Universal Resource Locator) into an HTML anchor, as shown below:

```

sub do_cmd_URL {
    local($_) = @_;
    s/$next_pair_pr_rx//o;
    join(' ', "<a href=\"$$\">$$</a>", $_);
}

```

This procedure creates a link to the specified URL by returning an anchor with the URL as its target and an anchor description along with the rest of the as yet unprocessed document.

Our next example shows an enumerated list `EnumZW` of a special type whose “numbers” are icons available on a www server. The name of the icon depends on the value

of the `perl` variable `count`, which is incremented for each `\item` command used inside the `EnumZW` environment. Everything takes place inside an HTML description list `<DL>`.

```
sub do_env_EnumZW {
    local($_) = @_;
    local($count) = 0;
    s|\\item|do {++$count; qq!<DT><IMG ALIGN=TOP ALT=""
SRC="http://somewhere/icons/circled$count.xbm"><DD>!}|eog;
    "<DL COMPACT>$_</DL>";
}
```

Two or more arguments can also be handled graciously, as shown by the following two commands, which have two and three arguments, respectively, and are typeset by L<sup>A</sup>T<sub>E</sub>X as follows:

```
\Command{arg1}
```

```
\Command[arg1]{arg2}
```

The translation in `perl` is straightforward, since one must merely extract the relevant arguments from the input stream, one after the other.

```
sub do_cmd_BDefCm { # \BDefCm{Command}{arg1}
    local($_) = @_;
    s/$next_pair_pr_rx//o; $command = $&;
    s/$next_pair_pr_rx//o; $mandatory1 = $&;
    join(' ', "<strong>\\$command\\{$mandatory1}</strong>", $_);
}

sub do_cmd_BDefCom { # \BDefCom{Command}{arg1}{arg2}
    local($_) = @_;
    s/$next_pair_pr_rx//o; $command = $&;
    s/$next_pair_pr_rx//o; $optional1 = $&;
    s/$next_pair_pr_rx//o; $mandatory1 = $&;
    join(' ', "<strong>\\$command\\{$optional1\\}\\{$mandatory1}</strong>", $_);
}
```

Explaining all this `perl` code would lead us a little too far, but it should be fairly clear by now that before trying to develop new code for L<sup>A</sup>T<sub>E</sub>X2HTML it is a good idea to study in detail the way Nikos Drakos coded his program, not only in order to write `perl` code compatible with his conventions, but also as a source of inspiration for one's own extensions. Below we show definitions for frequently-occurring regular expressions in the L<sup>A</sup>T<sub>E</sub>X2HTML `perl` code.

```
$delimiters = '\[\s[\]\{\}\<>().,#;:\|/!-~';
$delimiter_rx = "([\s$delimiters])";

# $1 : br_id
# $2 : <environment>
$begin_env_rx = "[\s*\begin\s*%0(\d+)%C\s*([\s$delimiters]+)\s*%0\1%C\s*";

$match_br_rx = "\s*%0\d+%C\s*";
```

```

$optional_arg_rx = "\s*\[[^\]]+\]"; # Cannot handle nested []s!

# Matches a pair of matching brackets
# $1 : br_id
# $2 : contents
$next_pair_rx = "[\s%]*\d+\$C([\s\S]*)\d+\$C";
$any_next_pair_rx = "\d+\$C([\s\S]*)\d+\$C";
$any_next_pair_rx4 = "\d+\$C([\s\S]*)\d+\$C";
$any_next_pair_rx5 = "\d+\$C([\s\S]*)\d+\$C";

# $1 : br_id
$begin_cmd_rx = "\d+\$C";

# $1 : largest argument number
$tex_def_arg_rx = "#[0-9]*#[0-9]\$0";

# $1 : declaration or command or newline (\)
$cmd_delims = q|-#,\./\''^=|; # Commands which are also delimiters!
# The tex2html_dummy is an awful hack ...
$single_cmd_rx = "\\\\[($cmd_delims)|[$delimiters]+|\\\\(tex2html_dummy)";

# $1 : description in a list environment
$item_description_rx =
  "\\\item\s*[\s]*((($any_next_pair_rx4)|([\[\]]*[\]])|[\^])*)[\]]";

$fontchange_rx = 'rm|em|bf|it|sl|sf|tt';

# Matches the \caption command
# $1 : br_id
# $2 : contents
$caption_rx = "\\\caption\s*([\s]*((($any_next_pair_rx5)|([\[\]]*[\]])|[\^])*)[\]])?\d+\$C([\s\S]*)\d+\$C";

# Matches the \htmlimage command
# $1 : br_id
# $2 : contents
$htmlimage_rx = "\\\htmlimage\s*\d+\$C([\s\S]*)\d+\$C";

# Matches a pair of matching brackets
# USING PROCESSED DELIMITERS;
# (the delimiters are processed during command translation)
# $1 : br_id
# $2 : contents
$next_pair_pr_rx = "[\s%]*\d+\$CP([\s\S]*)\d+\$CP";
$any_next_pair_pr_rx = "\d+\$CP([\s\S]*)\d+\$CP";

# This will be used to recognise escaped special characters as such
# and not as commands
$latex_specials_rx = '\[$]|&|%|#|{|}|_';

# This is used in sub revert_to_raw_tex before handing text to be processed by latex.
$html_specials_inv_rx = join("|", keys %html_specials_inv);

# This is also used in sub revert_to_raw_tex
$iso_latin1_character_rx = '(&\d+;)'

# Matches a \begin or \end {tex2html_wrap}. Also used by revert_to_raw_tex
$tex2html_wrap_rx = '\[\\\\(begin|end)\s*{\s*tex2html_wrap[_a-z]*\s*}';

$meta_cmd_rx = '\[\\\\(renewcommand|renewenvironment|newcommand|newenvironment|newtheorem|def)';

# Matches counter commands - these are caught early and are appended to the
# file that is passed to latex.
$counters_rx = "\[\\\\(newcounter|addtocounter|setcounter|refstepcounter|stepcounter|".
  "arabic|roman|Roman|alph|Alph|fnsymbol)\$delimiter_rx";

# Matches a label command and its argument
$labels_rx = "\[\\\\label\s*\d+\$C([\s\S]*)\d+\$C";

# Matches environments that should not be touched during the translation
$verbatim_env_rx = "\s*{(verbatim|rawhtml|LVerbatim)[*]?";

# Matches icon markers

```

```

$icon_mark_rx = "<tex2html(" . join("|", keys %icons) . ")>";

# Frequently used regular expressions with arguments
sub make_end_env_rx {
  local($env) = @_;
  $env = &escape_rx_chars($env);
  "[\\]end\\s*$0(\\d+)$C\\s*$env\\s*$0\\1$C";
}
sub make_begin_end_env_rx {
  local($env) = @_;
  $env = &escape_rx_chars($env);
  "[\\](begin|end)\\s*$0(\\d+)$C\\s*$env\\s*$0\\2$C(\\s*$)?";
}
sub make_end_cmd_rx {
  local($br_id) = @_;
  "$0$br_id$C";
}
sub make_new_cmd_rx {
  "[\\]((" . join("|", keys %new_command) . ")")
}
if each %new_command;
}
sub make_new_env_rx {
  local($where) = @_;
  $where = &escape_rx_chars($where);
  "[\\]$where\\s*$0(\\d+)$C\\s*((" .
    join("|", keys %new_environment) .
  ")\\s*$0\\1$C\\s*"
    if each %new_environment;
}
sub make_sections_rx {
  local($section_alts) = &get_current_sections;
  # $section_alts includes the *-forms of sectioning commands
  $sections_no_delim_rx = "\\(\\($section_alts)";
  $sections_rx = "\\(\\($section_alts)$delimiter_rx"
}
sub make_order_sensitive_rx {
  local(@theorem_alts, $theorem_alts);
  @theorem_alts = ($preamble =~ /\ntheorem\s*([^\s]+)/og);
  $theorem_alts = join('|', @theorem_alts);
  $order_sensitive_rx =
    "(equation|equarray|caption|ref|counter|\\the|\\stepcounter" .
    "\\arabic|\\roman|\\Roman|\\alpha|\\Alpha|\\fnsymbol)";
  $order_sensitive_rx = " s\\)/|$theorem_alts|/ if $theorem_alts;
}
sub make_language_rx {
  local($language_alts) = join("|", keys %language_translations);
  $setlanguage_rx = "\\setlanguage{\\($language_alts)}";
  $language_rx = "\\(\\($language_alts)TeX";
}
sub make_raw_arg_cmd_rx {
  # $1 : commands to be processed in latex (with arguments untouched)
  $raw_arg_cmd_rx = "\\(\\(" . &get_raw_arg_cmds . ")($delimiters|+|\\|\\#|\\$)";
}
# Creates an anchor for its argument and saves the information in the array %index;
# In the index the word will use the beginning of the title of
# the current section (instead of the usual pagenumber).
# The argument to the \index command is IGNORED (as in latex)
sub make_index_entry {
  local($br_id,$str) = @_;
  # If TITLE is not yet available (i.e the \index command is in the title of the
  # current section), use $ref_before.
  $TITLE = $ref_before unless $TITLE;
  # Save the reference
  $str = "$str###" . ++$global{'max_id'}; # Make unique
  $index{$str} = &make_half_href("$CURRENT_FILE#$br_id");
  "<A NAME=$br_id>$anchor_invisible_mark</A>";
}
}

```

## Appendix D: Technical details of the MATH2HTML program

### D.1 Different approaches

Various people have approached the problem of translating L<sup>A</sup>T<sub>E</sub>X into SGML or HTML using different programming paradigms. Joachim Schrod of the Technical University of Darmstadt, Germany has written a lisp parser for T<sub>E</sub>X code which can also be used for conversions [6].<sup>19</sup> As already discussed in Section 5, Xavier Leroy used Cam1 to achieve the same goal, while LaTeX2HTML uses perl (other approaches based on sgm1s also use that language).

Common to all approaches, whether using a procedural or a functional language, is the basic implementation. A lexer is used to recognize tokens from the input, a parser to create an internal representation and the conversion process produces the wanted output.

The major difference between functional and procedural languages is the way a language such as T<sub>E</sub>X can be parsed. Since the T<sub>E</sub>X language can at any point in the input define new rules for delimiters and symbols, the program parsing this input should also be able to cope with these dynamic features. Functional programming languages can do this by their nature, easily introducing new rules to the parser at runtime. This is what the parser written by Joachim Schrod can do. In comparison this cannot easily be done with a fixed grammar inside a parser.

Xavier Leroy's translator resembles a bison<sup>20</sup> input file. It sees groups of tokens and reduces the stacked input by given BNF-like rules. When it reduces the tokens it produces HTML output for L<sup>A</sup>T<sub>E</sub>X counterparts.

### D.2 Implementation of the Translator

The math2html program, written in C++, takes L<sup>A</sup>T<sub>E</sub>X mathematics input, parses it and converts it into HTML3 mathematics (if possible). The program consists of the following components:

- flex, a fast lexical analyzer generator;
- bison, a parser generator;
- C++ code.

The parsing of L<sup>A</sup>T<sub>E</sub>X source code is, however, non-trivial, since its grammar has been developed step-by-step to cope with all L<sup>A</sup>T<sub>E</sub>X syntactical notations. The basic mathematical notation is presented here in detail.

<code>\[...]</code>	Display mathematics.
<code>txt1 \$...\$ txt2</code>	Inline mathematics.
<code>{abc}</code>	Characters a, b and c are grouped into one.

<sup>19</sup> The system is available at URL <ftp://ftp.th-darmstadt.de/pub/tex/src/et1s/>.

<sup>20</sup> Bison is a parser generator in the style of yacc.

<code>\abc</code>	Characters a, b and c are a control sequence.
<code>a^b</code>	Superscripts (b can be a group of characters).
<code>a_b</code>	Subscripts (b can be a group of characters). Superscripts and subscripts can be nested.

The lexical analyser recognizes L<sup>A</sup>T<sub>E</sub>X primitives by generating tokens for the parser. A control sequence, plain text, superscript, subscript, begingroup, endgroup, fraction, array, column separators and end of row are examples of typical tokens. These tokens correspond to classes. These classes are depicted in Figure 17 with the object modeling technique (OMT) [5].

The class library presents the supported structures of L<sup>A</sup>T<sub>E</sub>X mathematics as sums, integrals, fractions, plain input, sequences and groups. These are currently the only primitives which can be reasonably converted into HTML3 mathematics. A few examples of basic primitives that can be treated by `math2html` are shown below:

Sum:	<code>\sum_{i=1}^n i</code>	Integral:	<code>\int_0^1 f(x) dx</code>
Fraction:	<code>\frac{1}{n}</code>	Sequence:	<code>\infty</code>
Group:	<code>{ x+1 }^2</code>		
Table:	<code>\begin{table}{lr}</code> <code>a &amp; b \ c &amp; d</code> <code>\end{table}</code>	Eqntable:	<code>\begin{eqnarray}</code> <code>y=&amp;x^2 \ z&lt;=&amp; x^3</code> <code>\end{eqnarray}</code>

The parser analyzes the tokens using an ad-hoc BNF grammar generated specifically to parse L<sup>A</sup>T<sub>E</sub>X code. When reducing the input according to the grammar rules, the parser generates instances of C++ classes (see Figure 17), which correspond to these L<sup>A</sup>T<sub>E</sub>X primitives. Once the whole input has been parsed, the internal representation is linked together so that all these instances can be reached from one top-level list.

The conversion is implemented by calling a conversion method to each instance in the list. Each primitive knows how to convert itself and also propagates the conversion to all its children nodes.

An instance of the runtime organization of the parsing tree corresponding to the example of Figure 11 is shown in Figure 18 on the next page.

### D.3 Mapping of control sequences

Since the wide variety of different control sequences is quite impossible to hardcode into the program, an external configuration file is read every time the program starts. The mapping between control sequences and HTML3 counterparts is read into a hash table and in this way the user can configure the program to cope with special control sequences not natively supported by the converter. An example of this is the Particle Entity Notation scheme [2], a set of standard control sequences for representing elementary particles. This naming scheme consists of about 240 control sequences and



their presentation counterparts. The configuration file maps each control sequence into its HTML3 counterpart using the following format:

<code>\Pgppm</code>	<code>&amp;pi ; &lt;sup&gt;&amp;plusmn ; &lt;/sup&gt;</code>	<code>\Pgpz</code>	<code>&amp;pi ; &lt;sup&gt;0 &lt;/sup&gt;</code>
<code>\Pgh</code>	<code>&amp;eta ;</code>	<code>\Pgr</code>	<code>&amp;rho ; (770)</code>
<code>\Pgo</code>	<code>&amp;omega ; (783)</code>	<code>\Pghpr</code>	<code>&amp;eta ; ' (958)</code>
<code>\Pfz</code>	<code>&lt;t&gt;f &lt;/t&gt; &lt;sub&gt;0 &lt;/sub&gt; (975)</code>		

#### D.4 Program heuristics

The program uses a few heuristics in order to be able to parse L<sup>A</sup>T<sub>E</sub>X code successfully. If these coding rules are not used, parsing may fail.

Optional parameters specified between square brackets ( `[]` ) after a control sequence are not parsed with respect to the control sequence. Therefore, there should be no space left between the control sequence and the opening bracket where optional parameters are used. Space should be left if the brackets are used as delimiters. An example is the difference between the following two control sequences:

<code>\root[3]{\pi}</code>	<code>\left [ \pi+2 ]</code>
----------------------------	------------------------------

It is also worth noticing that all control sequences not supported primitively in `math2-html`, apart from integrals, fractions, roots, sums and a few others, are dropped out during the conversion, for example, no text is produced in the HTML3 version. The only way to convert them is to create specific code or map it in the configuration file.

#### D.5 Interfacing with other programs

This application was built to make it easy for other applications to call it. The program can either be compiled into a single executable program with a command line interface or into a library that can be linked with any other applications.

The modular approach has the advantage of being both simple and straightforward. The object-oriented implementation makes the linearisation of the internal representation almost effortless and eases the future addition of new HTML3 primitives by the user. The program is quite flexible and, as pointed out above, can be used in different contexts: embedded or stand-alone.

#### D.6 Drawbacks of the presented solution

The end-user may find extending the program too difficult, especially if one has no experience with `flex`, `bison`, or C++. The configuration file that comes with the program provides an easy way to do simple mappings, but if one wants to add more functionality, one must understand the organization of the program.

As trickier tables and equations need to be converted, the program will need extension for analyzing the internal tree structure and to add, modify or delete specific nodes.

If the L<sup>A</sup>T<sub>E</sub>X input code uses low-level T<sub>E</sub>X commands the program will not be able to handle the input.

## Appendix E: Using the CAML system for translating L<sup>A</sup>T<sub>E</sub>X to HTML

The program works by expressing the L<sup>A</sup>T<sub>E</sub>X grammar in a YACC-like format and parsing the L<sup>A</sup>T<sub>E</sub>X input lines rule by rule, converting all recognized patterns into HTML. An example of Caml Light grammar rules for L<sup>A</sup>T<sub>E</sub>X to HTML conversion is given below.

(\* Font changes \*)

```
| "\\it" | "\\em"
    { print_string "<i>"; upto '}' main lexbuf;
      print_string "</i>"; main lexbuf }

| "\\bf"
    { print_string "<b>"; upto '}' main lexbuf;
      print_string "</b>"; main lexbuf }

| "\\tt"
    { print_string "<tt>"; upto '}' main lexbuf;
      print_string "</tt>"; main lexbuf }

| "' '
    { print_string "<tt>"; indoublequote lexbuf;
      print_string "</tt>"; main lexbuf }
```

(\* Verb, verbatim \*)

```
| "\\verb" _ { verb_delim := get_lexeme_char lexbuf 5;
               print_string "<tt>"; inverb lexbuf;
               print_string "</tt>"; main lexbuf }

| "\\begin{verbatim}"
    { print_string "<pre>"; inverbatim lexbuf;
      print_string "</pre>"; main lexbuf }
```

Unlike LaTeX2HTML the program does not pass mathematics on to the T<sub>E</sub>X engine in order to create bitmap images for unparsable input, but produces plain text only. As the L<sup>A</sup>T<sub>E</sub>X control sequences recognized by the program are read from a separate file, the addition of new commands and their HTML counterparts is relatively easy. An example of such mappings is the following:

```
def "\\chapter" [Print "<H1>"; Print_arg; Print "</H1>\n"];
```

```
def "\\chapter*"          [Print "<H1>"; Print_arg; Print "</H1>\n"];

def "\\begin{itemize}"    [Print "<p><ul>"];
def "\\end{itemize}"      [Print "</ul>"];

def "\\begin{enumerate}"  [Print "<p><ol>"];
def "\\end{enumerate}"    [Print "</ol>"];

def "\\begin{description}" [Print "<p><dl>"];
def "\\end{description}"  [Print "</dl>"];

def "\\begin{center}"     [Print "<blockquote>"];
def "\\end{center}"       [Print "</blockquote>"];
```

The use of this program requires the compilation of the Caml Light distribution, available for a variety of platforms. The language is compiled with an intermediate step in the C language. The executable program suffers from some overhead, mainly affecting execution time.

Because the program does not deal with mathematics and tables, it can only be used for a restricted set of documents. To be useful for the general user it will have to be extended to convert mathematics and tables either into bitmaps or into HTML3.

# ~~pa~~ ~~al~~sc: formatting Pascal using T<sub>E</sub>X

Pedro Palao Gostanza and Manuel Núñez García

Departamento de Informática y Automática

Universidad Complutense de Madrid

gostanza@eucmax.sim.ucm.es, manuelnu@eucmvx.sim.ucm.es

## Abstract

This paper is based on our ideas about how a system which formats programs written in a structured language must work. Particularly, tools which help in typesetting texts where algorithms are described. Most of our ideas have been put in practice in the ~~pa~~<sup>pa</sup>sc system, which automatize the elegant layout of Pascal programs. This system is programmed as a T<sub>E</sub>X macro package.

## 1 Introduction

Almost every programming language have a structured syntax, and usually, there are several standard ways for the layout of programs in these languages. Both facets accomplish the same goal: programs must be easily understood. But, while the first one is used in order to facilitate the task to the compiler, the second one is used exclusively<sup>1</sup> in order to facilitate the comprehension of programs to readers.

During the history of programming languages, two variants in the representation of programs have been developed. First, programmers in a given language usually organize their programs in a similar way. Then, it is easier to read programs written by other people, and this fact gives rise to the development of particular modes for text editors like Emacs, which partially formats programs while writing them. On the other hand, a typesetting tradition has been developed for presenting programs in books or journals, which usually must have a very important aesthetic component.

There have been programs which partially solve these problems. For example, most programming languages environments have a pretty-printer program. Some systems, like WEB, go one step beyond extending the programming language so that it is possible to

---

1. This is not true in some languages, like Occam or Miranda, in which written representation prevails over syntax.

mix texts and codes in the same program. Then, the compiler sorts codes and texts out, obtaining a correct program in the given language, and a file containing the documentation ready to be processed by  $\text{\TeX}$ . In this documentation, codes appears very well formatted.

Our experience mixing  $\text{\TeX}$  and programs (writing class exercises, our own papers, or reading papers written by other people) says that, most of the times, a *verbatim mode* is used. As a matter of fact, if the program is not very large, it is customary to format it *by hand* using different type styles. We disagree with both solutions. The main advantage of using a verbatim mode is the simplicity and clearness of the source code, and the similitude between the result and what it is given to the compiler. Nevertheless, visual quality of this result is very poor. On the other side, formatting programs by hand is very hard, error prone, and it is rather difficult to understand the final code. We think that so many programs are presented in these two forms because there do not exist tools which format programs as they usually appear in papers or in lecture notes. These programs have the following characteristics:

- Programs are split in several fragments, usually unordered.
- Each of these fragments is not necessarily complete.
- In these programs, notation which does not belong to the programming language appears (either mathematical one or natural language).
- Programs are very related to the text around them, and thus, it is not so convenient to *input* them, but it is preferably to *paste* programs in the text.

We planned to develop a tool which helps us to write Pascal programs with these characteristics. The last item gives rise to two different alternatives: a preprocessor, or a  $\text{\TeX}$  macro package. The first option has been widely used in the  $\text{\TeX}$  world as well as in the pretty-printers world. It is enough to write a program which leaves the text without any change, adding the adequate  $\text{\TeX}$  indications for typesetting programs. The latter task is more difficult than usual, because of the first three previous items, but it is not substantially new.

While an adequate tool dealing with these items helps in the preparation of texts where programs are a fundamental part, it still has the same important problem of the usual solutions (either formatting by hand or using verbatim mode): a complete and explicit indication of all format aspects instead of *declaration*. This is a similar problem to that which appears when writing big documents using  $\text{\TeX}$  without macros that organize the final result from a logical point of view. For example, a change in the indentation of a language component (e.g. the indentation of a **while** clause) would oblige to correct previous programs trying to fix it. From the reader's point of view, there exist another big problem: a well organized text can be easily understood, even if the final layout is not standard, but a reader would hardly understand a program formatted slightly away from his own style. This shows the importance of a declarative format, where final presentation of programs depends on some *mayor modes* and on a small set of explicit

```
\Program \Hello( \input, \output );  
\Begin \WriteLn( 'Hello word!' ) \End.  


---

program Hello (input, output);  
begin  
  WriteLn('Hello word!')  
end. {End of program}
```

Figure 1: The first program scheme

indications. If one wants to read a text containing programs, it is enough to choose his favorite mode and to process it again.

In this paper we present a system which automatically formats Pascal programs, fulfilling the previous requirements. This system is called  $\frac{pa}{al}sc^2$  and it is entirely programmed in  $\TeX$ . Developing  $\frac{pa}{al}sc$ , we have experimented most of our previous ideas. In fact, we built prototypes for Pascal and for Modula-2. At last, we decide to implement a complete version for Pascal because it always stated more difficult problems, and there exist several format styles for Pascal which are quite different.

$\frac{pa}{al}sc$  is fully declarative. It has three explicit format *hints*, which change the default option of the system. Although we only have implemented one format mayor mode, we will show how other modes can be simulated using these explicit hints. This shows us that these format hints are enough expressive and that more modes may be easily added to the system, just by simulating them internally.

## 2 Basic usage

$\TeX$  recognizes the beginning of a piece of program by the control word `\beginPascal`, while the program must be finished with the control word `\endPascal`. In  $\LaTeX$  version, there exist an environment called `pascal`. We will call the piece of program that appears between these two control words a *program scheme*, because it does not need to be neither a complete program nor an acceptable program by a Pascal compiler.

In the following, we will show program schemes together with its the final layout (as they are formatted by  $\frac{pa}{al}sc$ ) using the following convention: program schemes in `\tt` font and bellow, separated by a rule, the result of processing this program scheme with  $\frac{pa}{al}sc$ . See figure 1 for an example. Some important characteristics of  $\frac{pa}{al}sc$  appear in the simple program of this figure. A correct Pascal program is almost a correct program

---

2. The name is chosen in order to remark the formatting aspect of the system parameterized by the used programming language.

scheme, but it is necessary to add the character `\` before each symbol (reserved words or identifiers) converting it in a control sequence. In order to understand why this addition is mandatory, it is enough to know how `paalsc` internally works. There is no parser which formats a program scheme, but each of the elements of the program performs certain local actions, contributing to the final result. Reserved words usually carry out decisions, like changing indentation or breaking a line, while identifiers usually just write themselves with the adequate font. For this reason they need to be control sequences, in order to associate them a `TeX` macro.

Another significant detail of `paalsc` is that reserved words are capitalized. In order to avoid possible interferences with `TeX` internal macros (e.g. `\if` or `\else`). This is not a constraint because, as we show bellow, `paalsc` has an option that chooses the final result (upper case, lower case, etc). But the problem still remains for identifiers. In the previous program, we had a variable called `output`, recognized inside the program by the control sequence `\output`. When `paalsc` finds this sequence, it is redefined as a macro that expands to `"output."` But `\output` is a fundamental register in the pages generation mechanism of `TeX`. If it is activated when this variable is redefined, a very strange error is produced. In order to solve this problem, we allow identifiers to have a format such that even a user who is not a `TeX`nician will be sure that there are not interferences with `TeX`. We decide to provide an special character to begin identifier names. This special character cannot appear neither in correct Pascal identifier names nor in `TeX` control words. Due to the first characteristic, we can detect and delete it from the final result, while due to the second one we can be sure that there are no interferences with macros. The special character is `!`<sup>3</sup> and it must be used exclusively as the first letter of identifiers, because this is the only place where it is deleted. Then, in the previous program we would write `!\input` and `!\output` instead of `\input` and `\output`. Note that `!` is not needed in `\Hello`, because `paalsc` does not define it as a macro. Also, it cannot be used in `\WriteLn`, because this is an identifier introduced automatically with this capitalization by `paalsc`.

Let us remark that while Pascal is case insensitive, `TeX` is case sensitive, and thus some coherence must be kept when writing identifiers along a program.

### 3 Piecemeal programs and options

Let us remember that a program scheme is a *self-contained* piece of a program in the following sense: it can format itself. For example, a simple sentence

```
\WriteLn( 'End of file' )
```

---

```
WriteLn('End_of_file')
```

3. The character `@` may seem more suitable because it is used when defining private macros, but precisely for this reason it is not guaranteed absence of interferences.

a declarations sequence

```
\Var \!x: \Integer;  
\Const \!c = 100;
```

---

```
var x: Integer;  
const c = 100;
```

or structured sentences,

```
\Var \!c: \Char;  
\Repeat  
  \WriteLn( 'Do you want to continue? ' );  
  \ReadLn( \!c );  
\Until (\!c = 'y') \lor (\!c = 'n');
```

---

```
var c: Char;  
repeat  
  WriteLn('Do you want to continue?');  
  ReadLn(c);  
until (c = 'y')  $\vee$  (c = 'n');
```

where all the identifiers are explicitly declared. In the previous examples, we have seen that all of the identifiers are explicitly declared inside the program scheme. This is because pa<sub>al</sub>sc encloses a program scheme in a group, so that all the declarations appearing in this scheme remain until the end of this group.

pa<sub>al</sub>sc provides two methods for writing programs which depend on identifiers that we do not want to introduce explicitly. Both methods are part of the *options* mechanism. Options, enclosed by brackets ([ ]), can appear prefixing a program scheme. Particularly, there exists a family of options to declare identifiers which are used in the subsequent program scheme. In the program

```
[\var\!power\!x\!y;  
\!power := 1;  
\While \!y \not= 0 \Do  
  \Begin  
    \!power := \!power * \!x; \!y := \!y - 1  
  \End
```

---

```
power := 1;  
while y  $\neq$  0 do begin  
  power := power * x;  
  y := y - 1  
end
```

we have *declared* the variables `\!power`, `\!x` and `\!y`. The list of identifiers declaration options is: `\var`, `\type`, `\const`, `\proc`, `\func`, `\pseudoVar` and `\field`. This options are used as `\var` in the example: prefixing some control sequences without separation between them, finishing with a semicolon. The first five options correspond to the usual Pascal identifiers. Option `\pseudoVar` introduces a identifier name representing a function name, and it is necessary for representing an isolated function body. Option `\field` introduces record field names.

These options are only useful if the different pieces of code are not related among them. But usually, the same identifier is used in different pieces, and in a grouped form (e.g. a data structure with types and operations). <sup>pa</sup><sub>al</sub>sc introduces the concept of *declarations set*. A declarations set is an object that records the declarations of a program scheme, allowing that these declarations may be used in another program scheme. For example, let us suppose that one wants to write a function which calculates the number of nodes of a binary tree. First, the type must be introduced

```
[\newDecls{treedec}\memoDecls{treedec}\type\!Element\!TreeNode;]
\Type \!Tree = ^\!TreeNode;
  \!TreeNode = \Record
    \!elem: \!Element;
    \!left, \!right: \!Tree
  \End;
```

---

```
type Tree = ↑TreeNode;
  TreeNode = record
    elem: Element;
    left, right: Tree
  end;
```

The first option, `\newDecls`, creates a new declarations set called `treedec`. The second one indicates that we want to record in `treedec` all the declarations appearing from this point until the end of the program scheme. Particularly, `treedec` records declarations given by the third option which introduces types “Element” and “TreeNode” which is used before it is declared.<sup>4</sup> Briefly, the second line of options indicates that reserved words are presented with capital letters and that lines are numbered starting with 1. Following with our example, the function “nodes” is

```
[\useDecls{treedec}\memoDecls{treedec}]
\Function \!nodes( \!t: \!Tree ): \Integer;
function nodes( t: Tree ): Integer;
```

---

4. <sup>pa</sup><sub>al</sub>sc does not deal with recursion in pointer types, but it is not very complicated to fix it.

`\useDecls` allows to use in this program scheme the identifiers recorded in `treedec`. The second option adds to `treedec` the declarations appearing in this program scheme. Then, the function body is

```
[\useDecls{treedec}\var\!t;\pseudoVar\!nodes;]
\If \!t = \Nil \Then \!nodes := 0\>
\Else \!nodes := \!nodes( \!t^\!left ) + \!nodes( \!t^\!right );


---


if t = Nil then nodes := 0
else nodes := nodes(t.left) + nodes(t.right);
```

which presents an example where the option `\pseudoVar` is necessary. Finally, an example using the function “nodes” is

```
[\useDecls{treedec}\var\!t;
 \noMarkStringSpaces]
\WriteLn( 'Number of nodes in tree :', \!nodes(\!t) );


---


WriteLn('Number of nodes in tree :', nodes(t));
```

As it is shown in this example, it is usual to use `\newDecls` or `\useDecls` preceding `\memoDecls`. The option `\decls` produces one of these two sequences depending if the declarations set already exists. In fact, `\decls{name}` is equivalent to `\newDecls{name}\memoDecls{name}` if `name` has not yet been declared, and otherwise it is equivalent to `\useDecls{name}\memoDecls{name}`.

Using declarations set, it is very easy to change the capitalization of the predefined identifiers:

```
\beginPascal[
 \decls{predefined}
 \const\!nil\!true\!false;
 \type\!integer\!boolean\!real\!char\!text;
 \proc\!write\!writeln\!read\!readln\!new\!dispose;
 \func\!succ\!pred\!sqr\!sqrt;
]\endPascal
```

## 4 Layout hints

Previous examples show the *pa*<sub>al</sub>sc default formatting mode. Possibly, this is not a very standard style and, as we suggest in the introduction, it is difficult to understand programs when the reader is not used to this style. But this is not a problem for the philosophy behind *pa*<sub>al</sub>sc: the reader can choose another mode and recompile the file. Unfortunately, by now, this is the only implemented mode in *pa*<sub>al</sub>sc.

In this section we present the *layout hints*. These elements allow to locally change defaults options but they must be used exclusively in those places where the understanding of the code would improve if it is not presented in the default mode. Anyway, their use must be limited because it is against the declarative form behind <sup>pa</sup>asc.

<sup>pa</sup>asc only has three layout hints. Each of them specifies a kind of operation which is normally used to write Pascal programs: to break a line, to join two lines, and to align to a point. Respectively, they are activated by the three control symbols `\>`, `\<` and `\!`. For example:

```
[\decls{listdec}\type\!Element\!Node;]
\type\! \!List = ^\!Node;
      \!Node =\! \Record\<
                \!value: \!Element;
                \!next: \!List;
      \End;


---


type List = ↑Node;
      Node = record value: Element;
                next: List;
      end;
```

Layout hints work in a coherent form: if a line is broken, then the rest of the text is indented using a value (that depends on the context); if two lines are joined, a small separation is inserted; an alignment only remains in the corresponding context. In short, the layout hints *know* the mechanism of the structured construction of Pascal programs, and thus they are much more *abstract* than formatting by hand.

Using layout hints in a systematic way, other format styles can be obtained. For instance, the previous example presents a very frequent style where type declarations are aligned. Another style appears when these declarations are split:

```
[\useDecls{listdec}]
\type\> \!List = ^\!Node;
      \!Node =\> \Record\<
                \!value: \!Element;
                \!next: \!List;
      \End;


---


type
  List = ↑Node;
  Node =
    record value: Element;
          next: List;
    end;
```

This shows the expressiveness of the chosen layout hints and why it is so easy to add new format styles to ~~pa~~<sub>al</sub>sc.

## 5 Other options

In addition to the definition of identifiers, the ~~pa~~<sub>al</sub>sc options system allows to indicate many aspects of the final result. Below, we summarize some of the most significative options

`\lineNumbers, \noLineNumbers` This option allows (or does not) the numeration of lines.

`\firstLine` The count of lines is made globally. This option has an argument which changes the default value (i.e. 1).

`\cap, \Cap, \CAP` Alternative options indicating the capitalization of reserved words.

`\autoEnd, \noAutoEnd` This option indicate that a message corresponding to the end of functions, procedures or programs appears (or does not).

`\markStringSpaces, \noMarkStringSpaces` These options indicate that blank spaces must be substituted by `\_` or just a space is left.

`\abstractAssign / \textualAssign` Complementary options indicating if assignation is represented either by “:=” or by “←.”

An example showing these features follows:

```

[\useDecls{listdec}
 \global\abstractAssign
 \noAutoEnd\Cap
 \lineNumbers\firstLine{1} ]
\Function \!exists( \!e:\!Element; \!l:\!List ): \Boolean;
 \Var\! \!find: \Boolean; \!aux: \!List;
\Begin
 \!find := \False; \< \!aux := \!l;
 \While \Not\!find \And (\!aux \not=\Nil) \Do\> \Begin
 \!find := \!aux^.\!value = \!e;
 \!aux := \!aux^.\!next
 \End;
 \!exists := \!find
\End;

```

---

```

1 Function exists( e: Element; l: List ): Boolean;
2   Var find: Boolean;
3     aux: List;
4 Begin
5   find ← False;  aux ← l;
6   While Not find And (aux ≠ Nil) Do

```

```

7      Begin
8          find ← aux↑.value = e;
9          aux ← aux↑.next
10     End;
11     exists ← find
12 End;

```

If these options are used together with `\global`, their effects remain in subsequent program schemes. For instance, in the previous example we have declared `\global\abstractAssign`, and thus, all the assignments appearing in the rest of the paper will be denoted by  $\leftarrow$ .

## 6 The fields problem

Nowadays, almost every Pascal compiler allows several record declaration sharing field names. They also allow that field identifiers can be used to denote other objects. <sup>pa</sup><sub>al</sub>sc also allows this. For example, we can define a function computing the left subtree of a tree:

```

[\decls{treedec}]
\Function \!left( \!t: \!Tree ): \!Tree;
\Begin
  \!left := \!t^\!left;
\End;

```

---

```

function left( t: Tree): Tree;
begin
  left ← t↑.left;
end; {End of left function}

```

In the left hand side of the assignment, “left” denotes a function, while in the right hand side, “*left*” denotes the field of the record implementing the type “Tree.”

<sup>pa</sup><sub>al</sub>sc can distinguish from the context among the different uses of an identifier which is simultaneously used as a record field and as another object. Nevertheless, <sup>pa</sup><sub>al</sub>sc is more limited than a Pascal compiler, because it does not keep type informations. This problem appears when using the **with** sentence:

```

[\useDeclS{treedec}\useDeclS{listdec}\var\!l\!t;]
\With \!l \Do
  \!value := \!value + \!nodes(\!left(\!t));

```

---

```

with / do value ← value + nodes(left(t));

```

In the previous example, pa<sub>al</sub>sc has misunderstood the call to the function “left” for a use of the field “left.” In general, whenever pa<sub>al</sub>sc analyzes a **with** command, it does not use the type information of the expression associated with the **with**. All the identifiers associated with field declarations will expand as a field, without taking care of the possible association with another object. In order to indicate pa<sub>al</sub>sc that some symbol does not represent a field, one must prefix it with \), which indicates a *local closure* of a **with**. For example, the previous program would be

```
[\useDecls{treedec}\useDecls{listdec}\var\!l\!t;]
\With \!l \Do
  \!value := \!value + \!nodes()\!left(\!t));


---


with / do value ← value + nodes(left(t));
```

In addition to \), the prefix \(\ *locally opens* a **with**. For example, the previous program, assuming an external **with**, would be

```
[\useDecls{treedec}\useDecls{listdec}\var\!l\!t;]
\(\!value := \(\!value + \!nodes(\!left(\!t));


---


value ← value + nodes(left(t));
```

## 7 Differences with respect to Pascal and T<sub>E</sub>X

There exist two aspects in Pascal syntax which pa<sub>al</sub>sc implements in a slightly different way: comments and subrange types.

Comments are written using the control sequence \Comment, which has two arguments: the first one is an optional dimension (enclosed between squared brackets) and the second is the text.

If the first one is omitted, the text is written using a horizontal box. Otherwise, the optional parameter indicates the horizontal size of a vertical box. For example

```
\Const \!max = \cdots; \Comment{Maximum size of the stack}
\Type \!stack = \!\Record
  \!top: [0..\!max];
  \!data: \Array [1..\!max] \Of \Integer
\End;\< \Comment[7cm]{Invariant: the
  {\it top} field contains the
  index of the last pushed element.}


---


const max = ...; {Maximum size of the stack}
type stack = record
```

```

    top: 0 .. max;
    data: array [1 .. max] of Integer
end;    {Invariant: the top field contains the index
          of the last pushed element.          }

```

In this example, we show another difference with respect to Pascal: subrange types must be enclosed between square brackets, as done in Modula. With this, we allow arbitrary expressions appearing in both range limits:

```

[\const\!a\!b\!c\!d;]
\Type\! \!range = [(\!a+\!b)*(\!c+\!d) .. 100];
    \!colors = (\!red,\!green,\!blue);

```

---

```

type range = (a + b) * (c + d) .. 100;
    colors = (red, green, blue);

```

Without using “[ ]”, an expression beginning with “(” will be taken as an enumeration.

As we have shown along the paper, any *math mode* control sequence can be used inside Pascal expressions. Nevertheless,  $\hat{\phantom{x}}$  and  $\prime$  have been redefined in order to respectively represent an indirection and the beginning of a string. Superscripts appears in Pascal expressions using  $\wedge$ . If one wants to write *primes*, the whole definition must be used, that is  $\wedge\prime$ .

## 8 Conclusions and future work

In this paper we have presented the system  $\text{pa}_{\text{al}}^{\text{sc}}$ , which has been developed for helping in the typesetting of texts where Pascal programs appear.  $\text{pa}_{\text{al}}^{\text{sc}}$  is very operative, and excepting the absence of several modes, fulfills all of our proposed objectives. We have used it to write exercises and it has proved to be very useful when presenting either bottom-up decompositions or top-down ones.

We think that the most important future task is to implement some major modes including the most usual ones. This is relatively easy because of the organization of  $\text{pa}_{\text{al}}^{\text{sc}}$ : most of the code is independent of the format style and the three major Pascal syntactic groups (types, sentences and declarations) are distributed in different files. Then, modes can be chosen by syntactic groups.

Another task is to translate the ideas behind  $\text{pa}_{\text{al}}^{\text{sc}}$  to other programming languages such as Modula-2, Ada or ML. Most of the code related with formatting and the declarations sets is independent of the language, and thus it can be shared.

# Beyond the bounds of paper and within the bounds of screens; the perfect match of T<sub>E</sub>X and Acrobat

J. Hagen

Pragma  
P.O.Box 125  
8000 AC Zwolle  
The Netherlands

## 1 Introduction

T<sub>E</sub>X, both a typographic programming language and a typesetting engine, has some powerful primitives, that enable communication between its internal processes and the outside world. Of these primitives `\special` has proved that, although more than 10 years old, T<sub>E</sub>X will certainly survive the next decades.

With `\special` we can tell other applications what special things we want them to do. It's `\special` that makes T<sub>E</sub>X one of the first systems that supports the new portable document format PDF from Adobe Systems. Adobe supports this format with the Acrobat-family of viewers and conversion programs. PDF is a portable, extensible, readable and programmable page description language. In fact, PDF is a subset of PostScript with hypertext extensions. It has the duality of T<sub>E</sub>X: it's both a programming language and a viewing engine.

At the moment PDF does not have the stability of T<sub>E</sub>X, because it's still under development. This means that future PDF-options can cause trouble with older Acrobat viewers. We think however that within a few years PDF will be stable enough to become one of the most important standards in the distribution of documents. We can still read books that are hundreds of years old. Electronic texts need a descriptive medium that is at least as stable as paper.

One of PDF's characteristics is interactivity. It supports active documents in which we can use all kind of hyperlinks. One does not have to program texts in the PDF-format directly, because it can be generated from PostScript code. The interface between the

hyperlink mechanism and PostScript is provided by the PostScript `pdfmark` operator. This operator accepts arrays of commands, that are unique to PDF.

PDF describes a page in terms of typography and not in terms of structure, like for instance HTML does. Because HTML viewers format on the fly, authors and designers have no guaranties of the typographic quality of their products. With PDF however one has complete control, but the programs that are used to produce portable documents that are highly interactive *have* to support `pdfmarks`.

When Acrobat entered the market, no programs were available that supported the generation of `pdfmarks`. One reason for this is that it's not a trivial task to include verbatim PostScript in texts. Another reason is that the necessary positioning information is not available to users. Depending on the DVI-drivers used,  $\text{\TeX}$  has the means to include verbatim PostScript, i.e. `pdfmarks`. Users of  $\text{\TeX}$  also have all the necessary information at hand. That's why only a few days after Acrobat arrived in fall 1993, we could produce interactive documents with  $\text{\TeX}$ .

Adobe claims that with Acrobat mankind will go *beyond the bounds of paper*. When using  $\text{\TeX}$ , one is accustomed to high quality portable output on paper. We think that Adobe (again) has proven that it is possible to produce high quality portable output on screens too. This is however an area where software is years ahead of hardware: most computers simply are not yet able to show this high quality, especially not portable ones.

Experimenting with  $\text{\TeX}$  and Acrobat learns that new boundaries show up: those imposed by screens. Some we can use to our advantage, some we can't. Both programs enable us to find these boundaries. In this article we will show some results from our experimenting. We highlight some aspects of typesetting that are closely related to portable documents, at least we think they are. We use the terms portable document and interactive text for documents presented on computers. We talk of screen in stead of computers and displays.

Looking at the options supported by `pdfmark` we see that, in version 1.0 (1993) as well as in version 2.0 (1994), many of them are tuned to desktop publishing programs. These programs are page oriented and so are nearly all `pdfmark` options. Our experiments show however that PDF lacks some document oriented options, but we believe they will be supported in future versions.

## 2 Aspect ratios

One of the more prominent differences between paper and screen is the difference in aspect ratio. As a result compatible layouts are nearly impossible: paper has height and screens have width. Long lines can't be read too well, so we can only use the full width of the screen when we use big fonts. This is why in most cases the amount of text on the screen is limited.

The aspect ratio of screens influences the overall esthetics quality of the text. When we have a lot of tables, figures and/or formulas and also use white space between

paragraphs, it's a tough job to get a good layout. Because we don't have enough height (\vsize) available for moving things around, floats do float indeed.

So, while waiting for displays with high resolutions, we have to use small fonts to get things right. When we also pose limits on the width of the text (\hsize), we have a lot of marginspace available. But too much white space doesn't look to well either, especially when it's not used. In the near future some sort of standard has to be defined for the aspect ratio of portable documents. The Personal Data Assistants that are showing up definitely have a different aspect ratio than desktop and portable computers.

### 3 Enhanced pagebody

Because portable documents are not as tactile as books, one needs navigation tools to manoeuvre through the text. Navigation tools are active typographic elements, that are provided by the viewers or, which is preferred, by the document itself. As we will see further on, we need some space to provide these navigation tools, and fortunately the width of screens allows for this. Figure 1 shows us an example of this.

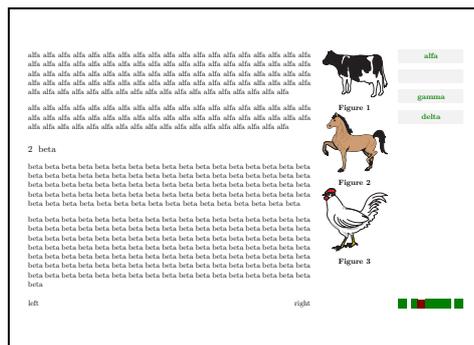


Figure 1: Screen layout

We distinguish three areas: text, margin and border. We can use the margin for marginal notes or floats and the border for navigation tools. Not shown are the left margin and left border area. Even more room is available when we extend the header and footer lines into these areas, as shown. Still more room is available above header and under footer lines, but because the height is already limited, maybe we should avoid to use them. Also not shown, and in most cases not used either, are the areas for company logos. Of course everything is implemented and available.

At the moment we are developing and adapting the relevant macros to support a dual layout, depending on a switch. In an interactive version floats are put in the margins and in a non-interactive version, they are placed as specified.

## 4 Parallel documents

Reading from paper still is, and perhaps will be forever, more pleasant than reading from screen. However, when we print a part of a portable document, with a layout adapted to the characteristics of screens, the results look rather silly. For instance, only half of the paper is used and minimal white space is on top and left or right.

With  $\text{\TeX}$  it's not too difficult to generate different layouts from the same source. This enables us to provide documents in different versions tuned for screen or paper. We can even offer more paper variants, like single sided and double sided, color and grayscale, letter and A4.

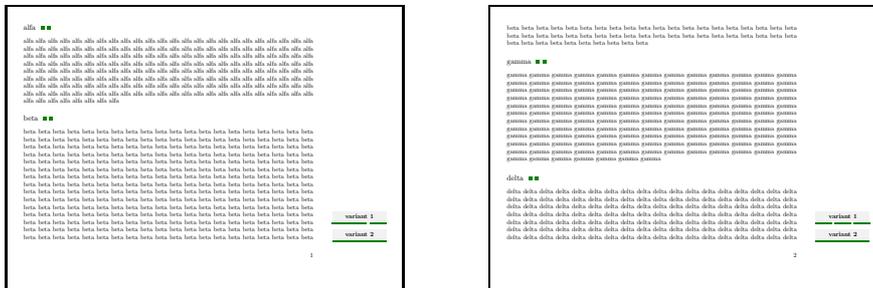


Figure 2: Document synchronization

To facilitate multiple versions we have to enhance our portable document with 'parallel document jumps'. Maybe the most simple and elegant alternative is providing small buttons after the titles of chapters and sections, one button for each parallel text. Although simple, it does not always look nice. A second solution lies in a visual analogy of  $\text{\TeX}$ 's  $\backslash\text{mark}$ . We can offer the reader up to three possible jumps on each page: to the end of the previous page (for experts:  $\backslash\text{topmark}$ ), to the first on the current page ( $\backslash\text{firstmark}$ ) and to the last on the current page ( $\backslash\text{bot}$ ). The corresponding buttons can be placed anywhere on the page. In Figure 2 we see both alternatives. The titles are followed by two buttons, that let us jump to two alternative versions. The buttons in the lower right corner show the jumps per page: on page 1 there are two jumps: **alfa** (top and first) and **beta** (bot), on page 2 we have three jumps: **beta** (top), **gamma** (first) and **delta** (bot).

At the moment both mechanisms are implemented. They work well but a connection with the printing mechanism would be nice. Technically spoken, we can launch programs with PDF, e.g. a printer driver, that prints the corresponding pages.

## 5 Typographic interface

The user-interface of a book is contained in the book itself. The only tools we use when reading are our hands. The look and feel of a book is determined by the designer. One of the characteristics of programs and user-interfaces is that they change. The beauty of books is that they don't change. Apart from their content, they give us insight in the era in which they were written and produced. Also, in high quality books, the layout is adapted to the content and the intended use. Although a lot of the underlying principles are fixed and based on years of typographic experience, books look different. So why should we give every portable document the same interface? And what exactly is a typographic interface?

Because the medium (with such quality) is quite new, there is no unique answer to this question yet. We can think of hot spots, menus or active words. But an interface does not have to be explicit. One may expect that clicking in a table of contents of index may result in some kind of jump. At the moment there are many non-interactive documents around. This means that we must provide readers with some cues, at least for the next few years. For the moment color suffice.

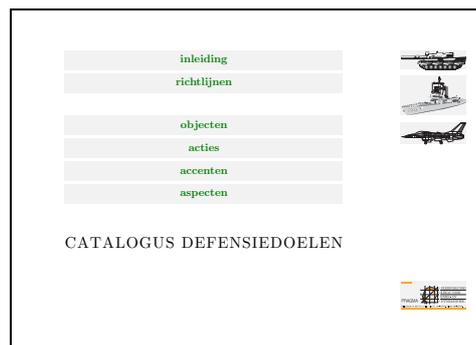


Figure 3: Typographic interface

The interface of a portable document must be determined by designers and not by programmers. The non intelligent part of the user interface, like *goto first*, *next*, *previous* or *last page*, can be programmed by means of references. Options like going to the previous jump or searching for words on the other hand, are to be provided by the program. At this moment these 'intelligent' options are not accessible as pdfmark.

We can think of some sort of dynamic typography, but still some constraints should be defined. One day, when computers will be real fast,  $\text{\TeX}$  can be used for real-time formatting.

With Acrobat, texts can be presented full screen, i.e. without windows, standard buttons and other items. This feature enables typographers to determine the interface. Some day there will be portable computers, tuned for reading text, with simple manual

controls, a high resolution display ( $\geq 300$  dpi), no battery problems and lots of memory. Personally I hope printed matter will be around for at least my lifetime. I just don't want to think of electronic bookstores yet. All those 'metaphores' or visual look-alikes will lose their meaning when one doesn't know the originals any more.

## 6 Selective printing

An integrated user-interface confronts us with a problem: when we print (part of) a document, we also print the user-interface. This means that viewing programs must be able to hide marked parts of the text from printing. Acrobat 2.0 supports embedded printer-only commands, so will probably support its counterpart, embedded viewer-only commands, in the near future .

## 7 Layers of control

We can think of many extensions to both Acrobat and our  $\TeX$  macros. One for instance is 'layers of control'. By this we mean that certain properties of the document, like typographic menus and statusbars, can be hidden or made visible by users. Typographic elements that enable us to navigate through the text, don't always add to the beauty of the text. It would be nice if users could turn them off when they are not necessary or when they irritate when reading.

## 8 Tables of contents

The concept of tables of contents slightly changes when developing portable documents. Acrobat provides bookmarks. These are entries to some sort of system table of contents. Because we don't have typographic control over such lists and because they claim too much space on the screen we don't use them. To be honest, we don't even need them because  $\TeX$  can generate them.

Of course, clicking on an entry in a table of contents, has to result in a jump to the corresponding chapter or section. Because active words are colored (as users expect), we only make the numbers of chapters and sections active. Too much color simply doesn't look so well. We don't click on pagenumbers, because in many cases we don't show them.

Tables of contents are active by default. The necessary links are generated without any interference of the author. That's the way it should be and that's the way it's done. But to make this possible, we had to enhance the table of contents macros, that are written to an auxiliary file, with two extra arguments: an internal referencing number and, to accommodate Acrobat 1.0, an extra real pagenumber. We can't use section

Kwaliteit		
1	Inleiding	3
2	Kwaliteit	4
2.1	Inleiding	4
2.2	Wat is kwaliteit?	4
2.3	De gebuikgerichte benadering van kwaliteit	4
2.4	Kwaliteit en de klant	5
2.5	Interne en externe klassen	6
2.6	Kwaliteit en de organisatie	6
2.7	Kwaliteitsbeleg	8
2.8	Kwaliteitsstelsels	10
2.9	Normalisatie	11
2.10	Normen	12
2.11	De ISO 9000 serie	13
2.12	Het documentatiesysteem	15
2.13	Kwaliteitskosten	16
2.14	De kwaliteitskostengrafiek van Juran	17
3	Meting van kwaliteit	20
3.1	Inleiding	20
3.2	Spreading en tolerantie	20
3.3	Het gemiddelde	21
3.4	De modus	23
3.5	De range	24
3.6	De standaardafwijking	24
3.7	Het in beeld brengen van gegevens	25

Index

Arbeidsomstandigheden

Milieu

Figure 4: Table of contents

numbers as reference, because they are not unique – how about five chapters 1 in five parts of a manual – and sometimes they are not even there.

We see that there can be more than one reference to a chapter or section: a hidden one, supplied by the system and used for tables of contents, and visible, user defined ones. Actually in Con $\text{T}\text{E}\text{X}$  we can give lists of references, just because chapters can handle more subjects and labels for references have to be meaningful:  $\backslash\text{chapter}[a,b,c]\{\text{Alphabet}\}$ .

## 9 Structuring elements

Not every document has chapters and sections. Quality System manuals for instance have their own ways of structuring, often with an alternative numbering of sections. These manuals contain a lot of references, like references to forms and specific procedures.

A consistent system of numbering is a necessity for robust (automatic) referencing and local tables of contents. This means that, when producing documents with alternative numbering, we have to supply  $\text{T}\text{E}\text{X}$  with structuring commands like  $\backslash\text{nextchapter}$ , which means as much as: 'here we go to another level 1 structuring element'. When using these commands, we can still produce a table of contents for this specific chapter. This problem is not unique to portable documents, but it showed up producing them.

## 10 Multiple indexes

Indexes, of which there may be many, have to be active too. Because pagenumbers are not unique as reference, macros had to be extended. Because an entry in an index can contain more references, resulting in more pagenumbers after a typeset entry, we make



Figure 5: Structuring elements

those references active. Of course we could make a jump list out of the pagenumbers (see 'reader profiles' below), but this has no real use.

Pagenumbers in portable documents seldom have a meaning, so why should we make them active and/or even show them? But isn't it a bit overdone to keep pagenumbers in a document for the sole purpose of a register? That's why we have some alternatives. Because the number itself doesn't say much we can use substitutes like ●, ■ or letters. It looks better and works fine.

## 11 Cross referencing

A whole book can be written on referencing and the mechanisms to support this. There are references to structuring elements, like chapters and sections, and to typographic ones, like figures and tables. Referencing can be done by page or by text: *see page 5* or *see also figure 3.1*. Portable documents offer some more, like references to locations: highlighted words in the text that one can click on. Clicking brings us to the world behind these words. These three types of references can be characterized by `\on`, `\in` and `\to`, shorthand for *on page*, *in something* and *to somewhere else*. There is another one: `\from`, that stands for *from document*.

Of course there are more types of references. In portable texts we can activate (jump to) programs. We can refer to lists of publications (`\by`), to items in lists, to margin words and to enumerations, like questions and answers (all `\on`'s and `\in`'s). Although it's best not to refer too much, electronic documents can't do without. It's just one way more to navigate them.

## 12 Local and global referencing

Take a table of contents or an index. In a book, we can hold our fingers at an index page and walk through the text with our other hand. In a portable document, there is no clear (tactile) concept of the beginning and the end of a document. On the other hand, we can offer as much tables of contents and indices as we want, because it doesn't spill paper. We can for instance start each chapter with a table of contents and offer the reader the possibility to jump to this table from each page in this chapter. For this purpose we can provide a button in one of the margins.

It is quite logical to label a table of contents with the reference [contents]. But with many tables of contents, let's say one for each chapter, we can't distinguish between them when we refer to one. This problem can be solved by introducing reference prefixes. By giving each reference an invisible prefix, like `chapter this:`, `section that:` or just a system supplied number, we have unique local tags and references. When a reference can't be solved locally, the underlying mechanism can always check if there is an not-prefixed global one. In our example this can be the table of contents of the whole document. It's even possible to walk through all prefixes until one is found, but because this is not very transparent, we have this option disabled.

Maybe the mechanism described isn't easy to understand without seeing it at work in practice. One has to take our word that from the users point of view, things are simple. The mechanism, that most of the time operates behind the screen, is also enhanced with cross-document referencing. We don't think we would have needed and developed such a complete and complex mechanism when only paper text was to be produced.

## 13 Multiple word references

Multiple word references normally only occur with `\to` or `\from`. On paper it's no problem if a reference crosses a line or page boundary. In an portable document we have to make each word in such a reference active. At the moment, we don't break up individual words. Because words belonging to the references are to be given explicitly and references are to be broken up in individual words, the mechanism of typesetting references is a bit more complicated than the one normally needed for paper texts.

One could state that Acrobat should handle this, but we don't agree. Acrobat has no knowledge of the meaning of text and the typographic used in it. Acrobat is perfectly able to recognize words and its search engine works fine – in version 2.0 it even accepts legislatures like `ff`, `fl` and `fi!` – but it would be a nearly impossible job to guess what a typographer means. Do we want to let the user click beyond the end of a line (specified with a `pdfmark`) or do we want to continue on the next line? Enhanced `pdfmarks` would be necessary to accomplish this.

## 14 Exact positioning

Version 1.0 of Acrobat only had a primitive referencing mechanism: one had to supply pagenumbers. Normally, when tagging a location to refer to, the pagenumber is remembered. A reference mechanism uses the page numbers as they appear in the text. When we number by chapter, pagenumbers look like *6.14* or *2–4–37*. Acrobat 1.0 expected real pagenumbers. Because of this incompatibility in pagenumbers, we had to adapt our macros. With version 2.0, referencing by label became available, so the reference mechanism could be simplified. In fact we didn't and now we have a dual mechanism, just in case we have to support more document formats.

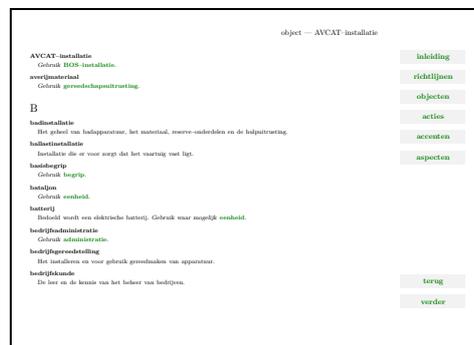


Figure 6: Exact positioning

Before Acrobat came available, we were already experimenting with `dviwindo` from Y&Y. This viewer offers a very simple, but powerful hyperlink mechanism, that only uses labels. It offers more: jumping to some location results in an exact positioning of the cursor. When this feature comes available in Acrobat too, PDF-documents will be more powerful. Think of dictionaries in which clicking on references like *see that other word too* brings us to right page and positions the cursor at *that other word*.

Acrobat offers the option to magnify the target location of a jump. It is also possible to show only part of the page. Because we layout our portable documents for screens, we don't use this feature. We think it is better to show the reader the page as it is, but future applications can make us change our mind. For instance, I think it is possible to define cyclic references, e.g. clicking on a figure means going to the figure itself. Because it is possible to magnify the target location, we could use such a cyclic reference to magnify the figure to full-screen size (`click: magnify`, and `click again: previous view`). To accomplish this in Acrobat, one needs to specify the part of the page that is to be magnified. Because  $\text{\TeX}$  has no absolute coordinates available – as a result of the dynamic character of composing pages – this feature can't be used. A solution to this would be an extension of PDF in which relative coordinates can be used.

## 15 Multiple documents

Referring to other documents was not hard to implement, but again, it required an adaption of our reference mechanism. We already mentioned prefixes. A reference to another document is done by an extra prefix. In `[texbook::somewhere]` we have the document prefix `texbook::`. When using multiple documents it is not necessary to have the other documents references available when we are only using `\to`. When we want to refer to text (`\in`, like *figure 3.2*) or page (`\on`, like *see page 12 of the T<sub>E</sub>Xbook*), we do have to load the references of the other document. Each reference generates one macro, in which the several components of the reference are packed (pagenumber, text and real pagenumber). As long as we do have enough memory available, this is no problem. Because our macro-package ConT<sub>E</sub>X is over three times the size L<sub>A</sub>T<sub>E</sub>X and has a memory extensive user-interface, the memory left for names of macros is already low. Of course all references can be packed in one macro, but only at the cost of processing time. (Packing works fine for two-pass optimization. We use lists of references for marginword placement, float reordering, optimal list breakup, version control and more.)

## 16 Common data

Using more documents together means that they must be as consistent with each other as possible, specially when they change a lot, like some kind of manual. This means that information sensitive to changes has to be as abstract as possible and has to be defined only once. This is one of the powers of T<sub>E</sub>X, but at the cost of memory. As we've seen before, time will solve this.

## 17 Parallel constructs

One of the first things we implemented in ConT<sub>E</sub>X was a block-move mechanism. We use this mechanism to relocate blocks of text, independent of their location in the ASCII-text. We put the answers after the questions, hide the answers and recall questions and answers in an appendix. It is also possible to label blocks and call them up by name. The mechanism, that is completely handled by T<sub>E</sub>X itself, keeps track of the original structure of the text. This is necessary because the numbers of questions and answers can be prefixed by the numbers of chapters. It's also necessary to keep track of the local character of blocks: do we call them up at the end of each chapter and/or at the end of the document. In our example questions and answers can be seen as parallel typographical constructs.

This already (in terms of macros) complicated mechanism seemed complete. But portable documents make it possible to link corresponding blocks together: click on a question and go to the answer and vice versa. Here we see a kind of reference that is unique to portable documents.

## 18 Alternative pagenumbers

One of the disadvantages of portable documents is its higher degree of abstractness. Numbers of pages, megabytes and scrollbars don't mean as much as a handful of paper or a bookshelf of volumes. Numbers of pages have lost their meaning, because we don't go to pages but to locations.

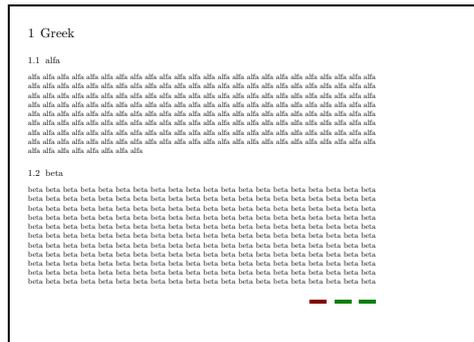


Figure 7: Subpagenumbering

Because we don't want our readers to get lost, we have to offer them some kind of status-information. One kind of pagenundering that makes sense is subpagenumbering. We can for instance give the reader some indication about the length of the current section he or she is reading. We can offer small bars, one for each subpage (screen) and highlight the current one. These bars are active: click on bar two means going to subpage two. Of course we can give cues like *page n of m*, but in that case we miss the interactivity.

## 19 Status bars

We can mimic the more traditional user-interfaces. One of the advantages of this is that users know what to expect. Although a text is no tape- or videoplayer, everyone seems to know what to do when < and > appear. Acrobat uses << and >> to navigate through the list of jumps. This analogies is a bit strange, because video recorders use this symbol for fast for- and backward and CD-players for going to a previous or next track. We now have at least three incompatible interfaces.

When we take volume 8 of 20 volumes of an encyclopedia from the shelf, we have some impression of what we are reading. Size and structure are obvious. Portable documents are much more abstract. One has to visualize its size and make some mental map of it. On computers, scroll bars can give some insight in the size of the information at hand. At the cost of a lot of overhead in terms of `pdfmarks`, it is possible to provide

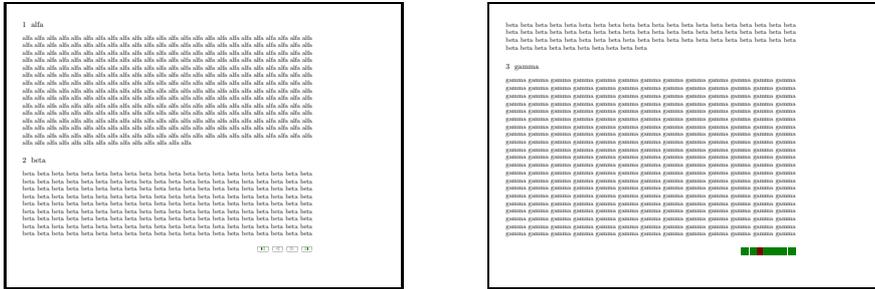


Figure 8: Statusbars

scrollbars in PDF. Because  $\text{\TeX}$  is such a strong typographic language, only our fantasy is the limit.

## 20 Active figures

If we can make text active, why not make figures active too? In instructional texts it would be handy when pointing to some part of an object (like a drawing of a machine, a photograph of some tool or a flowchart) would bring us to an explanation of this part (and vice versa).

It is common practice to let  $\text{\TeX}$  place figures.  $\text{\TeX}$  handles figures as an abstraction: a figure has dimensions and a name. Normally, figures are made with specialized programs and not by  $\text{\TeX}$ . Usually  $\backslash\text{special}$  is used to communicate the characteristics of figures (name, scale and/or dimensions) to the DVI-processor.

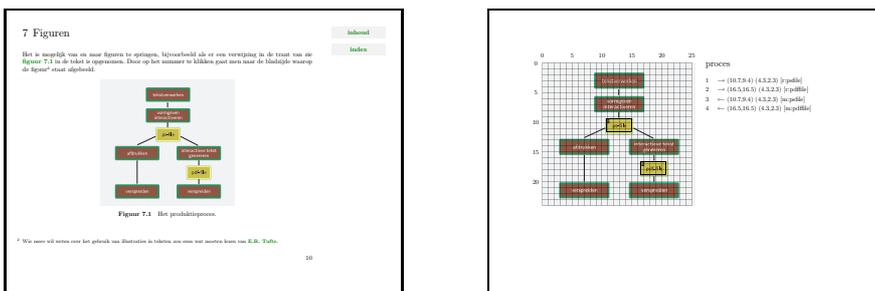


Figure 9: Active figures

It's not very handy to fall back on drawing-packages when we want to make figures interactive. First, drawing packages don't support this. Second, if they do, scaling is

not possible outside such a package unless buttons defined by `pdfmark` do scale too. Fortunately,  $\TeX$  can do what we want.

One way to make figures interactive is to overlay them with an invisible grid with a fixed number of (let's say 25) ticks. This grid can be scaled along with the figure. We now can define active areas (tagged by a label) in terms of coordinates and dimensions. The same can be done with areas that are referred to. These reference areas (of type `\to`) can be used in the common way.

## 21 Reader profiles

Tables of contents, menus and indices are examples of structuring elements of texts. In huge, complex and continuously changing documents, there is another one: the reader profile. In terms of text, such a profile tells specific readers what parts of the document are worth reading. In portable documents, viewing programs can take us by the hand and show us what's to be read.

To me, Acrobats 'article thread' mechanism seems originally meant for connecting columns in periodicals. When we look at examples of interactive periodicals, we often see a one to one similarity of the paper and portable version. This means that articles start on one page and continue on a following one. Because of their cramped pages, the article mechanism is useful: one starts at the column, that is or can be magnified for better reading, and jumps from column to column and page to page. This is just one more example of the page-oriented character of PDF.

We can abuse this mechanism for reader profiles. When we produce the text, we can tag parts of it by logical names. A collection of such tags, connected to a starting point, forms a reader profile. Clicking on the starting point starts the article reading mechanism and the reader just has to click to follow the path. At the moment the mechanism works but misses some userfriendliness, this as a result of Acrobats pagewise behavior. We are in need for some mechanism that spans pages and provides some background color of gray when viewing the threaded text.

## 22 Version control

A variant of reader profiles is version control. We can mark the adapted parts of a document and offer readers the option of reading only these. This mechanism works in concert with a mechanism for selective printing of updated pages. Version control works by numbers and not by tags. This makes it possible to offer more than one reading path: version 1.2 and higher, 1.3 and higher, 2.0 and higher and so on.

A combination of version control and reader profiles are supported too. A reader can read only the updated parts of his or her profile. This is useful in for instance updated manuals.

## **23 Color**

Color can also be used to make texts more attractive. Of course color can be used in figures. Used with care and with consistency in mind, color can enhance texts. It's common use to mark active words in a text by color. When using colored figures, we can use a colorbar to tell readers which parts of a figure are active. This means that we must use our color with care, especially in figures. Although not unique to portable documents, the color mechanism used in  $\text{\TeX}$  must support consistent use of colors. One has to keep in mind that not Acrobat but  $\text{\TeX}$  is responsible for the quality of typesetting. This nearly always means that the users of  $\text{\TeX}$  are to blame for poor esthetics and not  $\text{\TeX}$  and Acrobat.

## **24 Conclusion**

As we mentioned before, PDF and its accompanying Acrobat programs, have to prove their stability yet. Users of  $\text{\TeX}$  know that quality and stability are no guarantee for a wide acceptance. On the other hand the nearly perfect match of  $\text{\TeX}$  and Acrobat will undoubtedly lead to acceptance in the  $\text{\TeX}$ -community.

We already mentioned that the PDF format is still under development. Some obvious options are still missing and the present options are not yet explored to their limits. One of the mayor problems we will face in the (near) future is incompatibility. We can use new options, but can not be sure if readers have the most recent version of the viewers. (Paper documents don't have this problem, except when we change the language.) Maybe this is no real problem because the basic viewer is in the public domain and can be distributed with the documents.

A maybe even bigger problem is that we don't know what new viewing devices will come available. With tools like  $\text{\TeX}$  at least we can explore the bounds of the devices already available.



# PPCH $\TeX$ : typesetting chemical formulas in $\TeX$

J. Hagen and A.F. Otten

Pragma  
P.O.Box 125  
8000 AC Zwolle  
The Netherlands

## Abstract

This article is about a package for typesetting chemical formulas. The primary interface of this package is in the dutch language. Because PPCH $\TeX$  has a multilingual interface, all commands and keywords can be toggled to english. The Dutch version of this article is published in *N $\mathcal{T}$ G's MAPS* (94.2) and is translated to English by H. de Weert.

## 1 Introduction

The macro-package PPCH $\TeX$  can be used to typeset chemical formulas. The macros are based on P $\mathcal{C}$  $\TeX$ , a macropackage that was created to facilitate the drawing of graphics and other line diagrams. P $\mathcal{C}$  $\TeX$  was brought to the public domain by M.J. Wichura. We consider a second implementation, using PSTRICKS of T. van Zandt.

The macros can be used within different  $\TeX$ -environments and only depend on Knuth's plain  $\TeX$ . In addition, some general macros of our Con $\mathcal{T}$  $\mathcal{E}$ Xt-library are utilized. Besides, macros are typeset in such a way that further development is quite easy.

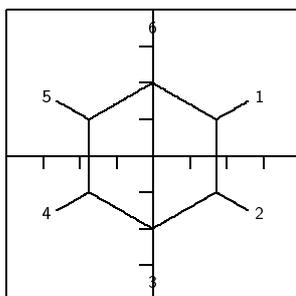
At first, macros are created to typeset chemical structure formulas. Moreover, reaction mechanisms can be reflected. Chemical structures can be typeset in different sizes and comparable formulas can be linked optically. Structures of frequent occurrence can be predefined and recalled.

During the development of the macros, processing speed is subordinated to flexibility, simplicity and quality. No use has been made of the mechanism (available in P $\mathcal{C}$  $\TeX$ ) to store parts of figures in a file. It turned out that this mechanism does not produce a gain of time.

*The macros are still being developed. For example, the mechanism to place texts still has to be refined and some structures like CHAIR still have to be added.*

## 2 Structures

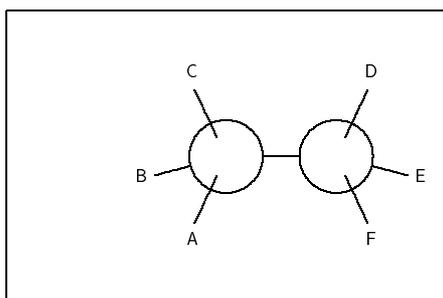
The number of commands that is used to typeset chemical structure formulas is reduced to four.<sup>1</sup> In the following example all of these commands are used.



**Example 1**

```
\setupchemical[axis=on,border=on]
\startchemical
\chemical[SIX,B,R,RZ][1,2,3,4,5,6]
\stopchemical
```

Different features of the typesetting can be set up with `\setupchemical`. If something is set up in this way, the setups are valid for all the following formulas.<sup>2</sup>



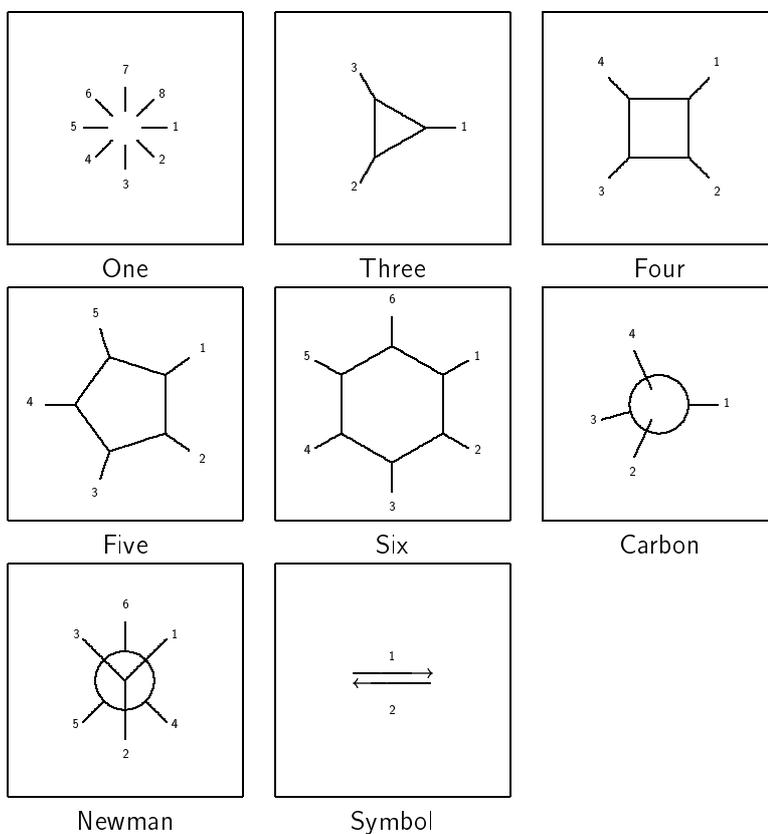
**Example 2**

```
\startchemical[border=on,width=6000]
\chemical[CARBON,CB1][A,B,C,D,E,F]
\stopchemical
```

As can be seen from both examples, `\chemical` is the central command. This command, that can be typed many times within a `\start-\stop`-pair, gets one or two arguments. These arguments are given between `[ ]`. The first argument refers to the bonds that are to be drawn. The second argument contains the atoms or molecules that are to be reflected. Text is typesetted in a mathematical mode, so everything that is normally allowed between `$ $` can be given.

We work out the first example. First of all the keyword `SIX` is given. By using this word we can indicate that we want to draw a sixring structure. In the same way we can use the keywords `ONE`, `THREE`, `FOUR`, `FIVE`, `NEWMAN` and `CARBON`.

1. The concept structure in this manual only refers to the chemical structure. It is not related to the structure of the text that is used to typeset the formula.  
2. Obviously the scope can be confined by using `{ }` and the grouping macros `..group`. The setups can also be given immediately after `\startchemical`. In that case the setups are applied to one formula.



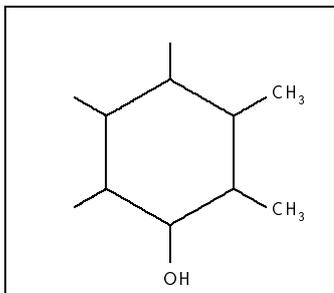
Within these structures chemical bonds between C-atoms can be indicated in a comparable way. For instance, in this example we use B and R. Bonds are numbered and can be indicated in different ways:

```
\chemical[SIX,B1,B2,B3,B4,B5,B6]
\chemical[SIX,B135]
\chemical[SIX,B1..5]
```

These commands create parts of a sixring structure. R enables us to add substituents to the sixring structure. The command R draws the beginning of a bond with a substituent from an angular point in the sixring structure ( $\angle 120^\circ$ ). The concerning angular point is indicated with a number.

```
\chemical[SIX,B1..6,R1..6]
```

The above mentioned command only places bonds to substituents. Substituents themselves are indicated with RZ. Therefore in this case, numbers are being used to mark the position. In the second optional argument substituents are given as text.



### Example 3

```
\startchemical[border=on,width=4500]
\chemical[SIX,B1..6,R1..6,RZ1..3][CH_3,CH_3,OH]
\stopchemical
```

If the second argument is omitted, no text is placed, so the command RZ1..3 has no effect.

## 3 Definitions

It is possible to build a library of structures. As we wish, we can recall these structures at a later point of time and provide them with extra components. Furthermore they can serve as building blocks for more complex structures. Structures can be predefined with the T<sub>E</sub>X-primitive `\def`.

If a structure, for example `[SIX,B,R,RZ]`, is often used, it is practical to predefine this structure.

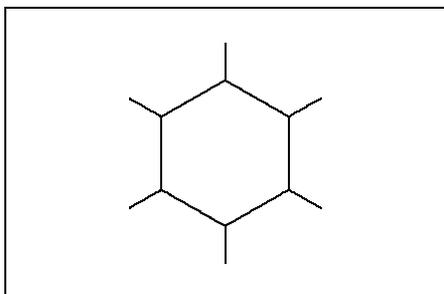
```
\def\sixring{\chemical[SIX,B,R,RZ]}
```

Instead of `\def` the following command can be used. In this case an already existing definition will be announced.

```
\definechemical[sixring]
{\chemical[SIX,B,R,RZ]}
```

Although both ways of defining are allowed, the second way is more robust. Protective measures are taken to avoid conflicts with existing commands.

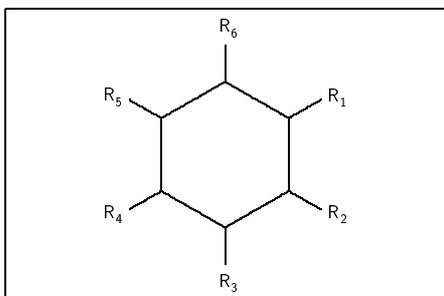
The commands `\chemical[sixring]` provides a sixring structure without substituents. No second argument is given.

**Example 4**

```
\definechemical[sixring]
  {\chemical[SIX,B,R,RZ]}

\startchemical[border=on,width=6000]
  \chemical[sixring]
\stopchemical
```

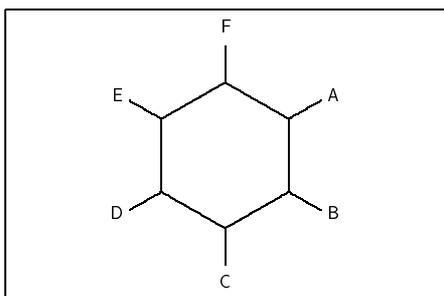
If we want to add six substituents, we have to carry out the following actions:

**Example 5**

```
\definechemical[sixring]
  {\chemical[SIX,B,R,RZ]}

\startchemical[border=on,width=6000]
  \chemical[sixring]%
    [R_1,R_2,R_3,R_4,R_5,R_6]
\stopchemical
```

The sixring structure can also be defined without substituents (RZ). In this case no substituents are expected if the command `\chemical` is given. Even now substituents can be placed, as is shown by the following example.

**Example 6**

```
\definechemical[sixring]
  {\chemical[SIX,B,R]}

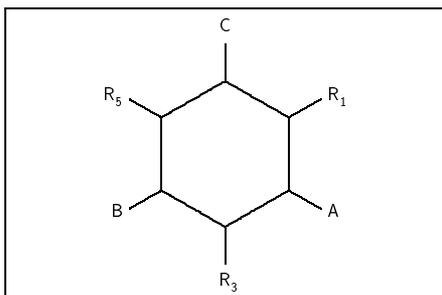
\startchemical[border=on,width=6000]
  \chemical[sixring,RZ][A,B,C,D,E,F]
\stopchemical
```

Essentially the number of possibilities is unlimited. One should be aware of the fact that the atoms and molecules of the second argument are raised in the sequence of the first argument.

In a definition atoms and molecules (texts) can also be placed.

```
\definechemical[sixring]
  {\chemical[SIX,B,R,RZ135][R_1,R_3,R_5]}
```

So in this definition always three substituents are added. If we decide to add more substituents, we have to explicitly state that we are dealing with a sixring structure (SIX).



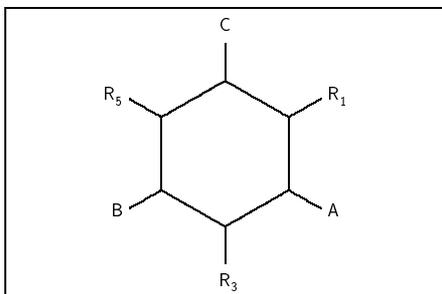
### Example 7

```
\definechemical[sixring]
{\chemical[SIX,B,R,RZ135][R_1,R_3,R_5]}

\startchemical[border=on,width=6000]
  \chemical[sixring,SIX,RZ246][A,B,C]
\stopchemical
```

So in definitions, the command `\chemical` has a global character and the command `\chemical[] []` has a local character. The idea behind this is that in the first case a series of commands is inserted and in the second case a complete independent structure is inserted.

In a definition the command `\chemical` can occur more than once. The last example can also be recalled with:



### Example 8

```
\definechemical[sixring]
  {\chemical[SIX,B,R,RZ135][R_1,R_3,R_5]
  \chemical[SIX,RZ246]}

\startchemical[border=on,width=6000]
  \chemical[sixring][A,B,C]
\stopchemical
```

If PPCH<sub>T</sub>E<sub>X</sub> makes mention of an unknown command, one has probably forgotten to type a structure command, like SIX or FIVE.

## 4 Bonds

In this chapter we show the bonds that can be found in the different chemical structures. The meaning of the commands will be explained by the reviews that are stated further in this article.

In the left column the complete bonds are shown, in the right column only the shortened bonds. Due to these shortened bonds, atoms and molecules can be attached to a bond. Bonds can be shortened on both sides, left (-) as well as right (+).

B	Bond	SB	Single Bond
		-SB	Left Single Bond
		+SB	Right Single Bond

Table 1: Saturated bonds

A bond can be followed by one or more numbers or a range, for instance: B1, B135 and B1..5. If all bonds are necessary, only B can be given.

In a ring structure an extra bond can be given and furthermore double or triple bonds can be introduced between atoms and molecules.

EB	Extra Bond	DB	Double Bond
		TB	Triple Bond

Table 2: Unsaturated bonds

A bond, in a sixring structure for example, can be shortcut. In this case the atom is given that has to be omitted, therefore a circle can be drawn in a sixring structure.

S	Shortcut	C	Circle
---	----------	---	--------

Table 3: Special bonds

Substituents can be attached to the angular points. Depending on the presence of atoms and molecules, bonds can be short or long.

R	Radical	SR	Single Radical
-R	Left Radical	-SR	Single Left Radical
+R	Right Radical	+SR	Single Right Radical

Table 4: Bonds to substituents

It is possible to bind substituents to the structure by double bonds.

Text can be linked to bonds. These texts are collected from the second set behind `\chemical` in the sequence that is given.

The atoms—/—molecules are numbered clockwise. In this case, combinations are allowed. With Z0 (z zero) a text can be placed in the middle of a structure.

ER	Extra Radical	DR	Double Radical
----	---------------	----	----------------

Table 5: Double bonds to substituents

Z	Atom	RZ	Radical Atom
		-RZ	Left Radical Atom
		+RZ	Right Radical Atom

Table 6: Atoms and molecules (radicals)

While positioning the atoms and molecules in the text, their (possible) dimensions are taken into account. In this case the width of C and the height of  $C_m^n$  play a prominent role. However, this mechanism can still be refined.

## 5 Combinations

Structures can be combined to complex compounds. Structures can be moved to other structures by using MOV, ROT, ADJ and SUB.

MOV	Move	move the same kind of structure in the direction of a bond
ADJ	Adjace	move a different kind of structure in the direction of the x- or y-axis, linked to a bond
SUB	Substitute	move a structure relative to another one in the direction of the x- or y-axis
ROT	Rotate	rotate a structure

Table 7: Displacements and rotations.

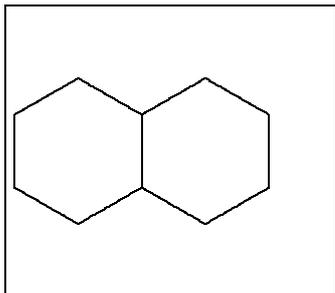
These four commands have another effect within the different structures. For example, the angle used to rotate at `\chemical[FIVE,ROT1,B]` differs from the angle that is used at `\chemical[SIX,ROT1,B]`.

In addition, within CARBON it is possible to mirror a structure. This can be done with MIR.

MIR	Mirror	mirror a structure
-----	--------	--------------------

Table 8: Mirroring

The direction of a displacement or the amount of the rotation is indicated by a number. Since these commands are closely related to the actual structure, they must be given before bonds and texts are drawn. It makes a difference whether `\chemical[FIVE,B,ROT1,R]` is given or `\chemical[FIVE,ROT1,B,R]`. The first call delivers an unwanted result.

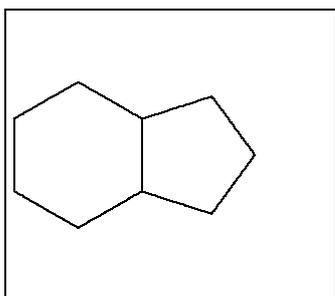


### Example 9

```
\startchemical[border=on,width=4500,right=3500]
\chemical[SIX,B,MOV1,B]
\stopchemical
```

Successively a sixring structure is drawn: `SIX,B`, a displacement is realized in the direction of bond 1: `MOV1`, and a second sixring structure is drawn: `B`. A displacement with `MOV` concerning a sixring structure can be realized in six directions, as opposed to a displacement with `ADJ`, which is realized in the four axis-directions ( $x$ ,  $-x$ ,  $y$ ,  $-y$ ). In a sixring structure some of these displacements coincide. The above example also could have been achieved with: `[SIX,B,ADJ1,B]`.

It is also possible to combine different structures. For instance, `SIX` can be linked to a structure `FIVE`. The mechanism that is responsible for this linking is for the greater part hidden from the user. In the following example a sixring structure is successively drawn: `SIX,B`, a displacement along the positive  $x$ -axis is achieved: `ADJ1`, and a rotated fiveering structure is drawn: `FIVE,ROT3,B`.

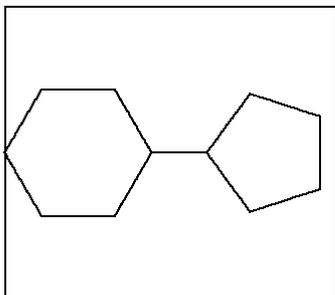


### Example 10

```
\startchemical[border=on,width=4500,right=3500]
\chemical[SIX,B,ADJ1,FIVE,ROT3,B]
\stopchemical
```

A transition to a connected structure can be achieved with `ADJ`. To get a good connection, one of the two structures have to be rotated with `ROT`. If a structure is not directly linked, but through a bond, one uses `SUB`. Rotations are made in steps

of 90 degrees, clockwise. Displacements with ADJ and SUB are achieved in the four axis-directions.



### Example 11

```
\startchemical[border=on,width=4500,right=3500]
\chemical[SIX,ROT2,B,R6,SUB1,FIVE,B,R4]
\stopchemical
```

We can therefore conclude that the sequence of the given commands is very important. An obvious sequence of commands is as follows:

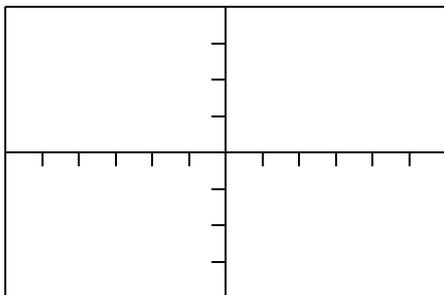
```
\chemical
[structure,                % SIX, FIVE, ...
  bonds inside the structure, % B, C, EB, ...
  bonds outside the structure, % R, DR, ...
  locations of atoms,        % Z
  locations of substituents] % RZ, -RZ, ...
[atoms,
  substituents]
```

As a rule, the connection of structures is reduced to some translations and rotations. Although it may not seem so, a certain systematic is enclosed. In fact, the process could be simplified. The automation that was already achieved in former versions, has been undone: it turned out that 'hidden' rotations induce misunderstandings with regard to the place of bonds. Furthermore, it is easier to provide a structure that is not rotated with bonds, atoms and molecules than to provide a rotated structure. It is better to define the parts of a complex structure first, possibly with translations, and to rotate the complete complex structure later.

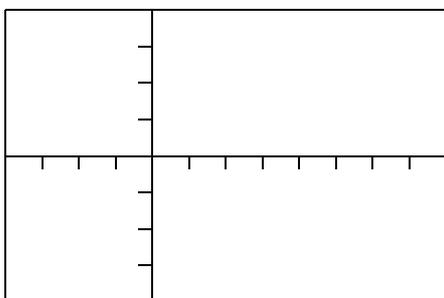
## 6 Axis

Structures are typeset in a bounded space, for convenience indicated by axis. The dimensions of the axis and the location of the origin can be defined in the setup. In addition

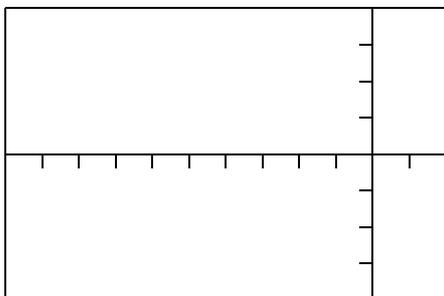
the axis can be made visible (for the sake of the location in the text) and a border can be drawn.

**Example 12**

```
\startchemical
  [axis=on,
   width=6000,height=4000]
  ...
  ...
\stopchemical
```

**Example 13**

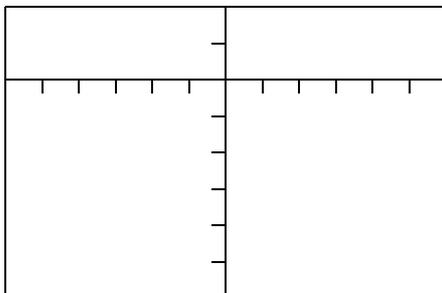
```
\startchemical
  [axis=on,
   left=2000,right=4000]
  ...
  ...
\stopchemical
```

**Example 14**

```
\startchemical
  [axis=on,
   width=6000,right=1000]
  ...
  ...
\stopchemical
```

The dimensions of the structure are determined by the dimensions of the axis. However, if `width=fit+` and/or `height=fit` is given, the dimensions of the total structure are determined by the real dimensions. Whatever is chosen is depending on the way structures are placed in the text. Side by side, on top of each other, etc. Example 12 shows the standard setups.

Within a `\start-\stop`-pair P<sub>T</sub>E<sub>X</sub>-macros can be used. Of course, some caution must be taken into account.



### Example 15

```
\startchemical
  [axis=on,
   width=6000,top=1000,bottom=3000]
  ...
  ...
\stopchemical
```

## 7 Setups

The behavior of the macro's and the layout of the formulas can be adapted to personal needs. After both `\startchemical` and `\setupchemical` setups can be given.

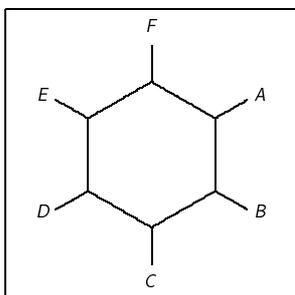
variable	values	default
width	number	4000
height	number	4000
left	number	
right	number	
top	number	
bottom	number	
resolution	number	see manual
corps	8pt 9pt 10pt etc.	see manual
style	<code>\rm \bf</code> etc.	see manual
scale	number small medium big	medium
size	small medium big	medium
status	start stop	start
option	test	
axis	on off	off
border	on off	off
alternative	1 2	1
offset	HIGH LOW	LOW

Table 9: Setups for structures

The axis reaches from  $-2000$  till  $+2000$ , in height as well as in width. The point Z0 is situated at  $(0,0)$ . Other setups can be set up with `left`, `right`, `above` and/or `under` in combination with `width` and `height`.

The dimensions of the characters can be set up with `size`. In doing so, the  $\TeX$ -primitives `\textsize`, `\scriptsize` and `\scriptscriptsize` are used. The dimensions of the structure itself can be set up with `scale` ( $1..1000$ ). The scale is also determined by `corps`. The keywords `small`, `medium` and `big` are attuned.

In the mathematical mode commands like `\rm`, `\bf` and `\sl` can be utilized in  $\TeX$  as well as in Con $\TeX$ t. PPCH $\TeX$  by default uses `\rm`. An alternative command can be set up with `style`. For instance in example 16 the substituents are typeset *slanted*.



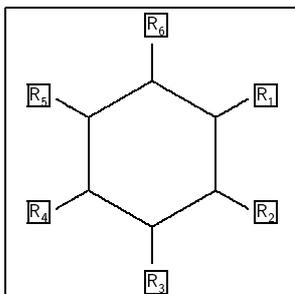
#### Example 16

```
\startchemical [border=on,style=\sl]
\chemical [SIX,B,R,RZ] [A,B,C,D,E,F]
\stopchemical
```

For the time being `style` is valid for chemical formulas in the text as well as in a figure. The sub- and superscripts also change, as can be seen from:  $\text{CH}_4$ ,  $\text{CH}_4$  and  $\text{CH}_4$ . Successively the setups are: `\rm`, `\bf` and `\sl`. When formulas are typeset in italic (`\it`), baseline-distances can be greater than normal. Within Con $\TeX$ t bold-slanted (`\bs`) and bold-italic (`\bi`) are also available. All these commands automatically adjust to the actual style (`\ss`, `\rm`, `\tt`):  $\text{CH}_4$ ,  $\text{CH}_4$  etc.

The time-consuming calculations can be short-circuited with `status`. The variables `border` and `axis` speak for themselves. A border round a text can be drawn with `option=test`. By doing so, one can see how things are aligned. The quality of the lines is set up with `alternative`. By default, P $\TeX$  uses a 5 point period to draw the lines. As one chooses alternative 2 smaller points are used. Therefore thinner lines are drawn.

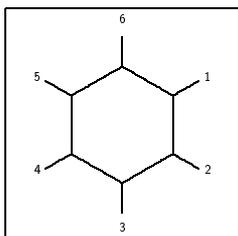
The offset refers to the position of the sub- and superscripts. By using `HIGH` the subscripts are positioned high ( $\text{H}_2\text{O}$ ). Self-evident, with `LOW` the subscripts are positioned a little lower ( $\text{H}_2\text{O}$ ).

**Example 17**

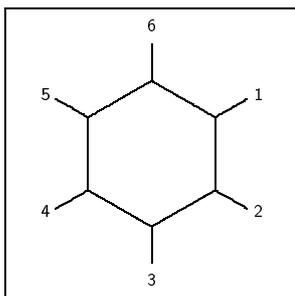
```
\startchemical[border=on,option=test,alternative=2]
\chemical[SIX,B,R,RZ][R_1,R_2,R_3,R_4,R_5,R_6]
\stopchemical
```

## 8 Dimensions

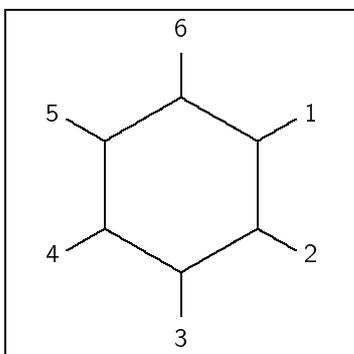
A structure can be shown in different sizes. Dimensions can be set up with `size` and `scale`.

**Example 18**

```
\startchemical[scale=small,size=small]
\chemical[SIX,B,R,RZ][1,2,3,4,5,6]
\stopchemical
```

**Example 19**

```
\startchemical[scale=medium,size=medium]
\chemical[SIX,B,R,RZ][1,2,3,4,5,6]
\stopchemical
```

**Example 20**

```
\startchemical[scale=big,size=big
\chemical[SIX,B,R,RZ][1,2,3,4,5,6]
\stopchemical
```

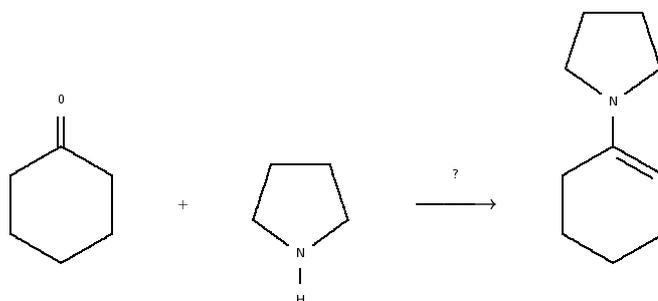
Scaling can take place between 1 and 1000. The values belonging to the keywords `small`, `medium` and `big` are attuned.

**9 Symbols**

To typeset reaction equations some symbols are available. In the following figure an equation is shown. This equation is defined as follows:

```
\setupchemical
[width=fit,
height=5500,
under=1500]
\hbox
{\startchemical
\chemical[SIX,B,ER6,RZ6][0]
\stopchemical
\startchemical
\chemical[SPACE,PLUS,SPACE]
\stopchemical
\startchemical
\chemical[FIVE,ROT4,B125,+SB3,-SB4,Z4,SR4,RZ4][N,H]
\stopchemical
\startchemical
\chemical[SPACE,GIVES,SPACE][?]
\stopchemical
\startchemical
\chemical[SIX,B,EB6,R6,SUB4,FIVE,ROT4,B125,+SB3,-SB4,Z4][N]
\stopchemical}
```

The command `\hbox` is necessary to position the structures behind each other. The symbols `GIVES` and `PLUS` speak for themselves. Extra space can be achieved with `SPACE`.



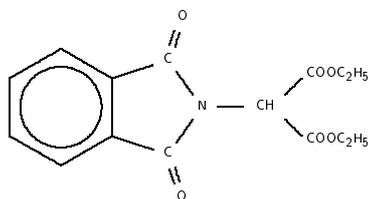
An equilibrium can be shown with `EQUILIBRIUM`. A text can be set above `GIVES` and `EQUILIBRIUM`. In the example this text is a question mark.

## 10 Special features

By using `ONE`, `Z0` can consist of more than one atom. In this case, the reserved space is insufficient. If more space is needed for the command `Z0`, the bonds 1, 2 and 8 can be moved with the command `OFF`, which means 'offset'. Below, an example of the use of this command is given.

```
\startchemical[width=fit]
\chemical
[SIX,B,C,ADJ1,
 FIVE,ROT3,SB34,+SB2,-SB5,Z345,DR35,SR4,CRZ35,SUB1,
 ONE,OFF1,SB258,Z0,Z28]
[C,N,C,0,0,
 CH,COOC_2H_5,COOC_2H_5]
\stopchemical
```

In this case the offset is 1, which means that we use one extra character. Similar, at first sight quite complicated, definitions can best be constructed by defining the separate parts first. The rotation can best be made last.



A new command is seen in the above example: CRZ. This command can be used to position an atom or a molecule in line with a bond, which is desirable in this case. This positioning could have been achieved with the command RZ, since one can influence the spacing with the second set:  $\{\backslash, 0\}$  instead of  $\{0\}$ .

OFF	Offset	CRZ	Centered Radical Atom
-----	--------	-----	-----------------------

Table 10: Special commands

## 11 Textformulas

Along with the typesetting of structure formulas, the typesetting of reaction equations is supported too. Therefore, the former described command `\chemical` has another two versions:

```
\chemical{formula}
\chemical{formula}{text}
```

This command will fit to the position in the text. This means that a difference is made between:

- a text-mode
- a mathematical text-mode
- a mathematical display-mode

If the command is given in the current text, automatically  $\$ \$$  are set round the command. In this way the command `\chemical{NH_4^+}` delivers the formula  $\text{NH}_4^+$ .

The same result is achieved by placing the command between  $\$ \$$ . Therefore in both cases the second argument is omitted. If we type the command between  $\$ \$ \$ \$$ , the second argument is obliged. The second argument may be empty:

```
$$
\chemical{2H_2}{} \chemical{PLUS}{} \chemical{O_2}{}
\chemical{GIVES}{} \chemical{2H_2O}{}
$$
```

gives:



This formula can also be defined in a shorter way:

```
$$\chemical{2H_2,PLUS,0_2,GIVES,2H_2O}{}$$
```

or even:

```
$$\chemical{2H_2,+,0_2,->,2H_2O}{}$$
```

The more experienced T<sub>E</sub>X-user will notice that the plus-sign as well as the arrow are located at the base-line. If you just compare + and +. Independent of the size of reflection, the + and the → are aligned.

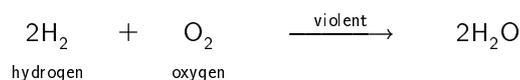
In addition to PLUS and GIVES, EQUILIBRIUM can be given, which delivers double arrows (<->).

This formula can also be typeset in the current text, where a smaller layout is chosen: 2H<sub>2</sub> + O<sub>2</sub> → 2H<sub>2</sub>O. It is also possible to show bonds. For example, the rather long sequence \chemical{H,SINGLE,CH,DOUBLE,HC,SINGLE,H} gives H—CH=HC—H. This can also be achieved in a shorter way: \chemical{H,-,CH,--,HC,-,H}. A triple bond can be called with TRIPLE or --: HC≡CH.

Now we just return to the display-mode. The second argument can be used to give an explanation of the formula:

```
$$
\chemical{2H_2}{hydrogen} \chemical{PLUS}{} \chemical{0_2}{oxygen}
\chemical{GIVES}{violent} \chemical{2H_2O}{}
$$
```

gives:



The size of the formulas that are placed in the text can be set up with the setup-command:

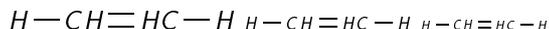
variable	values	default
textsize	small medium big	big

Table 11: Setups for textformulas

When `big`, `medium` or `small` is used,

```
\chemical{H,SINGLE,CH,DOUBLE,HC,SINGLE,H}
```

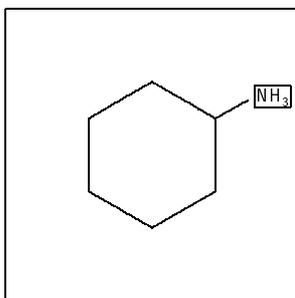
gives the successive formulas:



## 12 Subscripts

Sub- and superscripts are placed a little lower, as is recommended by Knuth in the  $\TeX$ book. Like this, the example on page 179 in the  $\TeX$ book, that is chemically a bit strange, can be typeset by typing `\chemical{Fe_2^{+2}Cr_2O_4}`. This command gives  $\text{Fe}_2^{+2}\text{Cr}_2\text{O}_4$ . Without a correction the command would have given the result:  $\text{Fe}_2^{+2}\text{Cr}_2\text{O}_4$ .

The location of the subscript is by default determined by the `offset`: `HIGH` or `LOW`. The `offset` can be overruled by the commands of the same name.



### Example 21

```
\startchemical[border=on,option=test,alternative=2]
\chemical[SIX,B,R1,HIGH,RZ1][NH_3]
\stopchemical
```

Although the location of the subscript can be set up by substituent, it is advisable to do this with `\setupchemical`. By doing so, all the formulas will be set up in the same way.

The keywords `LOW` and `HIGH` can also be used in text formulas, although this may lead to inconsistencies in layout. We can see that `\chemical{HIGH,H_2O}` results in  $\text{H}_2\text{O}$  and `\chemical{LOW,H_2O}` in  $\text{H}_2\text{O}$ .

## 13 Installation

The package PPCH $\TeX$  is developed as an extension to Con $\TeX$ t but can be used with other packages. The macros can be found in the file `m-chemie.tex`, with the `m` standing for module. Therefore the macros are not standard available.

Furthermore, the package can be used in combination with  $\text{\LaTeX}$ . In this case the file `m-chemie.sty` is used too.<sup>3</sup> PPCH $\TeX$  can be activated by means of `\documentstyle`:

```
\documentstyle[m-chemic]{}
```

3. The dutch word `chemie` stands for chemics in english.

In addition to the english interface that was described earlier, a dutch interface is available. The dutch version is loaded with:

```
\documentstyle[m-chemie]{}
```

## 14 Some dutch

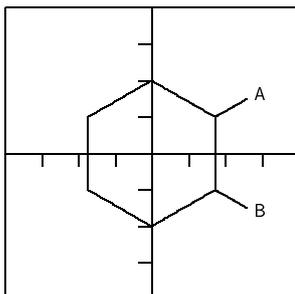
The original interface of PPCH<sub>T</sub>E<sub>X</sub> is dutch. One can switch back to dutch with:

```
\resetinterface
```

Switching back to english can be done with:

```
\setinterface[english]
```

Although more interfaces are possible, only the dutch and english are implemented. The file `m-chemie.sty` shows how an interface is defined.



### Example 22

```
\startchemie[assenstelsel=aan,kader=aan]
\chemie[SIX,B,R12,RZ12][A,B]
\stopchemie
```

## 15 Extensions

Of course, all T<sub>E</sub>X-users are allowed to use the PPCH<sub>T</sub>E<sub>X</sub>-macros in another, non-commercial, way. However, some caution has to be taken into account, since the macros are still being developed, optimized and made more robust. Some macros may seem quite complicated, but appearances are deceptive. For example, commands in the shape of `\setup...` utilize the macros that support nesting and different ASCII-layouts. Compare for instance:

```
\setupchemical[size=small]
```

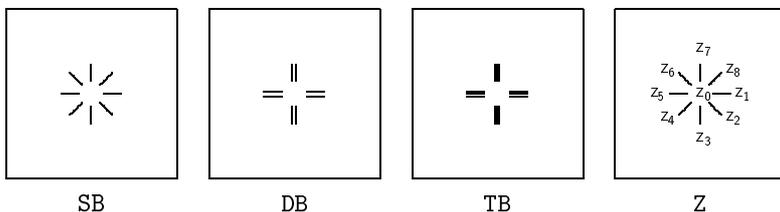
with:

```
\setupchemical
[size=small,
scale=500,
textsize=big]
```

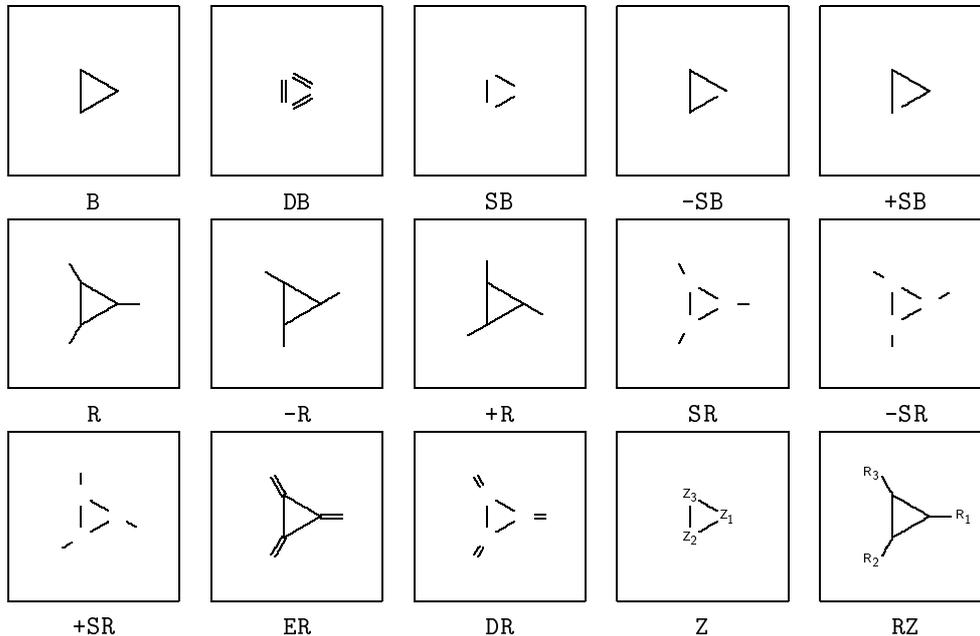
Setups can be given in an arbitrary sequence. When possible, spaces and newline-characters are suppressed and errors are announced.

The following overview shows the available structures. We use a somewhat smaller corps to save space. The manual of PPCHTEX is a bit more extensive in its overviews.

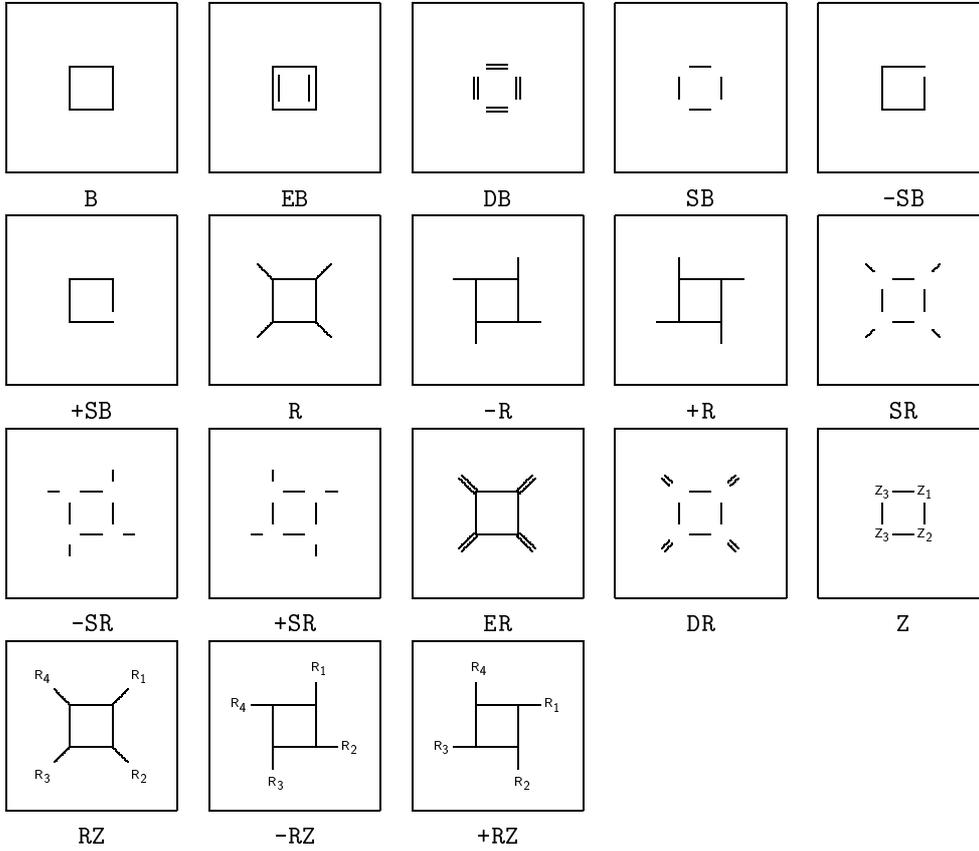
## 16 One



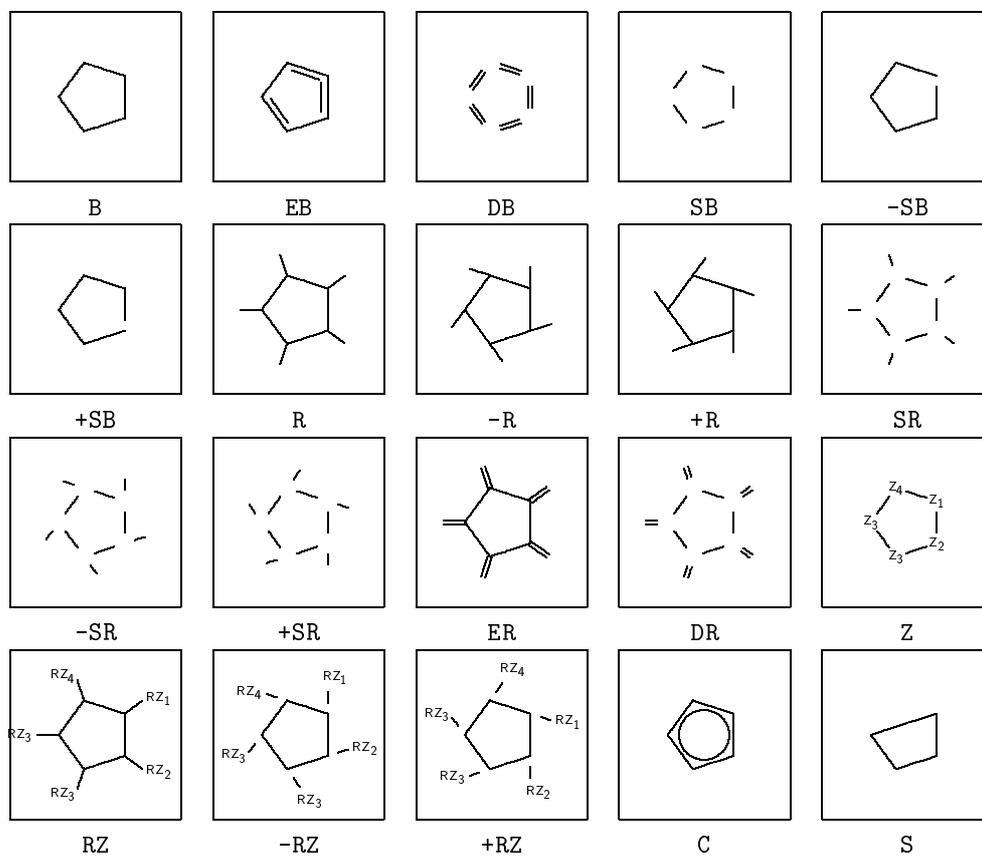
## 17 Three



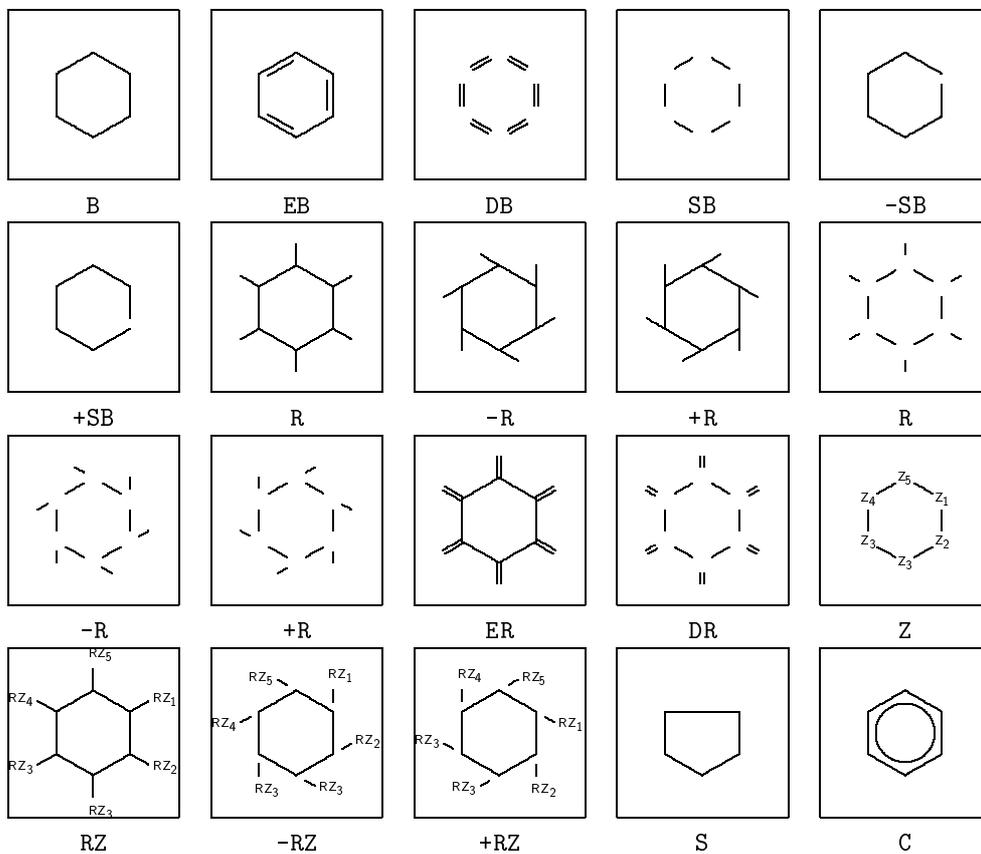
**18 Four**



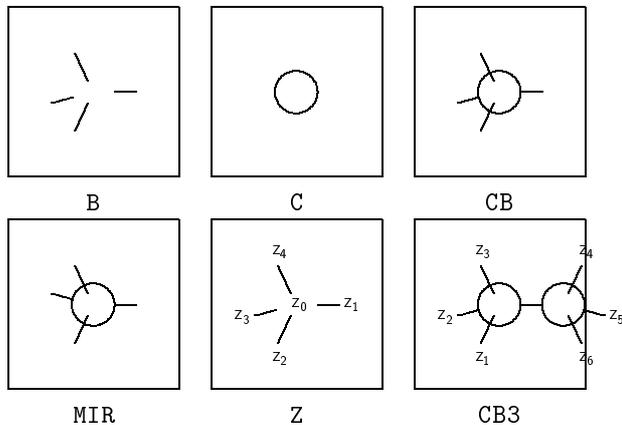
### 19 Five



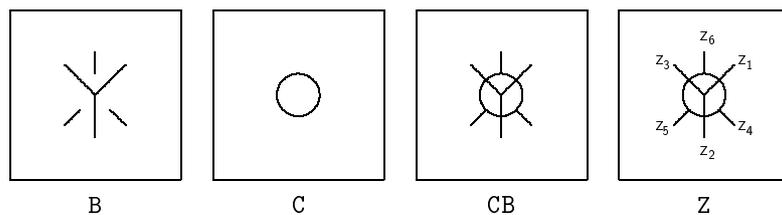
## 20 Six



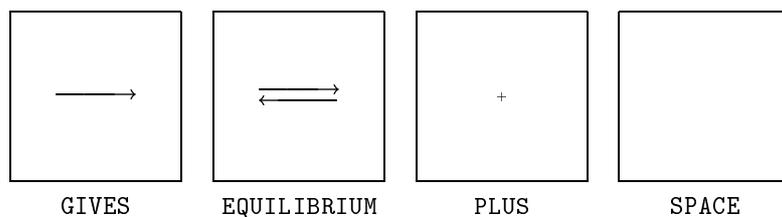
## 21 Carbon



## 22 Newman



## 23 Symbol



## 24 Other features

The following features of PPCH $\TeX$  are not described here, but one can find more on them in the manual.

- coloring parts of formulas
- active formulas in interactive texts

- automatic adapting to the corps size
- adapting to the resolution of printers
- the multi-lingual interface
- manipulating super- en subscripts

Also, the manual is a bit more extensive in its overview of the rotations and displacements.

# L<sup>A</sup>T<sub>E</sub>X, HTML and PDF, or the entry of T<sub>E</sub>X into the world of hypertext<sup>1</sup>

Yannis Haralambous and Sebastian Rahtz

haralambous@univ-lille1.fr, s.rahtz@elsevier.co.uk

## 1 The relationship between hypertext and L<sup>A</sup>T<sub>E</sub>X

Unlike *hypermarket*, *hypertension* and *hyperactivity*, where the prefix *hyper* expresses high quantity, and excess, *hypertext* is not a giant text, but a text with an internal structure that viewers can exploit to allow for arbitrary navigation through the document.

There is thus a relationship between the notion of hypertext and the markup system of L<sup>A</sup>T<sub>E</sub>X: both add structure to a document. For example, the L<sup>A</sup>T<sub>E</sub>X notion of cross referencing corresponds to the notion of linking in hypertext. The main difference between the two concepts is the lack of interest of T<sub>E</sub>X in screen interaction. T<sub>E</sub>X deals with boxes that can contain characters, rules, images, etc. The task of replacing these boxes with the actual characters falls to the screen or printer drivers. T<sub>E</sub>X being a tool for typographical composition, does not use a screen other than for previewing, and screen display is seldom considered the final aim of a T<sub>E</sub>X compilation.

This lack of interest in T<sub>E</sub>X of the screen is even more important when we recall that many PostScript constructions, introduced into a DVI file by `\special` commands, are typically ignored by previewers.<sup>2</sup>

We claim that, for effectively the first time in its existence, T<sub>E</sub>X is becoming seriously useful for creating documents whose aim is to be read on the screen. In fact, L<sup>A</sup>T<sub>E</sub>X is totally adequate for the automatic production of hypertext links, and the methods that will be presented in this article allow for an automatic conversion of almost every existing L<sup>A</sup>T<sub>E</sub>X document into an hypertext document. It is worth insisting that such a document

---

1. L<sup>A</sup>T<sub>E</sub>X This paper is based on one published by Yannis in *Cahiers GUTenberg* #19, January 1995. The translation from French was undertaken by Leonor Barroca and Sebastian Rahtz, who apologize to Yannis for the massacre of his elegant writing style. The article was revised and extended by Sebastian Rahtz.

2. Except for the lucky ones amongst us who can use operating systems with Display PostScript.

keeps all the typographical quality of L<sup>A</sup>T<sub>E</sub>X, and can be printed exactly in the same way as before.

## 2 Overview

T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X read a file that contains the text of a document with structural and visual labels, and create a second file which describes the printed page with great precision. This output file is called DVI (DeVice Independent) because it only contains abstract data: the position of each character on the page, the name of the font in which the program will find the pixels for this character, its code in this font, etc.

The visualization or printing of a DVI file presupposes the availability of a certain number of fonts. This is usually easy in the case of large systems or network-connected workstations, but it becomes problematic in the case of personal systems. The situation is even more critical when one wants to distribute electronic documents: a document that can be viewed and printed by a large number of people can hardly be distributed in DVI format (since this is only of interest to the T<sub>E</sub>X community, and one would be effectively limited to CM fonts, which are the only fonts (almost) guaranteed to be present on all T<sub>E</sub>X systems). In practice it is almost impossible, for anything but very simple documents, to keep on disk the document itself plus enough utilities to allow for its immediate previewing and printing, without having to install a complete T<sub>E</sub>X system.

Finally, hypertext links are not catered for in the syntax of the DVI format;<sup>3</sup> every attempt to develop an hypertext program to use the DVI format will lead to a new 'Hyper-DVI' format, with all the problems of compatibility with the T<sub>E</sub>X community which is (rightly) proud of the stability of its tools. It seems then that the DVI format is not the ideal candidate for a file format which is easily usable and sufficiently interactive to allow for the integration of hypertext links.

What then can we do? An obvious candidate for storing documents – at least, today – is the PDF format (*Portable Document Format*) developed by Adobe Systems. It is an extension of the PostScript language, closely resembling the syntax of files produced by Adobe's Illustrator program, with two important additions: support for device independent screen viewing and printing, regardless of the fonts used in the document, and the integration of hypertext functionality.

We shall return to the details of the PDF format in Section 7.1. For the moment, we describe how the PDF format can be integrated into the process of document production (electronic or printed). In Figure 1, input/output files are represented by oblique boxes, the software by rectangles, operations (visualization, printing, etc) by boxes with rounded corners, while arrows indicate the basic transformations described in this article.

The `.tex` file is our starting point (it is in the central circle). T<sub>E</sub>X will produce a `.dvi` file (it also uses macro files, and font metrics). If the L<sup>A</sup>T<sub>E</sub>X format is used, and

---

3. The description of the DVI format can be found on CTAN in the directory `dvivare/driv-standard/level-0/dvistd0.tex`.



to browse and navigate the document and print it on any printer supported by the host system.

It is easy to see that if the starting point is an HTML document, all the hypertext functionality will be kept, but we have also gained the typographic presentation of  $\text{\LaTeX}$ . A PDF document is a faithful copy of the printed document (it can even be photo-typeset to produce a professional quality result with color images, graphics, etc.). On top of that it offers hypertext navigation using links, which, with version 2 of Acrobat, refer not only to points inside the document, but also to other documents on the network.

The structure of a  $\text{\LaTeX}$  document can be exploited by a wide application domain: the most striking example is the vocal synthesizer ([3]) for the use of visually impaired people, which can pronounce a mathematical formula, and indicate the structure of the document by use of sound.

We will describe in this article another application: the creation of electronic books, whose presentation is no worse than the traditional books (since they can be printed with no loss of quality) but that offer some interactivity: hypertext navigation between table of contents, index, bibliography and text, on the same machine or across a network.

In the remainder of this article we will study each step of the process indicated by the fat arrows in the diagram of Figure 1.

### 3 HTML to $\text{\LaTeX}$

The HTML markup system is defined according to the SGML standard. It contains a limited number of tags, mainly for screen appearance; there are also various logical text styles (emphasis, address, quotation, lists, etc.), and visual styles like italic, bold, underline, etc, but there is no support for fundamental page objects like tables and footnotes. It is obvious that  $\text{\LaTeX}$  is a much richer language than HTML, and so the conversion from HTML to  $\text{\LaTeX}$  is essentially trivial.<sup>5</sup> The conversion has to do some simple jobs:

1. convert certain tags straight to a  $\text{\LaTeX}$  environments, such as `<CITE>` and `</CITE>` going to `\begin{quotation}` and `\end{quotation}`;
2. convert other tags to  $\text{\LaTeX}$  commands with arguments, such as `<em>` and `</em>` going to `\emph{...}`;
3. replace a very few tags with simple  $\text{\LaTeX}$  commands, like `\par` for `<P>`;
4. deal with accented character entities, so that `&eacute;` becomes `\'e` and `&ccedilla` becomes `\c{c}` and so on.

There are two classes of tags which present more problems:

1. Those which have no direct equivalent in  $\text{\LaTeX}$ , such as `<STRONG>`; the appropriate action for these is to convert them to new  $\text{\LaTeX}$  environments, and provide appropriate definitions in a style file. Thus

```
<strong>Very important!</strong>
```

---

5. Going in the other direction is much harder (see [2])

would be converted to

```
\begin{strong}
Very important!
\end{strong}
```

and an appropriate definition might be:

```
\newenvironment{strong}{\bfseries\itshape}{}

```

2. Tags for hypertext functions. For these we can conveniently use the *hyperref* package described below, to place the complete functionality of the hypertext commands into the dvi file. There are four situations we need to deal with:
  - a. Definition of a target (an “anchor” in HTML jargon) is achieved with `<A name="keyword"> ... </A>` (where `keyword`) is a unique (to the document) name chosen for the target; this is represented in L<sup>A</sup>T<sub>E</sub>X by `\hyperdef{}{keyword}{}{...}`, where `...` is the chosen text.
  - b. Definition of a link to an anchor in the same document, represented in HTML with `<A href="#keyword"> ... </A>` (where `keyword` is the name of the anchor to point to); in L<sup>A</sup>T<sub>E</sub>X we would write `\hyperref{}{keyword}{}{...}` where `...` would be the text which a user selects to make the hypertext jump.
  - c. Definition of a link to another document, which HTML marks as `<A HREF="address"> ... </A>` where `address` is a valid URL. The equivalent L<sup>A</sup>T<sub>E</sub>X markup would be `\hyperref{address}{}{}{...}`.
  - d. Linking to an image, which in HTML would be `<IMG SRC="address">`; in L<sup>A</sup>T<sub>E</sub>X, we convert this into `\hyperimage{address}`

## 4 L<sup>A</sup>T<sub>E</sub>X to DVI

Let us be clear from the start: any valid L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  document can produce a electronic equivalent, by the simple addition of

```
\usepackage{hyperref}
```

at the end of the document preamble. This loads Sebastian Rahtz' *hyperref* package, which redefines the following L<sup>A</sup>T<sub>E</sub>X macros to produce hypertext links:

- `\label`, `\ref` and `\pageref` (cross-referencing)
- `\chapter`, `\section`, `\subsection` etc (made into hypertext anchors)
- `\cite` (provides link to references; references can also be made to link back to their place of citation)
- `\index` (index creation)
- `\includegraphics` (inclusion of pictures)

Nothing more needs to be done to the document source, unless specific links are needed in a manner not supported by the generic L<sup>A</sup>T<sub>E</sub>X markup, in which case the “raw” commands `\hypertarget` and `\hyperlink` and `\hyperimage` can be used.

#### 4.1 The HyperT<sub>E</sub>X specification and the hyperref package

The *hyperref* package derives from and builds on the work of the HyperT<sub>E</sub>X project, described in the World Wide Web document <http://xxx.lanl.gov/hypertext/>. It aims to extend the functionality of all the L<sup>A</sup>T<sub>E</sub>X cross-referencing commands (including the table of contents) to produce `\special` commands which are parsed by DVI processors conforming to the HyperT<sub>E</sub>X guidelines (i.e. `xhdvi` and `dvihps`); it also provides general hypertext links, including those to external documents.

The HyperT<sub>E</sub>X specification<sup>6</sup> says that conformant viewers/translators must recognize the following set of `\special` commands:

**href:** `html:<a href = "href_string">`

**name:** `html:<a name = "name_string">`

**end:** `html:</a>`

**image:** `html:<img src = "href_string">`

**base\_name:** `html:<base href = "href_string">`

The *href*, *name* and *end* commands are used to perform the basic hypertext operations of establishing links between sections of documents. The *image* command is intended (as with current HTML viewers) to place an image of arbitrary graphical format on the page in the current location. The *base\_name* command is used to communicate to the *dvi* viewer the full (URL) location of the current document so that files specified by relative URL's may be retrieved correctly.

The *href* and *name* commands must be paired with an *end* command later in the T<sub>E</sub>X file – the T<sub>E</sub>X commands between the two ends of a pair form an *anchor* in the document. In the case of an *href* command, the *anchor* is to be highlighted in the *dvi* viewer, and when clicked on will cause the view to shift to the destination specified by *href\_string*. The *anchor* associated with a name command represents a possible location to which other hypertext links may refer, either as local references (of the form `href="#name_string"` with the *name\_string* identical to the one in the name command) or as part of a URL (of the form `URL#name_string`). Here *href\_string* is a valid URL or local identifier, while *name\_string* could be any string at all: the only caveat is that ‘” characters should be escaped with a backslash (\), and if it looks like a URL name it may cause problems.

The *hyperref* package redefines or overloads a lot of L<sup>A</sup>T<sub>E</sub>X macros to express all the common constructs in terms of this generic functionality. It is hoped that the redefinition is robust, but some aspects of it are quite complex, and some other packages may conflict with it – it should always be loaded last! Anything which uses cross-referencing and the internal `\setref` command should convert, but sophisticated packages like AMSL<sup>A</sup>T<sub>E</sub>X can cause problems.

The package supports the following options:

**draft** makes the low-level macros no-ops;

**colorlinks** colors the links and anchors (this needs the standard L<sup>A</sup>T<sub>E</sub>X<sub>2<sub>ε</sub></sub> color package).

The colors can be changed by redefining two macros; the default setting is:

---

6. This description is derived from Arthur Smith's documentation.

```
\def\LinkColor{red}
\def\AnchorColor{blue}
```

**nocolorlinks** turns off coloring, if it has been activated by default;

**backref** if the *backref* package is used, which lists citation points for each entry in the bibliography, this option sets up back-referencing to be hyper links by section number;

**pagebackref** sets up back-referencing by page number;

**hyperindex** makes index entries be links back to the relevant pages;

**nohyperindex** disables hypertext indexing;

**plainpages** in this package, every page is make a target for links; this option normalizes all page numbers to be plain arabic, since typesetting commands like `\textbf` can cause the main *hyperref* macros to break;

**noplainpages** turns off the above behavior, so that sequences like roman numbering of a preamble is respected;

**hyperfigures** makes included figures (assuming they use the standard graphics package) be hypertext links;

**nohyperfigures** turns off the above behavior;

**nonesting** currently, *dvihps* doesn't allow anchors to be nested within targets, so this option tries to stop that happening. Other processors may be able to cope;

**nesting** allows nesting to take place;

The following options are the default: *nocolorlinks*, *noplainpages*, *nonesting*, *hyperindex* and *nohyperfigures*

## 4.2 Creating an enriched PDF file with REPERE

As we can see in Figure 2, Acrobat gives us the possibility of displaying a hierarchical, active, table of contents on the left-hand side of the window. The *dvihps* program does not, in its current version, directly support this facility; to remedy this lack, Yannis Haralambous developed a post-processor for the output of *dvihps* which creates the necessary extra material. The program, *reper*, is written in Flex, and can be compiled on most platforms with a Flex implementation and a C compiler.

The *reper* program works in conjunction with the *hyperref* package, whose macros write all sectioning titles to an external file with the suffix `.rep`. After processing the file with L<sup>A</sup>T<sub>E</sub>X, and running *dvihps*, the `.rep` file is prepended and appended to the PostScript file, and the result run through *reper*. For a file `foo.tex`, the sequence would be (for Unix, or other systems with pipes:<sup>7</sup>)

```
latex foo
latex foo
dvihps -z foo -o footemp.ps
cat foo.rep footemp.ps foo.rep | reper > foo.ps
```

7. DOS or VMS users will have to use copy/append commands to create a temporary file

This would result in a PostScript file, `foo.ps` which can be given to Adobe Distiller which will produce the table of contents. The *reper* program works by writing `pdfmark` commands for Distiller.

The trickiest part of the operation is the conversion of the encoding of the  $\text{\LaTeX}$  file which is written to the `.rep` file into the PDF Encoding (an combination of the Windows, Mac and Adobe Standard encodings) needed for the table of contents. When  $\text{\LaTeX}$  writes the `.rep` file, it may expand accented characters and the like, depending on the encoding used; command sequences like `\TeX` also get expanded to strange forms. While *reper* tries to locate accented letters and replace them with the 8-bit equivalent from the PDF Encoding, there remain considerable problems in getting a totally clean table of contents without some manual editing. Luckily, this affects only the appearance – the hypertext links between the table of contents and the main document remain intact regardless of how horrible the contents may look.

### 4.3 Problems at the $\text{\TeX}$ level

The fact that `dvi` files were designed solely to produce printed pages means that we have to take some precautions when preparing material which is to be converted to PDF.

The precautions have largely to do with the fonts used in the document. The biggest problem for a program like Acrobat, which sets out to display and print any PostScript file whatsoever, is the range of PostScript fonts used in the document. 99% of the existing PostScript fonts (and there are thousands of them...) are commercial, and their usage is determined by the license agreement between the vendor and the user. How do we arrange it so that an author can distribute a document using one of these fonts, and be sure that the reader has a copy of the same font?

Adobe solved this problem with the *Multiple Master* technology; this is similar to the principles of METAFONT<sup>8</sup>, by which fonts have certain meta-characteristics which can be varied to produce different looking glyphs (in terms of their weight, width, etc. along up to four axes). Using the extended Adobe Type Manager (Super ATM, or ATM version 3), and two Multiple Master fonts (one serif, and one sans-serif), Acrobat is able to mimic the look of any PostScript font which is not present on the *reader's* system. The Acrobat document simply contains the font name, and a set of metrics; if the font can be found, it is used, but otherwise a Multiple Master instance is created to get (at least) the weight, spacing and size right.

Can Multiple Masters mimic *any* font? Not quite. If the font has a non-standard set of characters (i.e., it is not a Latin text font), such as mathematics, phonetic symbols or Greek, simply substituting characters from a text font will obviously produce catastrophic results. There are two solutions to this:

1. The 'exotic' font can be fully imbedded in the PDF document, so that it is available to the viewing system. This avoids the problem of inappropriate Multiple Master substitution, but raises copyright issues – the author needs permission from the

---

8. Compared to METAFONT, Multiple Master fonts are in fact quite simplistic.

font vendor to distribute it in this way. In Version 2 of Acrobat, Adobe implemented partial font downloading – for each font used, Distiller makes a subset containing just those characters actually used. This makes for smaller files, and goes a considerable way toward alleviating the fears of font vendors, many of whom do now permit their fonts be in distributed in this partial way.

2. In the case of T<sub>E</sub>X, fonts can be included in PK bitmap format. The copyright problem does not arise, since only bitmap representations are included in the PDF file.<sup>9</sup> Unfortunately, Acrobat Reader does not display such bitmap fonts at all well, since they need to be reduced for screen resolution, and the characters usually appear very emaciated. Printing, by contrast, presents no problems, if the resolution of the bitmap font corresponds to that of the printer, rather than the screen.

A third solution is to avoid the problem by using the standard fonts which you can be almost certain are available for any PostScript device (Times, Helvetica, Symbol, Courier, Palatino etc). Unfortunately, we cannot produce any mathematics or Greek of more than trivial quality using the Symbol font, so this approach is of limited effectiveness for traditional L<sup>A</sup>T<sub>E</sub>X documents.

A practical approach for mathematics is to use the Computer Modern fonts for symbols, and Times for alphanumeric characters (this can be done using Alan Jeffrey's *mathptm* package), and to use PostScript Type1 versions of the CM fonts. These can be purchased from Blue Sky Research, and Y&Y Inc, or there are free versions in the CTAN archives of almost equal quality. Prospective users of these latter fonts should check the license conditions which only allow non-commercial use.

A final problem to consider is the possible ill-effect of virtual fonts which produce accented characters by combining separate accents and characters (such as can be done by Alan Jeffrey's *fontinst* package). The reason for this is that Acrobat has a facility to search for strings in documents; if accented characters are in fact represented in the PostScript/PDF file by two separate glyphs, searching will not be complete or accurate (whereas genuine 8-bit characters can be searched for and found). For example, if the word 'dégénére' is represented as

`de<acute accent>ge<acute accent>ne<acute accent>re<acute accent>`  
in the PDF document, then a search for *dégénére*, where é is an 8-bit character, will not be successful.

The solution to this problem is to use PostScript fonts encoded in the L<sup>A</sup>T<sub>E</sub>X T1 (Cork) standard, and based on re-encoding at the PostScript level to allow access to the full range of accented characters. How this is achieved is beyond the scope of this article, but the CTAN archives contain sets of metrics for many common PostScript fonts derived in this way, suitable for immediate use. Some characters like ž are simply not present in most fonts, and so these will always have to be created by composite characters, but most Western European languages will come out 'correctly'. It is worth

---

9. However, if the bitmaps are derived from a commercial PostScript font, the user would be well advised to check with the vendor that bitmaps *can* be distributed in this way.

pointing out that  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}_{2_{\epsilon}}$  will automatically transform 7-bit markup like `\'e` into the 8-bit single character on output, if T1 encoding is used.

## 5 DVI to (hyper)PostScript

Like  $\text{T}_{\text{E}}\text{X}$ , *dvips* is a good example of a very high-quality public domain program, available for almost all operating environments and producing good quality PostScript output. In order to get the most out of the translation to PDF, however, it is necessary to alter the program a little. Mark Doyle undertook this task, and the result is the *dvihps* variant of *dvips*, which also runs on all systems.<sup>10</sup>

Why are changes necessary? To define hypertext links, Acrobat Distiller needs (at least) two bits of information: the active 'button' area, and the document element to be displayed. These areas are defined in terms of rectangular areas, whose page coordinates are given in PostScript points (72 to the inch) in relation to the bottom left corner of the page. In order to establish the coordinates of the target area, which may occur pages after the point of departure, it is necessary to make an extra pass through the output, after all the text has been positioned in PostScript coordinates. While it would theoretically be possible to program all this at the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  level, the transformation from DVI coordinates to PostScript coordinates is distinctly hair-raising, and it seems sensible to leave this to the modified *dvips* program. At all the points where links are desired, `\special` commands are inserted into the output by  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  macros, and these are converted by *dvihps* if the new `-z` command line option is used.

We may note that the PostScript file produced by *dvihps* contains code in the preamble to deactivate the hypertext commands if the file is processed by an application other than Acrobat Distiller. It also detects different versions of Distiller, since version 2 has more advanced features than version 1, which are used if possible.

## 6 PostScript to PDF

This stage, which is certainly the longest in terms of elapsed time for the user, is entirely under the control of the Acrobat Distiller program; anyone wishing to create serious PDF documents needs to purchase a copy. It is, on the side, a very good debugger of PostScript programs, and a good interpreter. It is a good way to preview any PostScript file, although the processing is rather slow.

## 7 Viewing, navigation, and printing of a PDF file

These operations are achieved with the help of the Adobe Acrobat Reader software. Search functions, zooming, navigation, text copy, etc, are available from menu options

---

<sup>10</sup>. It is to be expected that the functionality will be merged back into the 'real' *dvips* by Tom Rokicki in due course.

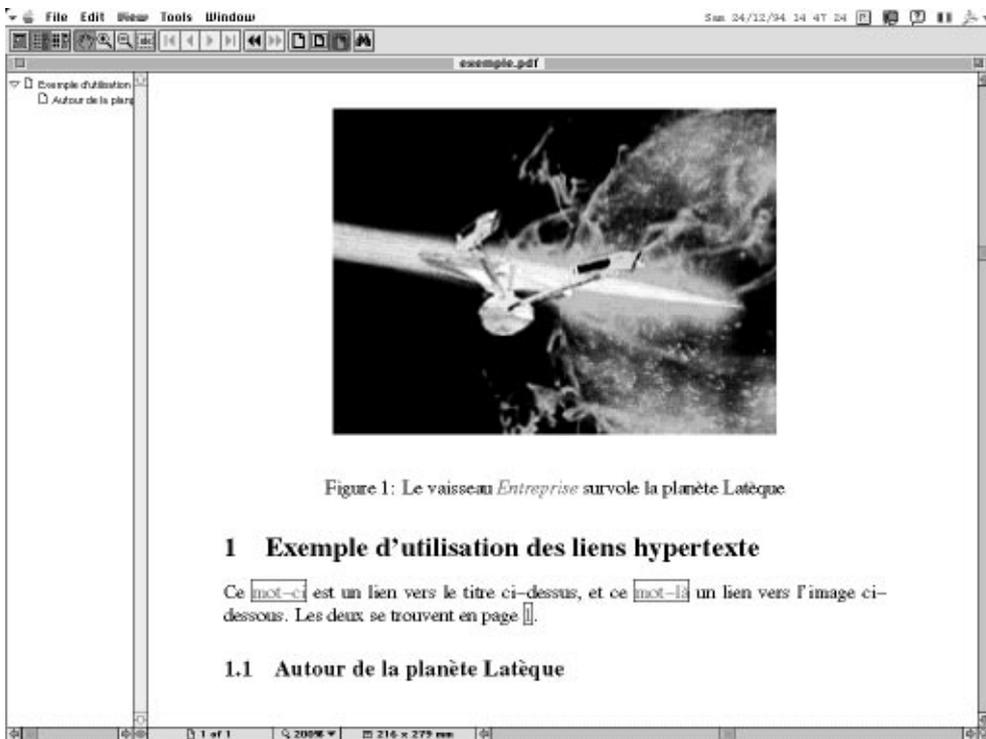


Figure 2: PDF file being displayed with Acrobat Reader

or key combinations. To compare the screen presentation of a T<sub>E</sub>X file, the reader will see in Figure 2 a copy of an Acrobat Reader screen on a Macintosh, and in Figure 3 the printed version of the same document. The L<sup>A</sup>T<sub>E</sub>X *hyperref* package allows the user to choose the presentation of active areas of hypertext links (in red by default) as well as the anchor areas (in green by default). The PDF format also allows us to frame the active areas.

### 7.1 Some information on the PDF format

The PDF format is even more hermetic and incomprehensible to the average user than the PostScript language. However, it is interesting to know a bit of its structure, to perform, if needed, some minor modifications to the presentation file (the PDF format is still quite new and we desperately lack tools to modify PDF documents).

A PDF file can be either a 7-bit ASCII file or an 8-bit binary file. It consists of four parts: the *header*, the *body*, the *cross reference table* and the *trailer*. The header, for the current version, is composed of a single line: %PDF-1.1. The body is composed of objects: each page is an object; the links, the notes, the marks, the font codes, the

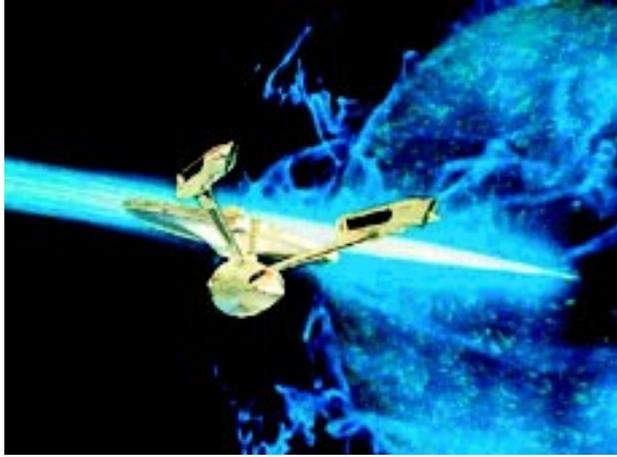


Figure 1: Le vaisseau *Entreprise* survole la planète Latèque

## 1 Exemple d'utilisation des liens hypertexte

Ce [mot-ci](#) est un lien vers le titre ci-dessus, et ce [mot-là](#) un lien vers l'image ci-dessous. Les deux se trouvent en page 1.

### 1.1 Autour de la planète Latèque

Figure 3: Result of printing the PDF file

font descriptors, and the systems for color description, are all objects. The advantage of using objects is that one can change the order, insert or remove pages, without breaking the existing hypertext links: the order of the pages is kept in the cross reference table, deleted pages are kept in the document and are only virtually removed. Each change will lead to a change in the cross reference table and in the trailer. A PDF display application starts by reading the end of the document, and retrieves the cross reference table of pointers to the objects in the document.

Most of the objects are compressed and then coded in 7 bits; four methods of compression can be used: Lempel-Ziv, run length, CCITT Fax group 3 or 4, and JPEG; then two methods can be used for the conversion to 7 bits: hexadecimal or "ASCII base 85" notation.

Adobe Acrobat Distiller allows for compression to be turned off, although this is not very interesting, since no tool is provided to allow for *a posteriori* compression of modified PDF objects.

We will describe here only some of the non-compressed objects, which can be freely modified by the user. However, it should be noted that each modification of a PDF file (except one that will be mentioned below) needs an update of the cross reference table: this table contains, for each PDF object, its offset relative to the start of the document. Each object has a number, which is the first item data for the object. The objects are not necessarily ordered by number in the PDF file. The cross reference table contains one line for each object; this line contains the offset of the object to the start of the file (a number of 10 digits), followed by a blank, a 5 digit number which is the number of times this object has been modified, another blank, and the letter 'n'. If the object is deleted, the number of the object will be available and the syntax of this line will change: the 10 digit number indicates the number of the next free object in the table (it is nil if it is the last free object) and the letter 'f' at the end of the line.

Every time an object is modified, it is necessary to change the offset of all the objects which physically follow it in the file; we must also change a number at the end of the file which indicates the offset of the table of cross-references relative to the start of the file.

As an example, the following is an extract from a PDF file, showing the start, the first object (a color descriptor), the last few objects, part of the cross-reference table, and the trailer.

```
%PDF-1.1
1 0 obj
  [/CalRGB
  <<
  /WhitePoint [0.9505 1 1.089]
  /Gamma [1.8 1.8 1.8]
  /Matrix [0.4497 0.2446 0.02518 0.3163 0.672 0.1412 0.1845 0.08334 0.9227]
  >>
  ]
endobj
.....
9 0 obj
<<
  /Type /Pages
  /Kids [2 0 R 10 0 R 14 0 R 20 0 R ]
  /Count 4
  /MediaBox [0 0 612 792 ]
  >>
endobj
41 0 obj
<<
  /Type /Catalog
  /Pages 9 0 R
  >>
endobj
42 0 obj
```

```

<<
/CreationDate (D:19950420210508)
/Producer (Acrobat Distiller 2.0 for Windows)
>>
endobj
xref
0 43
0000000000 65535 f
0000000017 00000 n
0000251348 00000 n
0000000182 00000 n
0000021336 00000 n
.....
0000211955 00000 n
0000220508 00000 n
0000251821 00000 n
0000251877 00000 n
trailer
<<
/Size 43
/Root 41 0 R
/Info 42 0 R
/ID [a7b776d0fb5478b29f5739c089a2c83f><a7b776d0fb5478b29f5739c089a2c83f>]
>>
startxref
251984
%%EOF

```

The number 251984 is the offset from the start of the file of the beginning of the cross-reference table. An extract from an object in the main part of the file shows the uncompressed version of some text being displayed:

```

3 0 obj
<<
/Length 21095
>>
stream
BT
/F4 1 Tf
7 0 0 7 72 759.67 Tm
0 Tr
0 g
0.014 Tc
[(T)108(e)7(s)19(t)-363(of)-352(c)7(m)25(r)14(1)0(0)-343(o)
0(n)-349(A)0(p)27(r)14(i)27(l)-383(20,)-349(1995)-308(at)-363(1712)]TJ
ET
129.36 719.59 0.48 -16.08 re
...
\end{verbatim}
A hypertext link is an object of type 'Annot'; an example is
\begin{verbatim}
17 0 obj
<<
/Type /Annot
/Subtype /Link
/Rect [107 565 171 577 ]
/Dest [16 0 R /FitH 842]

```

```

/T (page.5)
/C [0 0 1 ]
/Border [0 0 1 [3 3 ]
]
>>
endobj

```

While the `/Rect` key simply gives the coordinates of a rectangle around a link area, the `/Dest` area is more interesting. In this example, it points to a page number, and says that the page is to sized to fit a certain height, but it can also (in Version 2) point to an external file, or 'named' destination. This allows us to have the same functionality as HTML, opening another file at a named point, rather than having to know the actual page number and position in the other file.

The `/Border` key describes the appearance of the link; in Version 1, this was either a frame or nothing, but Version 2 allows for colored frames, and different line types. The values in this example indicate that the 'active' area which is to be clicked on is outlined with a blue dashed line (the color is given by the `/C` key, an abbreviation for `/Color`).

How can we modify this file? At the end of the example above, we see the key `/Producer` (Acrobat Distiller 2.0 for Windows); we may want to change this object, and use some of the other available keys, to produce:

```

/Author (Mr Kipling)
/Title (My favorite PDF sample)
/Creator (LaTeX, of course)

```

It is easy to simply edit this in, but we would also have to go through and change the cross-reference table for all the objects that follow it, a tedious and error-prone procedure. Yannis Haralambous has written another Flex program, *recticrt*, which performs this task for you, reading a PDF file and writing a new version with a checked and updated cross-reference table.

Full documentation of the PDF format can be found in [1], and in the PDF documents distributed with Acrobat Distiller.

## 8 Conclusions

We have tried to show in this paper that a complete, viable, electronic publishing system can be built with L<sup>A</sup>T<sub>E</sub>X as its base, and the Portable Document Format as its delivery medium. While the tools we describe, and those we have developed ourselves, are functional, we believe that only a small part of the potential has been realized. We hope that others will develop more tools to make richer and richer electronic documents, using T<sub>E</sub>X typography as a solid foundation.

### Obtaining the programs

The *hyperref* package can be obtained from any of the CTAN (Comprehensive T<sub>E</sub>X Archive Network) archives, from the directory

macros/latex/contrib/supported/hyperref. The *reper* and *recticrt* programs are supplied in source form (Flex code) and as compiled MSDOS binaries. The source of *dvihps* is available in *dviware/dvihps* in the CTAN archives, and an MSDOS binary is also stored in the *hyperref* directory.

The HyperTeX project, whose standards form the basis of the work described in this article, should be visited on the World Wide Web at <http://xxx.lanl.gov/hypertext/>.

Michael Mehlich has written another L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  package for encapsulating hypertext functionality in L<sup>A</sup>T<sub>E</sub>X output, to the same HyperTeX standards as *hyperref*, with comparable functionality. This is available on CTAN in `macros/latex/contrib/supported/hyper`.

The PostScript Type1 versions of the Computer Modern fonts by Basil Malyshev (the BaKoMa collection) can be obtained from CTAN, in the directory `fonts/cm/ps-type1/bakoma`.

The free Acrobat Reader for Windows, Macintosh and Sun Unix can be obtained from Adobe (Internet FTP site <ftp.adobe.com>, for instance) or from many other collections. In order to create good quality PDF files from PostScript, it is necessary to purchase the more expensive Acrobat Pro package (the PDF Writer included with Acrobat Exchange does not translate all the `pdfmark` information) from Adobe Systems.

## References

- [1] T. Bientz and R. Cohn. *Portable Document Format Reference Manual*. Addison Wesley, 1993.
- [2] M. Goossens and J Saarela. From L<sup>A</sup>T<sub>E</sub>X to html and back. *to be published in TUGboat*, 1995.
- [3] T.V. Raman. An audio view of T<sub>E</sub>X documents. *TUGboat*, 13.4, 1992.

# The release 1.2 of the Cork encoded DC fonts and the text companion symbol fonts

Jörg Knappen

Institut für Kernphysik  
Johannes Gutenberg-Universität Mainz  
D-55099 Mainz  
knappen@vkpmzd.kph.uni-mainz.de

## Abstract

I present the release 1.2 of the dc fonts and the companion text symbol fonts. I give an overview of the improvements on the dc fonts from version 1.1 to 1.2. The rationale for introducing a text symbol font is explained and the text symbol encoding TS1 is presented. In the appendix, there are font tables of the mentioned fonts.

## 1 Introduction

In 1990 at the TUG meeting at Cork, Ireland, the european T<sub>E</sub>X user groups agreed on a 256 character encoding supporting many european languages with latin writing. This encoding is both an *internal encoding* for T<sub>E</sub>X and a *font encoding*. This double nature is a consequence of the fact, that both kind of encodings cannot be entirely separated within T<sub>E</sub>X.

The design goals of the Cork encoding are to allow as many languages as possible to be hyphenated correctly and to guarantee correct kerning for those languages. Therefore it includes many ready-made accented letters.

It also includes some innovative features, which have not become very popular yet, though they deserve to become so. First to mention is a special, zero width invisible character, the compound word mark (cwm). The second is the separation of the two characters <hyphen> and <hyphenchar>. By appropriate design of the hyphenchar glyph, hanging hyphenation can be achieved.

A group around Norbert Schwarz started the implementation of a Cork encoded font in METAFONT in 1990. The first release was published in 1990, called dc fonts. It was stated that after some improvement and bug fixing they should be finally named

ec fonts. A second release was made public in 1993. Unfortunately Norbert Schwarz has not the time to complete the ec fonts, and he has resigned from the work at the DANTE meeting at Münster in spring 1994. The author of this article now coordinates the further development of the ec fonts.

In the meantime, other Cork encoded fonts have become available: The Malvern fonts, a sans serif face, are implemented with METAFONT and contain the full Cork character set. Many commercial faces are available in the Cork encoding implemented as virtual fonts with the fontinst package by Alan Jeffrey [4]. Those fonts lack some characters, especially the dotless j and the letter eng, which are replaced with black blocks and produce warning messages. The reason is, that there are no glyphs for them in the basic fonts and it turns out to be hard to fake them.

The need for a text companion font was first articulated in the discussion of new 256 character mathematical fonts in 1993. In order to achieve a better orthogonality between text and math, some text symbols stored in the math fonts should be moved to the text companion fonts.<sup>1</sup> The text companion fonts are also the ideal place to store some new characters, like currency symbols.

Users of commercial fonts with expert sets want to access the complete set of glyphs provided. Again, a text companion font is the appropriate place to store those glyphs.

## 2 Improvements to the DC fonts

### 2.1 Accents

In good typography, the accent marks should look different for capitals and lowercase letters respectively. The accent over a capital should be of a 'flat' design, while the accent on a lowercase letter should be 'steep'. The Computer Modern fonts by D.E. Knuth only have steep accents, well made for lowercase letters. The accented capitals grow too tall, leading to uneven line spacing.

óx ÓX

Figure 1: Letters with acute accent in the cmr font

There are no readily accented letters provided which leads to problems with proper hyphenation and kerning. However, the floating accent approach guarantees the consistency of all accented letters.

With the version 1.1 of the dc fonts, the situation is different. We have readily designed accented letters for all languages included in the ISO standards 8859-1 and 8859-2. If an 'exotic' accented letter is needed, it does not fit to the provided ones.

1. The archives of the math-font-discuss mailing list are available for ftp on `ftp.cogs.susx.ac.uk` in directory `pub/tex/mathfont`.

The image shows the lowercase letter 'ó' and the uppercase letter 'Ó' followed by the lowercase letter 'x' and the uppercase letter 'X'. The acute accents on the 'ó' and 'Ó' are notably steep and appear to be floating above the letters.

Figure 2: Letters with acute accent in the dcr font (v 1.1)

Note that the floating acute accent is the same for capitals and lowers, but different from both, being even steeper than the lowercase one.

With the version 1.2 of the dc fonts, all inconsistencies have gone. The accents are different between capitals and lowers as they should be, and floating accents can be applied in a consistent manner.

The image shows the lowercase letter 'ó' and the uppercase letter 'Ó' followed by the lowercase letter 'x' and the uppercase letter 'X'. In this version, the acute accents are more integrated with the letters, appearing less like floating marks compared to the previous version.

Figure 3: Letters with acute accent in the dcr font (v 1.2)

Since the Cork encoding does only provide one slot for each accent, the capital acute accent is taken from the *text companion* font tcr. This is possible, because T<sub>E</sub>X allows cross-font accenting.

The acute accent and the readily accented letters were taken with kind permission of the authors from the polish pl fonts [3], which provide the highest available quality for these shapes.

The hungarian double acute accent and the grave accent follow the design of the acute accent.

## 2.2 Quotation marks

The design of quotation marks provides another challenge to the ec fonts. In the Computer Modern fonts, they are optimised to english usage.

The image shows two square boxes. The left box contains a left double quotation mark (‘‘) and the right box contains a right double quotation mark (’’). The marks are positioned asymmetrically within their respective boxes, with more space on the left for the opening mark and more space on the right for the closing mark.

Figure 4: Quotation marks in the cmr font

They lie asymmetrically in their boxes, which widens the space before and after a quotation. However, this kind of design produces a disaster, if the same english opening quotation mark is used as a german or polish closing quotation mark. Currently, macros have to compensate for this.

In the dc fonts v 1.2, the quotation marks are lying symmetrically in a tighter box, the additional space is generated using kerning against the `boundarychar`.



Figure 5: Quotation marks in the dcr font (1.2)

The `boundarychar` feature was introduced with  $\text{T}_{\text{E}}\text{X}3$  and `METAFONT2`, it is reasonable to assume that nowadays every  $\text{T}_{\text{E}}\text{X}$  user has access to these or later versions.<sup>2</sup>

### 2.3 Miscellaneous

The shapes for polish letters are now taken from the polish `pl` font, leading to improved shapes on the ogoneked letters and the crossed `l`.



Figure 6: Polish special letters in the dcr font (1.2)

With the help of the czechoslovak  $\text{T}_{\text{E}}\text{X}$  users' group, the shapes of czech and slovak special letters [7] have been improved, too.



Figure 7: Some czech and slovak special letters in the dcr font (1.2)

The height of umlaut dots has been adjusted to the value contributed by the czechoslovak group (`ä` occurs in slovak), the value used in the version 1.1 of the `dc` fonts was considered too low even by german users.

Figure 8: The letter `ä` in `cmr`, `dcr v1.1`, and `dcr v1.2`

The `hyphenchar` is now designed to hang out of its bounding box, thus allowing for hanging hyphenation.

*The release 1.2 also contains a new shape, a classical serif italic font. It was already prepared in version 1.1, but no parameter and driver files were present for it.*

<sup>2</sup> Maybe it was not a reasonable assumption in 1990, when the Cork encoding was born and the above mentioned versions were brand new.



Figure 9: Hyphen and hyphenchar with their bounding box

*It is an italic with upper serifs instead of ingoing hooks. This paragraph is typeset the dcci font to show its appearance.*

### 3 The TC fonts

#### 3.1 A text symbol encoding

Over the years, many reasons have accumulated for a new text symbol encoding. There are some text symbols stashed in the math fonts, the footnote marks (\*, §, ¶, †, ‡, ||) and the bullet (•) are among them. In 256-character math fonts they should not be preserved, but moved to a text symbol encoding.



Figure 10: Footnote symbols from the tc font

The ISO standards 8859-1 (Latin 1), 8859-2 (Latin 2), and 6937 contain several custom signs. It will be easier to typeset text encoded according to those standards if the necessary symbols were easily accessible through a text symbol font.

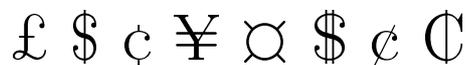


Figure 11: Some currency signs from the tc font

Finally, I wanted to have different style accents for capitals and lowercase letters. Since the Cork encoding does not have the space for another thirteen accent glyphs, I decided to have the lowercase accents which are far more often needed in the dc fonts, and to put the accents for capital letters into the text companion fonts.

The users of commercial font also want to access all glyphs stored in those fonts. Since most of those glyphs are textual, they all should be included into a text symbol font encoding.

#### 3.2 The font encodings TS1 and TSA

For mainly technical reasons, I think the candidates for a text symbol encoding should be distributed over two fonts, their encoding named TS1 and TSA respectively. There

are important differences between the technology supported by METAFONT and  $\TeX$  compared to the path most commercial font suppliers choose.

The computer modern family of fonts supports the notion of a designsize, i. e. there are subtle differences between the shapes at different point sizes as illustrated in the next section.  $\TeX$  is able to raise and lower letters, thus it does not need a readily raised digit to produce a superscript. It can also produce nice fraction using a macro from the  $\TeX$ book, exercise 11.6, like  $\frac{1}{2}$ ,  $\frac{1}{4}$ , and  $\frac{1}{6}$ .

Most commercial vendors took the easier path, their fonts come in only one size and are scaled up and down to the other sizes. Thus, a small superscript does not look right, and to compensate this a readily designed superscript is added to the fonts. A subscript is also added, because earlier text processors weren't able to raise or lower letters. For similar reasons, fraction glyphs were provided, or fractions were constructed out of a sequence  $\langle$ superscript digit $\rangle$   $\langle$ fraction $\rangle$   $\langle$ subscript digit $\rangle$ , where  $\langle$ fraction $\rangle$  is a special slash to construct fractions.

On the other hand, it is almost impossible to follow this path with  $\TeX$  and METAFONT: The size of the superscripts can be influenced by  $\TeX$  macros, and therefore there is no unique 'virtual designsize' for ready-made superscripts.

The selection of superscripts offered by commercial vendors is at the moment rather sparse, many often needed ones are lacking.

2<sup>ième</sup> 5<sup>th</sup> M<sup>c</sup>

Figure 12: Some superscript letters missing in expert fonts

Therefore the rule of thumb for the distribution of glyphs is the following: Put all glyphs which can be conveniently made with METAFONT and are needed with  $\TeX$  into the encoding TS1, and put the remaining glyphs, mainly superscripts and subscripts, into the encoding TSA. There are some duplications and deviations from this rule of thumb, e. g. superscript 1, 2, and 3 are part of ISO 8859-1, thus they occur in TS1 as well as in TSA.

### 3.3 Contents of the TS1 encoding

In the first thirteen positions are accents for capital letters. There are two new dashes with a length between endash and emdash, the longer of them corresponding to  $\langle$ threequartersemdash $\rangle$ . Oldstyle digits are included to facilitate the building of virtual fonts with oldstyle digits instead of the usual ones. The genealogical symbols from Knuth's gen fonts are included, as well as the remainder of the latin 1 and latin 2 standards. ISO-6937 contributes the arrows, the musical note, the trademark sign, and the Ohm sign, the mho sign is added for consistency. The two always troublesome glyphs \$ and \$ have found their final rest in the TS1 encoding. Note, that there are also oldstyle

variants of dollar and cent provided, as well as a lira sign different from pounds. According to the dutch authority Karel Treebus, the abbreviation for dutch guilders should just be the letter 'f' out of the current font, therefore the florin sign is designed to be identical to the letter f in the tc fonts. Other designers may not agree here.

There should be a kern between <degree> and <C> to form a proper centigrade sign. Since kerning is impossible (the two characters live in different fonts) a ready-made centigrade symbol is included. Last, but not least, a ready-made per thousand sign is included.

The assigned code points of the TS1 encoding are listed in appendix A of this paper. A font table of the TS1 encoded font tcr1000 can be found in appendix B.

## 4 Using the full power of the DC fonts

The dc fonts support very strongly the notion of a design size. Just as in the famous example sentence “Ten point type is different from magnified five point type” [6] exemplified, linear scaling a font over a large range of sizes gives wrong results. In fact, scaling should be avoided at all. Instead, the designsize fonts should be used.

The file `dcstdedt.tex` generates small parameter files for the standard plain  $\text{\TeX}$  and  $\text{\LaTeX}$  sizes. However, a class author may want to use unconventional sizes such as 9.5pt.

### 4.1 A new naming scheme

Here the problem occurs, how should a font with designsize of 9.5pt should be *uniquely* named. Of course, a completely general solution to the problem, which is also compatible with the famous 8+3-restriction of some operating systems, cannot be given. For example, any scheme will fail to name dcr with a designsize of 3.14159 dd.

Let us make some reasonable assumptions: The designsize is given in usual  $\text{\TeX}$ 's points (1/72.27 inch), it is less than 100pt, and the accuracy is two trailing digits (the same level of accuracy is employed by plain  $\text{\TeX}$  and  $\text{\LaTeX}$ ). Under these circumstances, the designsize can be given by four digits, the canonical sizes are represented by 0500, 0600, 0700, 0800, 0900, 1000, 1095, 1200, 1440, 1728, 2074, and 2488. The largest possible design size would be 9999 or 99.99pt.

We are left with four (or less) letters for the specification of the font. Here we can resort to the scheme described by Knuth [5] which compressed the Computer Modern font names to six characters, two of them denoting the design size.<sup>3</sup>

With this scheme, the font name consists of the two letters 'dc', one or two letters denoting the weight and shape, and four digits giving the design size. Most font names are unchanged except for the new way of specifying the designsize. Some names

---

3. This scheme was given by the rule: “Take the first three character + the last three characters to get an unique font name”.

which are contracted include `dcbi < dcbxti`, `dcsi < dcssi`, and `dcsq < dcssqi`. An irregularity occurs through `dcqi < dcssqi`.

## 4.2 Exploiting NFSS2

With the New Font Selection Scheme (NFSS2) of  $\text{\LaTeX}2_{\epsilon}$ , it is in principle possible to load the dc fonts at arbitrary size without resorting to magnification.

To achieve this, the following is necessary: Test, if the `tfm` file for the requested size is already available. If not, issue a warning and write an entry to a file `missing.bat`. Proceed with a magnified font, but write a note to the terminal, that the missing font should be generated. This should be programmable in NFSS2, the main difficulty is testing the existence of the `tfm` file.

There are two possible methods, both having their drawbacks.

1. Actually try to load the font, but do this in `\batchmode` to skip the two errors that could occur. The difficulty is to switch back to the original interaction mode after this. The failure can be detected by looking at some `fontdimension`.
2. Softly open the `tfm` file with `\openin`. Here you need to specify the path, where the `tfm` file is to be searched, except your  $\text{\TeX}$  implementation also searches `TEXFONTS` if you try to `\openin` a file. This needs an additional entry in the  $\text{\LaTeX}2_{\epsilon}$  file `texsys.cfg`.

At the time this paper is written, no decision on the final algorithm has been made, nor does a test implementation exist.

## 5 Outlook and summary

### 5.1 Changes to the Cork encoding?

In past years, several people have suggested some changes to the Cork encoding to support one other language, which is currently not completely supported, and to take out one letter or some symbols currently in the Cork encoding. Language missed by just one letter include catalan, azeri, and sami.

On the other hand, the Cork encoding has now really caught on and has a growing user base. It is not only implemented in two `METAFONT` font families (dc fonts and Malvern fonts), but also in virtual fonts for many popular commercial faces. It has become the T1 encoding of  $\text{\LaTeX}2_{\epsilon}$  and is well documented in the  $\text{\LaTeX}2_{\epsilon}$  literature [2]. Any change to the encoding would divide the user base. It would also slow down the completion of the dc fonts.

Therefore I suggest not to change the Cork encoding anymore. There are more font encodings (real or virtual) for latin fonts to come. Presently there are T1, the Cork encoding covering latin-1 and latin-2, and T4, the fc encoding for african latin. At least three more are needed, one for vietnamese which has plenty special letters, one covering

latin-3 and latin-6 (catalan, maltese, esperanto, baltic languages, sami)<sup>4</sup>, one for celtic languages (welsh, irish gaelic, breton, scotish gaelic), and maybe a fourth one for native american languages.

## 5.2 Suggestions to $\varepsilon$ -T<sub>E</sub>X

There are a few points, where an extension to the current T<sub>E</sub>X is suggested. The first point is the wish to switch between modern and oldstyle digits easily. At the moment, this can be done in math mode by just changing ten `\mathcodes`. However, in text mode one has either to tag each number or digit (including those generated by macros) explicitly as oldstyle, or to resort to virtual fonts, replacing the complete set of text fonts. Both solutions look very much like overkill to me, a decent `\textcode` command would be helpfull here.

Second, T<sub>E</sub>X can controll the space (both the extension and its stretch or shrink) by the help of the spacefactor. However, there is no mean to influence the space *before* a special character.

## 5.3 Promotion from DC to EC

The dc font should be promoted to the than fixed ec fonts as soon as possible, but not sooner as possible. As long as I am able to do, I'll stay the contact for bug fixes to the dc fonts. Hopefully in about a year, we can see the advent of the ec fonts.

## References

- [1] J.S. Bień. *Polish texts in multilingual environments (a case study)*. Proceedings of the eighth european T<sub>E</sub>X conference, Gdańsk, Poland, September 1994.
- [2] M. Goossens, F. Mittelbach, and A. Samarin. *The L<sup>A</sup>T<sub>E</sub>X companion*. Addison-Wesley, Reading, Mass, 1994<sup>1</sup>.
- [3] B. Jackowski and Ryćko M. *Polishing T<sub>E</sub>X: From ready to use to handy to use*. Proceedings of the seventh european T<sub>E</sub>X conference, Prague, Czechoslovakia, September 1992.
- [4] A. Jeffrey. *Building virtual fonts with fontinst*. File `bville.tex`, distributed with fontinst through the CTAN archives, 1994.
- [5] D.E. Knuth. *Computers and Typesetting, Vol. E: The Computer Modern Fonts*. Addison-Wesley, Reading, Mass, 1986.
- [6] D.E. Knuth. *Computers and Typesetting, Vol. A: The T<sub>E</sub>Xbook*. Addison-Wesley, Reading, Mass, 1989<sup>9</sup>.
- [7] J. Zlatuška. *Automatic generation of virtual fonts with accented letters for T<sub>E</sub>X*. Proceedings of the sixth european T<sub>E</sub>X conference, Paris, France, September 1991.

---

4. To make an encoding covering these languages was proposed by Janusz Bień at Gdańsk 1994 [1].



222	lira	256	circled R
223	recipe	257	macron
224	interrobang	260	degree sign
225	gnaborretni	261	plus-minus sign
226	dong sign	262	superscript 2
227	trademark	263	superscript 3
<hr/>		264	tick (ASCII-style acute)
Symbols from ISO-8859-1 (latin-1)		265	micro sign
<hr/>		266	pilcrow sign
242	cent	267	centered dot
243	sterling	271	superscript 1
244	currency sign	272	masculine ordinal indicator
245	yen	274	fraction one quarter
246	broken vertical bar	275	fraction one half
247	section sign	276	fraction three quarters
250	high dieresis	326	multiplication sign (times)
251	copyright	366	division sign
252	feminine ordinal indicator		
254	logical not		

## Appendix B: Font tables

### Font table of tcr1000

The layout of the text companion font tcr1000. The table shows the shapes implemented on June 30, 1995. Some more characters will be added to the first release.

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	˘	˙	ˆ	˜	¨	˝	°	˘	"0x
'01x	˘	-	·	˘	˘	˘			
'02x			"			—	—		"1x
'03x	←	→							
'04x	ˆ				\$			'	"2x
'05x			*					/	
'06x	o	1	2	3	4	5	6	7	"3x
'07x	8	9							
'10x									"4x
'11x						U			
'12x								Ω	"5x
'13x							↑	↓	
'14x	˘		*	o	†				"6x
'15x					⌘	∞	♯		
'16x									"7x
'17x							~	=	
'20x	˘	˘	"	"	†	‡		%	"8x
'21x	•		\$	¢	f	©			
'22x	Ⓔ		£						"9x
'23x									
'24x			¢	£	⊘	¥		§	"Ax
'25x	¨				⌞			—	
'26x		±			'	μ	¶	·	"Bx
'27x									
'32x							×		"Dx
'33x									
'36x							÷		"Fx
'37x									
	"8	"9	"A	"B	"C	"D	"E	"F	

**Font table of dcr1000**

This table shows the font dcr1000. The encoding is L<sup>A</sup>T<sub>E</sub>X<sub>2</sub> $\epsilon$ 's T1 encoding. The compound word mark in position '027 is an invisible character of zero width.

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	`	'	^	~	¨	ˆ	˚	ˇ	"0x
'01x	˘	-	·	˙	˚	,	‹	›	
'02x	“	”	„	«	»	–	—		"1x
'03x	o	l	J	ff	fi	fl	ffi	ffl	
'04x	˘	!	"	#	\$	%	&	'	"2x
'05x	(	)	*	+	,	-	.	/	
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	;	<	=	>	?	
'10x	@	A	B	C	D	E	F	G	"4x
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	"5x
'13x	X	Y	Z	[	\	]	^	_	
'14x	'	a	b	c	d	e	f	g	"6x
'15x	h	i	j	k	l	m	n	o	
'16x	p	q	r	s	t	u	v	w	"7x
'17x	x	y	z	{		}	~	-	
'20x	Ă	Ą	Ć	Č	Ď	Ě	Ę	Ĝ	"8x
'21x	Ł	Ł	Ł	Ń	Ń	Ń	Ń	Ŕ	
'22x	Ř	Ś	Š	Ş	Ť	Ť	Ů	Ů	"9x
'23x	Ÿ	Ž	Ž	Ž	IJ	İ	đ	§	
'24x	ă	ą	ć	č	ď	ě	ę	ğ	"Ax
'25x	ł	ł	ł	ń	ń	ŋ	ó	í	
'26x	ř	ś	š	ş	ť	ť	ů	ů	"Bx
'27x	ÿ	ž	ž	ž	ij	i	ı	£	
'30x	À	Á	Â	Ã	Ä	Å	Æ	Ç	"Cx
'31x	È	É	Ê	Ë	Ì	Í	Î	Ï	
'32x	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	Œ	"Dx
'33x	Ø	Ù	Ú	Û	Ü	Ý	Þ	ŠS	

'34x	à	á	â	ã	ä	å	æ	ç	"Ex
'35x	è	é	ê	ë	ì	í	î	ï	
'36x	ð	ñ	ò	ó	ô	õ	ö	œ	"Fx
'37x	ø	ù	ú	û	ü	ý	þ	`	
	"8	"9	"A	"B	"C	"D	"E	"F	

### Font table of fcr1000

The aFfrican Computer modern font fcr10. The fc encoding is now L<sup>A</sup>T<sub>E</sub>X<sub>2</sub> $\epsilon$ 's T4 encoding.

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	˘	˙	˚	˛	˜	˝	˚	˘	"0x
'01x	˘	˙	˚	˛	˜	˝	˚	˘	
'02x	“	”	„	«	»	–	—		"1x
'03x	o	l	J	ff	fi	fl	ffi	ffl	
'04x	□	!	"	#	\$	%	&	'	"2x
'05x	(	)	*	+	,	-	.	/	
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	;	<	=	>	?	
'10x	@	A	B	C	D	E	F	G	"4x
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	"5x
'13x	X	Y	Z	[	\	]	^	_	
'14x	'	a	b	c	d	e	f	g	"6x
'15x	h	i	j	k	l	m	n	o	
'16x	p	q	r	s	t	u	v	w	"7x
'17x	x	y	z	{		}	~	-	
'20x	Ɔ	Ɔ	Ɔ	Ɔ	Ɔ	Ɔ	Ɔ	Ɔ	"8x
'21x	K	N	Ɔ	Ñ	f	Ɔ	U	Y	
'22x	Č	P	Š	Ň	N	Š	Ɔ	T	"9x
'23x	Ě	Ě	T	Ɔ	Ɔ	Ɔ	đ	"	
'24x	Ɔ	Ɔ	e	ə	f	ě	Ɔ	h	"Ax
'25x	k	n	Ɔ	ñ	f	η	v	y	
'26x	č	Ɔ	š	ň	n	š	Ɔ	t	"Bx
'27x	ě	ę	Ɔ	t	"	i	ı	'	

'30x	ı	ı̇	Ě	Ā	Ḃ	Ō	Æ	Ç	"Cx
'31x	È	É	Ê	Ë	Ē	Ē	Ë	Ī	
'32x	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	Œ	"Dx
'33x	Ø	Œ	Œ	Œ	Œ	Œ	Œ	-	
'34x	ı	ı̇	Ě	Ā	Ḃ	Ō	Æ	Ç	"Ex
'35x	è	é	ê	ë	ē	ē	ë	ī	
'36x	đ	ñ	ò	ó	ô	õ	ö	œ	"Fx
'37x	ø	ø	ø	ø	ø	ø	ø	˘	
	"8	"9	"A	"B	"C	"D	"E	"F	

**Font table of wnrir10**

The Washington Romanised Indic font wnrir10. It is designed for typesetting sanskrit and other indic languages in scientific transscription. It is available from the CTAN archives in [tex-archive/fonts/wnri](http://tex-archive/fonts/wnri).

	'0	'1	'2	'3	'4	'5	'6	'7	"0x
'00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	
'01x	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl	"1x
'02x	ı	J	˘	˙	˚	˛	-	˚	
'03x	ı	ß	æ	œ	ø	Æ	Œ	Ø	"2x
'04x	-	!	"	#	\$	%	&	'	
'05x	(	)	*	+	,	-	.	/	"3x
'06x	0	1	2	3	4	5	6	7	
'07x	8	9	:	;	i	=	i	?	"4x
'10x	@	A	B	C	D	E	F	G	
'11x	H	I	J	K	L	M	N	O	"5x
'12x	P	Q	R	S	T	U	V	W	
'13x	X	Y	Z	[	"	]	^	.	"6x
'14x	‘	a	b	c	d	e	f	g	
'15x	h	i	j	k	l	m	n	o	"7x
'16x	p	q	r	s	t	u	v	w	
'17x	x	y	z	-	—	"	~	..	"8x
'20x	Ç	ü	é	ĸ	ä	à	ğ	ç	
'21x	ĥ	ë	è	ï	Ĭ	ì	À	Ĝ	"9x
'22x	É	æ	Æ	Ķ	ö	ò	Á	ù	
'23x	Ř	Ö	Ü	ē	Ē	ō	Ō	ř	

'24x	á	í	ó	ú	ñ	Ñ	ĩ	ṁ	"Ax
'25x	ă	ĩ	ũ	ž	Ž	ñ	ķ	Ḷ	
'26x	č	Č	š	ŗ	ṭ	ă	à	ĩ	"Bx
'27x	ì	Ḑ	Ḑ	Ṭ	Ḑ	ú	ù	ṁ	
'30x	ě	ö	γ	Ḥ	ḏ	ł	ř	ř	"Cx
'31x	Ÿ	z	z	Ḥ	Ḷ	ý	Ḷ	ř	
'32x	ã	ĩ	ũ	ẽ	õ	ě	ö	ł	"Dx
'33x	Ÿ	à	z	Š	ž	Ḑ	ḥ	Ḑ	
'34x	ā	ß	Ā	ī	Ī	ū	Ū	ŗ	"Ex
'35x	Ḷ	ř	Ḷ	ł	Ḷ	ĩ	Ḷ	ñ	
'36x	Ñ	ṭ	Ṭ	ḏ	Ḑ	ḥ	Ḷ	ś	"Fx
'37x	Ś	ş	Ş	√	ṁ	Ḑ	ḥ	Γ	
	"8	"9	"A	"B	"C	"D	"E	"F	

### Font table of wnpsr10

The Washington Puget Salish font wnpsr10. It is designed for the typesetting of american indian languages. The layout is rather chaotic, because it mimics the layout of some older font. It is available from the CTAN archives in [tex-archive/fonts/wnri](http://tex-archive/fonts/wnri).

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	"0x
'01x	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl	
'02x	ı	ı	`	'	˘	˘	-	°	"1x
'03x	ı	ß	æ	œ	ø	Æ	Œ	Ø	
'04x	-	!	"	#	\$	%	&	'	"2x
'05x	(	)	*	+	,	-	.	/	
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	;	i	=	i	?	
'10x	@	A	B	C	D	E	F	G	"4x
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	"5x
'13x	X	Y	Z	[	"	]	^	.	
'14x	'	a	b	c	d	e	f	g	"6x
'15x	h	i	j	k	l	m	n	o	
'16x	p	q	r	s	t	u	v	w	"7x
'17x	x	y	z	-	—	"	~	..	

'20x							λ	Λ	"8x
'21x	é	è	ε	é	è				
'22x		è	è	ì	ò	ù	è		"9x
'23x									
'24x		č	č	z	ə	j	k	l	"Ax
'25x	ł	ł	m̄	p̄	q̄	š	ł	w̄	
'26x	w̄	x̄	ȳ	ł	?	?	n̄	√	"Bx
'27x	x̄	·	ž	a	č	η	ł	k	
'30x		Č	Č	°	Θ	J	K	L	"Cx
'31x	Ł	Ł	M̄	P̄	Q̄	Š	T̄	W̄	
'32x	ɔ	X̄	Ȳ	Ł	?	?	N̄	ε	"Dx
'33x	X̄	ə	Ž	A	Č	IJ	Ł	K	
'34x	á	é	í	ó	ú	ó	à	è	"Ex
'35x	ì	ò	ù	è	ə	e	ì	ò	
'36x	u	ə	á	é	í	ó	ú	é	"Fx
'37x	Ā	Ē	Ī	Ō	Ū	Θ̄	?		
	"8	"9	"A	"B	"C	"D	"E	"F	



# A METAFONT–EPS interface

Bogusław Jackowski

BOP s.c.  
Gdańsk  
Poland  
ekotp@uni.v.gda.pl

*Do not explain too much.*  
W. Strunk Jr. and E.B. White,  
“The Elements of Style”

## 1 Introduction

$\text{T}\text{E}\text{X}$  is not a lion,  $\text{T}\text{E}\text{X}$  is an octopus. . . This sounds like a heresy, but it is my deepest conviction that one of the most wonderful features of the  $\text{T}\text{E}\text{X}$ /METAFONT system is its openness, i.e., the capability of collaboration with other systems. Hence the association with an octopus:



The paper illustrates this statement by presenting a brief description of an interface METAFONT-to-PostScript, MFtoEPS. The kernel of the package is a METAFONT

program (MFTOEPS.MF) which provides necessary definitions for translating the description of graphic objects from METAFONT to POSTSCRIPT. The POSTSCRIPT code is written to a log file. It can be extracted from the log file either manually or with a help of additional utilities. There are two programs in the package for performing this task: an AWK program and a T<sub>E</sub>X program, the latter a bit slower but more universal.

The POSTSCRIPT files (precisely, Encapsulated POSTSCRIPT files) produced by MFTOEPS are readable by some popular graphic programs, namely, by Adobe Illustrator (Macintosh and PC compatibles), CorelDRAW! (PC compatibles), and Fontographer (Macintosh and PC compatibles). In other words, graphic objects programmed using METAFONT can be further processed by these programs.

It should be stressed that not the idea of employing METAFONT to produce POSTSCRIPT code is important here. Much better tool for this purpose is J. D. Hobby's META-O T. This is a possibility of further processing of the objects generated by MFTOEPS which makes this package worthy of mention.

## 2 Overview of the MFtoEPS package

The MFTOEPS.MF program contains the definitions of the following macros which are meant to be used for generating EPS files:

```
eps_mode_setup    fix_line_width
write_preamble    fix_line_join
write_postamble   fix_line_cap
find_BB           fix_miter_limit
set_BB            fix_dash
fill_C            fix_fill_cmyk
draw_C            fix_draw_cmyk
clip_C
```

Obviously, not all possibilities of POSTSCRIPT are exploited, but the main idea was to provide a *simple tool* for producing output "eatable" by programs which are not POSTSCRIPT interpreters. Therefore only a small subset of the POSTSCRIPT language can be taken into account. Nevertheless, these 15 commands are enough to produce innumerable variety of graphic objects.

METAFONT programs using MFTOEPS have the following structure:

```
1  input mftoeps;
2  \input eps_mode_setup; % instead of mode_setup
3  < METAFONT code >
4  find_BB <list of paths>;
5  write_preamble jobname;
6  < METAFONT code containing fill_C, draw_C, clip_C, etc.>
7  write_postamble;
8  end.
```

The structure seems straightforward, except for some notational details which will be explained momentarily. Perhaps only the fourth line needs a few remarks. A properly formed EPS file should contain the coordinates of the corners of the bounding box in a comment line at the beginning of the file. Macro `write_preamble` needs to know the respective coordinates, as it is responsible for generating the header of an EPS file. Macro `find_BB` simply prepares the data for `write_preamble`.

As you can see, using the plain `beginchar` and `endchar` commands is not essential, although usually it is convenient to make use of them.

### Synopsis of the interface of the MFtoEPS package

Conventions: In the following I shall use words *number*, *pair*, *string*, and *path* as an abbreviation for *numeric expression*, *pair expression*, *string expression*, and *path expression*, respectively. The angle brackets, `<` and `>`, used for marking parameters of macros, are “meta-characters,” i.e., they do not belong to the METAFONT code.

COMMAND:

`eps_mode_setup`

USAGE:

`eps_mode_setup <an optional number (0 or 1)>;`

REMARKS:

This command should be used *instead* of the usual `mode_setup` command. The forms `eps_mode_setup` and `eps_mode_setup 1` are equivalent. One of them (preferably the former one) should be used for normal processing, i.e., for generating EPS files. Invoking `eps_mode_setup 0` is meant primarily for testing purposes and is supposed to be used by experienced programmers who know what they are doing.

COMMAND:

`write_preamble`

USAGE:

`write_preamble <string>;`

REMARKS:

This command initializes the process of writing of the POSTSCRIPT code. The string expression is the name (without extension) of the resulting EPS file; the extension is always EPS. METAFONT is switched to the `batchmode` in order to avoid slowing down the process by writing `mess(ages)` to the terminal. The inspection of a log file is thus highly recommended.

## COMMAND:

```
write_postamble
```

## USAGE:

```
write_postamble;
```

## REMARKS:

This command ends the writing of the PS code, switches METAFONT back to the `errorstopmode`, and performs necessary “last minute” actions (see below).

## COMMANDS:

```
set_BB find_BB reset_BB
```

## USAGE:

```
set_BB <four numbers or two pairs separated by commas>;
```

```
find_BB <a list of paths separated by commas>;
```

```
reset_BB;
```

## REMARKS:

Commands `set_BB` or `find_BB` should be invoked prior to invoking `write_preamble`. `set_BB` sets the coordinates of the corners of the bounding box of a graphic object; it is useful when the bounding box of a graphic object is known in advance or if it is required to force an artificial bounding box. `find_BB` computes the respective bounding box for a list of paths; if several `find_BB` statements are used, the common bounding box is calculated for all paths that appeared in the arguments. The result is stored in the variables `x1_crd`, `y1_crd`, `xh_crd`, and `yh_crd`. There are two functions, `llxy` and `urxy`, returning pairs `(x1_crd,y1_crd)` and `(xh_crd,yh_crd)`, respectively. The last command, `reset_BB`, makes `x1_crd`, `y1_crd`, `xh_crd`, and `yh_crd` undefined (the initial situation); `reset_BB` is performed by the `write_postamble` macro, which is convenient in the case of generating several EPS files in a single METAFONT run.

## COMMANDS:

```
fill_C draw_C
```

## USAGE:

```
fill_C <a list of paths separated by commas>;
```

```
draw_C <a list of paths separated by commas>;
```

## REMARKS:

These commands are to be used instead of the usual METAFONT `fill` and `draw` ones. They cause that a list of paths followed by the POSTSCRIPT operation `eofill` (`fill_C`) or `stroke` (`draw_C`) is translated to a POSTSCRIPT code. The list of paths constitutes a single curve in the sense of POSTSCRIPT.

## COMMAND:

`clip_C`

## USAGE:

`clip_C` <a list of paths separated by commas, possibly empty>;

## REMARKS:

The macro `clip_C` with a non-empty parameter works similarly to the `fill_C` command, except that the `eofclip` operator is issued instead of `eofill`. This causes an appropriate change of the current clipping area. According to POSTSCRIPT's principles, the resulting area is a set product of the current clipping area and the area specified in the argument of the `eofclip` command. The empty parameter marks the end of the scope of the most recent `clip_C` command with a non-empty parameter. In other words, nested `clip_C` commands form a "stack" structure. If needed, the appropriate number of parameterless `clip_C` commands is issued by the `write_postamble` macro, thus the user needs not to care about it. *WARNING: files produced using clip\_C are interpreted properly by Adobe Illustrator (provided paths directions are defined properly) but not by CorelDRAW! (ver. 3.0).*

## COMMANDS:

`fix_line_width` `fix_line_join`

`fix_line_cap` `fix_miter_limit`

`fix_dash`

## USAGE:

`fix_line_width` <a non-negative number (dimension)>;

`fix_line_join` <a number (0, 1 or 2)>;

`fix_line_cap` <a number (0, 1 or 2)>;

`fix_miter_limit` <a number  $\geq 1$  (dimension)>;

`fix_dash` (<a list of numbers (dimensions) separated by commas, possibly empty>) <a number (dimension)>

## REMARKS:

These command are to be used in connection with the `draw_C` command. The command `fix_line_width` fixes the thickness of the outline. The other four commands correspond to POSTSCRIPT operations `setlinejoin`, `setlinecap`, `setmiterlimit`, and `setdash` (see "POSTSCRIPT Language Reference Manual" for details). All commands should be used after `write_preamble`, as `write_preamble` sets the default thickness (0.4 pt), default line join (1), default line cap (1), default miter limit (10 bp), and a solid line as a default for stroking (`fix_dash` () 0).

## COMMANDS:

```
fix_fill_cmyk fix_draw_cmyk
```

## USAGE:

```
fix_fill_cmyk <four numbers separated by commas>;
```

```
fix_draw_cmyk <four numbers separated by commas>;
```

## REMARKS:

These commands define the colors of the interiors of graphic objects (`fix_fill_cmyk`) and colors of outlines (`fix_draw_cmyk`) using cyan-magenta-yellow-black model (the basic model of the MFTOEPS package). They should be used after `write_preamble` (because `write_preamble` defines the black color as a default for both macros) and prior to invoking the corresponding `fill_C` and `draw_C` commands. There are also (just in case) macros `fix_fill_rgb` and `fix_draw_rgb` using red-green-blue model; the argument to both macros is a triple of numbers. (The user can control the process of conversion from RGB to CMYK by the redefinition of macros `under_color_removal` and `black_generation`.) The numbers forming the arguments of the macros are supposed to belong to the interval [0..1].

Besides the fifteen basic macros there are two functions and two control variables that may be of some interest for a virtual user of the MFTOEPS package:

## ADDITIONAL FUNCTIONS:

```
pos_turn neg_turn
```

## USAGE:

```
pos_turn (<path>)
```

```
neg_turn (<path>)
```

## REMARKS:

Each function returns the path passed as the argument, except that the orientation of the path is changed, if necessary: `pos_turn` returns paths oriented anti-clockwise, `neg_turn`—oriented clockwise. This may be useful for creating pictures which are to be processed further by Adobe Illustrator, because this program is sensitive to the orientation of paths.

## CONTROL VARIABLE:

```
yeseeps
```

## REMARKS:

No EPS file will be generated unless the variable `yeseeps` is assigned a definite value. It is advisable to set this variable in a command line (see section “Examples”).

CONTROL VARIABLE:

testing

REMARKS:

If the variable `testing` is assigned a definite value, the whole POSTSCRIPT code is flushed to the terminal, thus slowing down significantly the process of generation of an EPS file (cf. the description of the `write_preamble` command).

### 3 Examples

All sample programs in this section are presented *in extenso*. The reader is not supposed to study the code thoroughly. Nevertheless, I prefer to leave the reader to decide which parts of the code are to be skipped.

Let us start with a trivial example of a “pure” METAFONT program:

```

1 beginchar(48, % ASCII code
2   2cm#, % width
3   1cm#, % height
4   0cm# % depth
5 );
6 fill unitsquare xscaled w yscaled h;
7 endchar;
8 end.
```

The program, obviously, generates a font containing one character: a darkened rectangle 2 cm × 1 cm. In order to generate an EPS file containing the same figure, a few modifications are necessary:

```

1 input mftoeps;
2 eps_mode_setup;
3 beginchar(48, % just something
4   2cm#, % width
5   1cm#, % height
6   0cm# % depth
7 );
8 set_BB 0,-d,w,h; % coordinates
9               % of the corners
10              % of the bounding box
11 write_preamble "rectan";
12 fill_C unitsquare xscaled w yscaled h;
13 write_postamble;
14 endchar;
15 end.
```

Four new commands appeared: `eps_mode_setup`, `set_BB`, `write_preamble`, and `write_postamble`; moreover, `fill` has been replaced by `fill_C`. This is a usual routine for converting an “ordinary” METAFONT program to a form suitable for generating EPS files. Obviously, `draw` should be replaced by `draw_C`, and `filldraw`—with two operations `fill_C` and `draw_C`. In the latter case the order of operations `fill_C` and `draw_C` is significant if the drawing and filling colors are different.

Having done this changes you can easily generate the respective EPS file, provided you are a DOS user. Assume that the modified program is stored in the file `RECTAN.MF`. In the package `MFTOEPS` you will find a DOS batch, `M2E.BAT` (subdirectory `PROGS`), which—perhaps after slight adjustments—can be used for this task. It is enough to write

```
m2e rectan
(no extension, please) from the command line in order to obtain the required
RECTAN.EPS file. The batch makes use of AWK for extracting the POSTSCRIPT code
from the log file. There is also an alternative batch, M2E-ALT.BAT, that employs
 $\TeX$  for this purpose. In both batches METAFONT is called in the following way:
mf386 &plain \yesepts:=1; input %1
```

Observe the assignment `yesepts:=1`. In fact, assigning a definite (arbitrary) value to the `yesepts` variable triggers the action of the generation of an EPS file.

I hope that making scripts for other operating systems should not be extremely difficult. I would be very much obliged if others could contribute such scripts to the package.

Let us consider now a more complex example. Suppose that the file `POLYGON.MF` contains the following definitions:

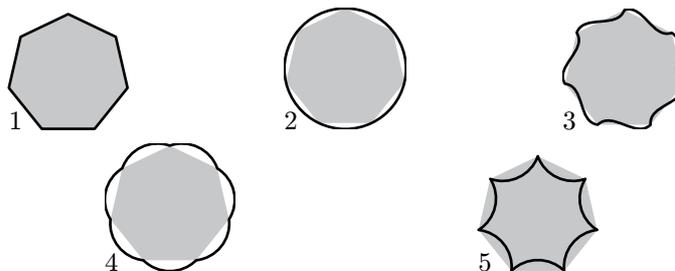
```
1  vardef regular_polygon(expr n) =
2  % n is the number of vertices;
3  % the diameter of the circumscribed
4  % circle is equal to 1, its centre is in the origin
5  (up % first vertex
6  for i:=1 upto n-1:
7  -- % next vertices:
8  (up rotated (i*(360/n))) endfor
9  -- cycle) scaled .5
10 enddef;
11 vardef flex_polygon(expr n,a,b) =
12 % n is the number of vertices,
13 % a, b are the angles (at vertices)
14 % between a tangent to a ‘flex side’
15 % and the corresponding secant
16 save zz;
17 pair zz[ ]; % array of vertices
18 for i:=0 upto n-1:
```

```

19  zz[i]:=up rotated (i*(360/n));
20  endfor
21  (zz[0] {(zz[1]-zz[0]) rotated a}
22   for i:=1 upto n-1:
23     .. {(zz[i]-zz[i-1]) rotated b}
24     zz[i]
25     {(zz[(i+1) mod n]-zz[i]) rotated a}
26   endfor
27   .. {(zz[0]-zz[n-1]) rotated b} cycle)
28  scaled .5
29  enddef;

```

The first function, `regular_polygon`, returns a closed path being—as the name suggest—a regular polygon with a given number of vertices. The second function, `flex_polygon`, returns a curve being in a sense a “generalised polygon”—the following examples show why this epithet is adequate:



The first picture was generated by the following program:

```

1  input polygons;
2  input mftoeps;
3  eps_mode_setup;
4  beginchar(0,16mm#,16mm#,0);
5  path P[ ]; % ‘‘room’’ for two polygons
6  % preparing:
7  P[1]:=regular_polygon(7)
8   scaled w shifted (.5w,.5h);
9  P[2]:=flex_polygon(7,0,0)
10 scaled w shifted (.5w,.5h);
11 % exporting:
12 find_BB P[1], P[2];
13 write_preamble jobname;
14 % 25 percent of black for filling:
15 fix_fill_cmyk 0,0,0,.25;
16 fix_line_width 1pt;

```

```

17 fill_C P1; draw_C P2;
18 write_postamble;
19 endchar;
20 end.

```

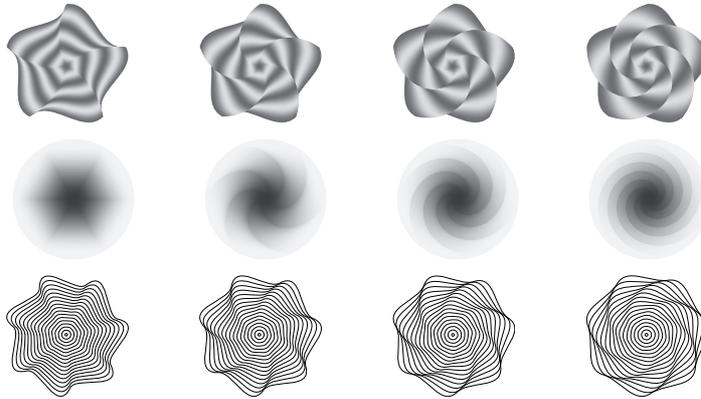
The remaining four figures can be obtained by a simple modification of the line 9 of the program:

```

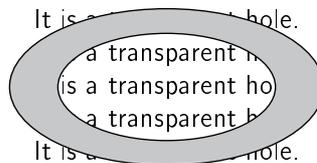
P[2]:=flex_polygon(7,-180/7,180/7) % 2
P[2]:=flex_polygon(7,45,45) % 3
P[2]:=flex_polygon(7,-45,45) % 4
P[2]:=flex_polygon(7,45,-45) % 5

```

These fairly trivial objects can be used for achieving not so much trivial effects (METAFONT sources are included in the MFTOEPS package):



So far the examples have contained `fill_C` and `draw_C` commands with arguments being single paths. POSTSCRIPT, contrary to METAFONT, accepts groups of paths as a single curve. Therefore the `fill_C` and `draw_C` commands were defined to accept the lists of METAFONT paths as arguments. In the resulting POSTSCRIPT code they constitute a single object. The main reason is that such objects may contain transparent holes. This enables achieving such effects as:



The graphic object was generated by the following simple program:

```

1 input mftoeps; eps_mode_setup;
2 w#=4cm#; h#=2cm#; define_pixels(w,h);
3 set_BB origin, (w,h);

```

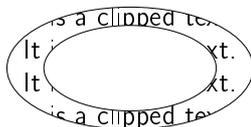
```

4  write_preamble jobname;
5  % 25 percent of black for filling:
6  fix_fill_cmyk 0,0,0,.25;
7  fix_line_width 1pt;
8  for oper:="draw_C", "fill_C":
9  scantokens oper
10 % outer edge:
11  fullcircle
12   xscaled w yscaled h
13   shifted (.5w,.5h),
14 % inner edge:
15  reverse fullcircle
16  xscaled .7w yscaled .7h
17  shifted (.5w,.5h);
18 endfor
19 write_postamble;
20 end.

```

One innocent trick was used in order to shorten the code: the loop in the combination with the `scantokens` command (lines 8 and 9). It is advisable to have paths that form transparent holes appropriately oriented—therefore the operator `reverse` is used line 15. A  $\TeX$  code for obtaining the above figure is obvious: it is enough to put the picture on the top of a text box, using, e.g., the `\llap` command.

Removing the command `fix_fill_cmyk` (line 6) and replacing the command `fill_C` (line 8) by `clip_C` gives the opportunity of obtaining yet another effect:



In this case, however, the  $\TeX$  code is somewhat complicated, since macros for inclusion of an EPS file (I use Tomas Rokicki's `EPSF.TEX`) embed the code of the EPS file into a `POSTSCRIPT save – restore` group. A clipping path is subjected to such a grouping, contrary to the state of the currently painted picture. Therefore some `\special` hackery is needed (the respective  $\TeX$  source is included with samples in the `MFTOEPS` package).

The difference between single and multiple paths in the context of drawing outlines (`draw_C`) is meaningless.

The final example shows how to use clipping for generating a geometric figure known as “Sierpiński’s carpet.” In order to construct the “carpet” you start with a square with a central hole being a square thrice smaller. Now you divide the figure into nine squares

and replace all filled small squares with a scaled down thrice the original square. Then you apply the same procedure to the smaller squares, an so on, *ad infinitum*.

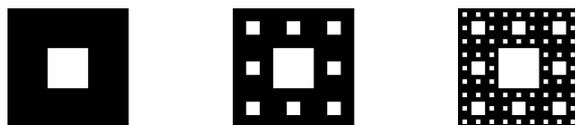
Here you have the program accomplishing this task (infinity “equals” three):

```

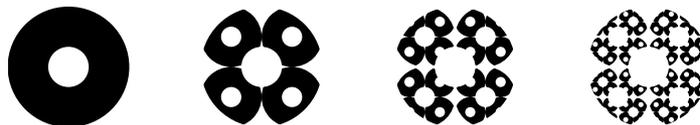
1  input mftoeps; eps_mode_setup;
2  % ---
3  def ^ = ** enddef; % syntactic sugar
4  primarydef i // n = % ditto
5    (if n=0: 0 else: i/n fi)
6  % why not to divide by 0?
7  enddef;
8  def shifted_accordingly(expr i,j,n,D)=
9    shifted ((i//n)[0,w-D],(j//n)[0,w-D])
10 enddef;
11 % ---
12 w#=16mm#; h#=16mm#; define_pixels(w,h);
13 for N:=1,2,3: % 4, 5, 6, ..., infinity
14   set_BB 0,0,w,h;
15   write_preamble jobname & decimal(N);
16   D:=3w;
17   for n:=
18     0 for q:=1 upto N-1: , 3^q-1 endfor:
19   % i.e.:
20   % ‘‘for n:=0, 3^1-1, ..., 3^(N-1)-1:’’
21   path p[], q[]; D:=1/3D; k:=-1;
22   for i:=0 upto n: for j:=0 upto n:
23     k:=k+1;
24     p[k]=unitsquare scaled D
25     shifted_accordingly(i,j,n,D);
26     q[k]=reverse unitsquare scaled 1/3D
27     shifted (1/3D,1/3D)
28     shifted_accordingly(i,j,n,D);
29   endfor; endfor;
30   clip_C p0, q0
31   for i:=1 upto k: , p[i], q[i] endfor;
32   endfor;
33   fill_C unitsquare scaled w;
34   write_postamble;
35   endfor;
36 % ---
37 end.

```

The program is lengthy mainly because of technical details that are not especially interesting, however, there are three points worthy of comment. First, observe that a couple of EPS files is produced in one METAFONT run (the loop in line 13 is relevant here); second, loops are used for forming arguments to the loop in line 18 and to the `clip_C` command in line 31—it is a very useful feature of METAFONT that loops behave exactly like macros; and third, observe that only once the operation `fill_C` is used. The resulting EPS files are shown in the following picture:



You may argue that such a figure can be generated easily without clipping. True, yet I like this approach—can you imagine a simple method for generating a “circular carpet”



without clipping? But, on the other hand, finding the precise bounding box for a clipped figure becomes a non-trivial task. You must remember, moreover, that clipping consumes a lot of the resources of a POSTSCRIPT interpreter, thus it should be used with a great care.

#### 4 Final remarks

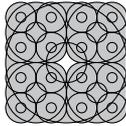
The MFtoEPS package was not devised as a competitive software for such giants like Adobe Illustrator or CorelDRAW!. On the contrary, it can be regarded as their little ally. Interactive programs cope not so well with tasks that bear logical structure. In such cases METAFONT—with its wealth of *programmable* path operations, absent “by definition” from the menus of interactive programs—is certainly a preferable tool.

One of the advantages of the applied approach is its portability—the only software needed is METAFONT and either AWK or T<sub>E</sub>X. Another advantage is its flexibility. It is not particularly difficult to modify the MFtoEPS package to produce another POSTSCRIPT dialect, if for some reason the dialect of Adobe Illustrator is inconvenient. MFtoEPS can also be modified to produce output in other lingos, e.g., HP-GL (Hewlett-Packard Graphic Language).

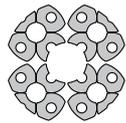
There is still a lot of work to be done. Of course, every program can be improved, but perhaps more important would be preparing a library of METAFONT routines useful for creating objects with a vector representation.

For example, it would be convenient to have a procedure which for a given set of graphic objects finds a single curve (outline) filling of which would give the same optical result. In other words, such a procedure would perform the task of finding an outline for a set union of graphic objects. Such a procedure is known as *removing overlaps*. The example of the “circular carpet” (see above) illustrates a similar problem: to find an outline for a set intersection of a group of graphic objects.

If the carpet is generated using clipping, the POSTSCRIPT file contains, in fact, the following elements:



They are partially invisible because of clipping, still they are there. In some contexts, e.g., if the figure is to be cut on a cutting plotter, it is crucial to replace such a multiplicity of objects by a single object:



Note that routines for finding the outline of a set union or a set intersection of a group of graphic objects are not MFTOEPS-oriented. I guess that METAFONT programmers would appreciate having it as well META O T programmers. Universal routines of that kind are important from the point of view of the openness of the T<sub>E</sub>X/METAFONT system, and the openness—as was already mentioned—is one of the most powerful features of the system.

Note also that the openness of a system concerns both *output* and *input*. MFTOEPS accomplishes the first part of the conjunction, but one can think also about an import from POSTSCRIPT to METAFONT. A “prototype” of such a package is under testing. Its kernel is the converter (written in POSTSCRIPT and using the GHOSTSCRIPT interpreter of POSTSCRIPT) of a general POSTSCRIPT code into a canonical Encapsulated POSTSCRIPT form; the result of such a conversion can be translated to a METAFONT program using, e.g., AWK. This would complete a link between METAFONT and POSTSCRIPT. I do believe that providing such links is one of the most efficient ways towards a limitless development of the T<sub>E</sub>X/METAFONT system.

## 5 Glossary

**AWK** a simple yet powerful batch text processor.

**Bounding box** the smallest rectangle surrounding the glyph of a picture; coordinates of its lower left and upper right corners (in big points) should appear in a structural comment in a header of an EPS file.

**EPS file** Encapsulated POSTSCRIPT file; a single-page POSTSCRIPT document; the purpose of the EPS file is to be included (“encapsulated”) as a part of other POSTSCRIPT programs and to exchange graphic data among applications.

**Even-odd rule** a rule that specifies the interior of a (multiple) path in the following way: if for a given point and for any ray drawn from this point to infinity the number of intersection points of the ray and the path is odd, the point is inside; if the number is even, the point is outside; command `eofill` and `eoclip` operators follow this rule.

**Path orientation** nodes of a closed single path are ordered; if traversing a path following the order of its nodes results in an anti-clockwise turn(s), the path is positively oriented, if it results in a clockwise turn(s), its orientation is negative; number of turns (signed) is called a turning number (METAFONT) or a winding number (POSTSCRIPT); the operators `fill` and `clip` make use of a winding number, the operators `eofill` and `eoclip` ignore it.

**TEX** is a trademark of the American Mathematical Society.

**METAFONT** is a trademark of Addison Wesley Publishing Company.

**POSTSCRIPT** is a registered trademark of Adobe Systems Incorporated.

**Fontographer** is a registered trademark of Altsys Corporation.

**Adobe Illustrator** is a trademark of Adobe Systems Incorporated.

**CorelDRAW!** is a registered trademark of Corel Corporation.

**GHOSTSCRIPT** is a copyrighted product of Aladdin Enterprises.

## 6 Availability

The MFtoEPS package can be found at `ftp.pg.gda.pl`  
in the directory `TeX/GUST/contrib/BachoTeX95/B_Jackowski`

## References

- [1] Adobe Systems Inc. *POSTSCRIPT Language Reference Manual*. Addison-Wesley, 1991.
- [2] A.V. Aho, B.W. Kernighan, and P.J. Weinberger. *The AWK Programming Language*. Addison-Wesley, 1988.
- [3] B. Jackowski and M. Ryko. *Labyrinth of METAFONT paths in outline*. conference proceedings, EuroTEX'94, Sobieszewo, 1994.
- [4] D.E. Knuth. *The METAFONTbook*. Addison-Wesley, 1992.



# Use of T<sub>E</sub>X as database with AnyT<sub>E</sub>X

Kees van der Laan

Hunzeweg 57  
9893 PB Garnwerd  
The Netherlands  
cgl@rc.service.rug.nl

## Abstract

The use of BLUe's format databases has been treated. A new issue is introduced since the emerge of BLUe's Format system this spring. Boolean tags can be added to for example address.dat entries to denote fields and their contents. Together with `\search` one can easily obtain the list of names – and via these names the full entries, i.e., the addresses – of those who have not paid their membership fee, for example.

## 1 Introduction

Why couple the buzzword database to T<sub>E</sub>X? What has T<sub>E</sub>X got to do with it? Vaguely the answer is that we like to store *collections* of the right granularity, such as addresses, references, copy parts, tools and formats outside of T<sub>E</sub>X – or one of its flavours – and only borrow what we need. You don't have to pay for what you don't use. Important is also the data integrity aspects which can be achieved via databases. We only store the information once for use in different contexts. At the heart is the process of selective loading. The benefits are that next to your stable T<sub>E</sub>X formatting system, you have also a stable T<sub>E</sub>X database tool,<sup>1</sup> which can be adjusted to your applications. And because it is written in plain it can be used with AnyT<sub>E</sub>X.

But, just embracing the database approach is not enough. It also important to hide T<sub>E</sub>Xnical details of storing and have access to material in a transparent way. Let me show you by example what can be achieved with databases as a T<sub>E</sub>Xnical tool.

---

1. And as a consequence stable data files. No conversions!

## 2 Formats

I consider it very convenient, when I need a format, that I just can say `\<formatname>` without to worry where the formats are stored, especially when I have to work on several different computers.

This is trivial as such except when we wish to have variant formats stored separately.

The effect of the above command is that the database, which name you can forget about, will be searched and the requested format will be selectively loaded.

## 3 Tools

After having made the decision which tools should be made available by default – as part of the kernel – and which on request – as part of a module – we have the problem how to provide the commands such that the user does not have to worry about T<sub>E</sub>Xnical details. The tools must be transparently available. Examples are:

```
\beginbintree{<no>}
  {<contents>}...{<no>}{<contents>}{<order>}
\endbintree
\hanoi3
\beginpascal... \endpascal
\begincrosswords... \endcrosswords
\beginbridge... \endbridge
\loadntglogo\copy\ntglogobox
etc.
```

For all of the above cases the user does not know whether the environment is already available or whether it will be loaded first. IMHO, the user should not be aware of it, normally.

## 4 Pictures

I use pictures generally twice: in the article and for a transparency. Of late I also adopted the possibility to vary the visibility. For example the user can ask for a full-blown picture or for the simple variant.

The idea is that we load all the pictures we will need by name, via

```
\pictures{\<name1>... \<namen>}
```

The picture can be used, at best within math display, via the invocation

```
\<namex>
```

preceded by for example `\thispicture{\fulltrue}`, or with an adjustment of the default scaling.

## 5 References

With references we have to deal with the cross-referencing aspects, and that the list of references is usually typeset at the end. To meet this requirements I chose to *specify* the references at the beginning of the script, via the `references` command, in the order you wish.

```
\references{\langle name1 \rangle ... \langle namen \rangle}
```

The effect of this command is that the specified references will be loaded and set in a box to be pasted up later. In order to allow cross-referencing the names are redefined with the (implicit) sequential number in the specified list as replacement text. A reference to an entry in the list of references can be simply done via

```
\langle namex \rangle
```

The list of references can be put at the appropriate place via `\pasteupreferences`, preceded by for example `\thisreferences{...}`.

## 6 Addresses

Usually addresses are merged with letters (and a background) to format letters. First we have to activate the `\letter` format. Similarly as with references and pictures you are requested to specify the address(es) and the addressee(s), next to other issues like `\subject` and `\ourreference`, `\yourreference`.

```
\input blue.tex \letter           % Preliminary initializations
\subject{\TeX{ } for BLU}
\ourreference{22 1 95}
\yourreference{\TB}
\addresses{\knuthde}
\addressee{\knuthde}              % Initializes \addressee name
                                   % \affiliationbox
                                   % \thisscript{\notlastscript}
\beginscript                       % \beginletter is an alias
\dear
First of all ...
...
\sincerely
%backmatter such as \ps, \cc
\endscript
```

It also possible to send out a letter to a bunch of people from the database, for example to all in the database.<sup>2</sup>

```
\input blue.tex \letter
\subject{\TeX{ } for BLU}
```

---

2. The letter proper is assumed to be available in `letter.tex`.

```

\yourreference{\TB}
\ourreference{1 2 95}
\lettertoall
\bye

```

Apart from the above it is also desirable to manipulate addresses for address labels.

**Example:** Search for addresses from RUSSIA in the database address.dat

In order to justify my use of the word database in relation to T<sub>E</sub>X, I provided the search macro to browse the databases lit.dat and address.dat. Below the input has been given of the search of address.dat for the pattern RUSSIA. The result is contained in the toks variable \name1st. The names are also written to the log file to give you a check for whether the right names have been selected.

```

\input blue.tex
\searchfile{address}
\search{RUSSIA}
\bye

```

For the detailed use see the chapter Formats in the ‘Publishing with T<sub>E</sub>X,’ user’s guide.

## 7 And what about fields?

As can be seen from the conventions for the database entries I did not elaborate much on the database fields. However, there are a few nice exceptions. First, I allow for the markup tag \annotation with an argument. This gives us the possibility to add annotations to the literature entries. While typesetting we can control the layout via an appropriate definition of \annotation. Most of the time I use the empty definition, and occasionally the identity, i.e., the argument as such is typeset.

Another nice extension is to allow for logicals, for example \ifregisteredblue, with the functionality of a binary field. These ‘fields’ can be inserted in an address entry if you like. For ordinary use they don’t hinder. However, with the command \search{\registeredbluetrue} all the names of entries with the field on will be selected, i.e., all the registered users of BLUE’s Format system. For NTG this can be applied to monitoring the status of the payment of the subscription fee, if the treasurer thinks of letting T<sub>E</sub>X take care of the membership database.

## 8 What more?

As usual goodies like those mentioned above have their price. To extend a database needs some attention. First, because of the conventions adopted, and second, we have to do some more, also add storage allocations and the user interface commands. The addition of references, pictures and addresses is simple, once the convention for the

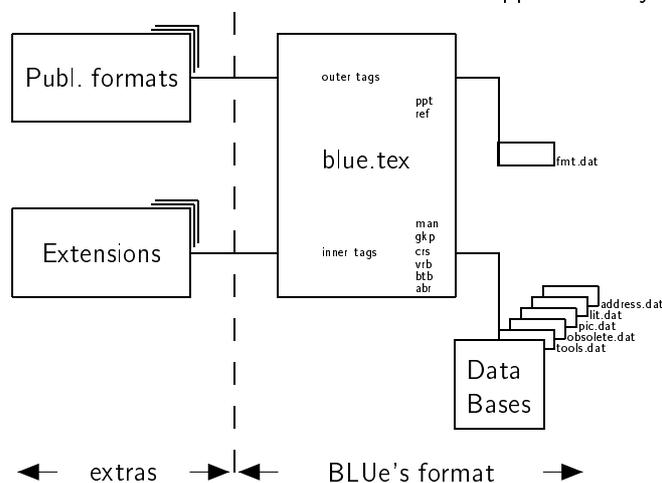
entries has been obeyed. For the details I refer to the appropriate places in PWT. It is advised to verify the database for its integrity via for example making a list of entries.<sup>3</sup>

The ‘Publishing with  $\TeX$ ,’ user’s guide which comes with BLUE’s Format system has been made available on the CTAN in directory `/pub/archive/info/pwt/<filename>`

Trivial details which are entailed by the personal format idea have been omitted as a consequence. Remember BLUE’s Format system is a personalized format. It knows about you!

### 8.1 $\TeX$ nical details

The deeper aspects – like selective loading – have been treated in ‘BLUE’s format databases.’ All the user aspects have been addressed in the earlier mentioned PWT guide. The picture which summarizes the model has been appended for your convenience.



### 8.2 Future work?

Perhaps, I should supply a selection macro which allows multiple queries.

3. The commands are `\contentstoolsorfmt`, or `\contentsdatabase{<database>}`, with for database address, lit, or pic.



# Indexing in T<sub>E</sub>X with AnyT<sub>E</sub>X

Kees van der Laan

Hunzeweg 57  
9893 PB Garnwerd  
The Netherlands  
cgl@rc.service.rug.nl

## Abstract

The creation of a modest index within a one-pass T<sub>E</sub>X job has been treated. In general a proof run and a final run are needed.

## 1 Introduction

Making an index is an art. The fundamental problem is *What to include in an index?*

Computer-assisted indexing is not simple either. Issues are

- the markup of keywords or phrases
- to associate page numbers
- to sort and compress raw Index Reminders (IRs), and
- to typeset the result.

My approach is to create proof indexes – also called mini-indexes – for each chapter and learn from those what should be included in the total index. I perceived this as very pleasant in practice. Even if you prefer `\makeindex` for the real index, this processing on the fly of a chapter index can be of great help.<sup>1</sup>

This paper is essentially a chapter from the user's guide 'Publishing with T<sub>E</sub>X,' which comes with BLUe's Format system.

## 2 Use

I'll show how to mark up Knuth's four types of IRs, how to mark up accents, how to mark up font switching, and how to mark up spaces as part of the IR.

---

1. It is said that the automatic generation of an index is a feature of the Literate Programming tools. For LP with T<sub>E</sub>X as such, as for example Gurari's ProT<sub>E</sub>X, this on-the-fly indexing within T<sub>E</sub>X can be used.

**Example** Markup, commands and resulting index

The right column has been obtained via

- `\loadindexmacros`, at the beginning of the script
- `\sortindex`, at the place of indexing, and
- `\pasteupindex`, for the pasteup of the index.

```
Types of IR
0  ^{return}
1  ^|verbatim|
2  ^|\controlsequence|
3  ^\langle syntactic quantity\rangle
Accents ^{\'e!}\'eve!},
font changing ^{\bf bold}
and spaces ^{control\ symbol}
Control sequences
  ^{\TeX, and \AmSTeX}
  ^{Lamport and \LaTeX}
brackets ^{\tt< \rm and \tt>}
\newpage ^{return}
\newpage ^{return}%on purpose
\sortindex\pasteupindex\bye
```

```
< and > 1
bold 1
control symbol 1
\controlsequence 1
é!è! 1
Lamport and LATEX 1
TEX, and AMS-TEX 1
return 1–3
⟨syntactic quantity⟩ 1
verbatim 1
```

The representation of page numbers as a range comes out automatically.

**Question** What makes a good index? Of course this is a million-dollar question. Let us concentrate on the number of entries and on the number of page numbers per entry. Which of the two extremes sketched below is the better one in your opinion? One with many entries pointing to issues spread throughout the book – like *The T<sub>E</sub>Xbook* ;-))), and pushing the limits just for the imagination, an index with pointers to related work on the internet, accessible by just clicking the mouse – or one with few page numbers per entry?<sup>2</sup>

**Answer** As usual it all depends on your application. End of answer. But – there is always a but – the complaint I heard most about *The T<sub>E</sub>Xbook* was that the information is spread all over, and that it is hard to find what you are looking for. Therefore I consider a few page numbers per entry beneficial. (Let us forget about the intrinsic complexity of the subject, certainly at the time.) BLUE's format supports scrutinizing parts of an index, because it is so easy to generate an index per chapter on the fly. It is hardly not more difficult than generating a table of contents. An index per chapter can be scrutinized more easily, and redundancies removed. That the index provides a mechanism to link things over chapters is a good thing, however. Don't misunderstand me. But don't overuse it, IMHO, with all respect. Remember DeVinne's adage 'The last thing to learn is simplicity.'

2. Courtesy Erik Frambach.

### 3 Markup of Index Reminders

IR-s are at the heart of the process. Knuth distinguished 4 types to facilitate the outside processing. I'll adopt his IRs syntax and types.

#### 3.1 Syntax

Knuth's IRs obey the following syntax. IR, syntax

$$\langle \text{word(s)} \rangle_{\square} ! \langle \text{digit} \rangle_{\square} \langle \text{page number} \rangle.$$

The digits 0, 1, 2, or 3 denote the types: words, verbatim words, control sequences, and syntactic quantities. A user does not have to bother about the digits nor about the page numbers. Knuth has adopted the accompanying conventions for the word(s) of IRs.<sup>3</sup>

Mark up	Typeset in copy*	IR
$\wedge\{\dots\}$	...	... !0 $\langle \text{page no} \rangle.$
$\wedge \dots $	\dots	... !1 $\langle \text{page no} \rangle.$
$\wedge \backslash\dots $	\backslash\dots	... !2 $\langle \text{page no} \rangle.$
$\wedge\langle \dots \rangle$	$\langle \dots \rangle^{**}$	... !3 $\langle \text{page no} \rangle.$

\* |...| denotes manmac's, TUGboat's, ... verbatim  
 \*\* in  $\backslash\text{rm}$

For the user the word(s) is (are) important. The markup allowed for the IRs and the result in the copy are given in the accompanying table.

#### 3.2 Markup

The markup for IRs is near to natural. Precede the entry by a circumflex, or a double one in case of a silent<sup>4</sup> index entry.

##### Example IR markup

```

 $\wedge\{\backslash'e1\backslash'eve!\}$   $\wedge|\text{verbatim text}|$   $\wedge|\backslash\text{controlsequence}|$ 
 $\wedge\langle \text{a metalinguistic variable} \rangle$ 
 $\wedge\wedge\langle \text{a metalinguistic variable} \rangle$  %for silent ones, double the  $\wedge$ 
 $\{\backslash\text{sl}\wedge\{\text{ligatures}\}\}$  | '$|^|\backslash,| '$' | %from the TeX book script
 $\wedge\wedge\{\text{markup commands, see control sequences}\}$ 
 $\wedge\{\text{Lamport and } \backslash\text{LaTeX}\}$  %text and control sequences
%with sort keys

```

3. See *The  $\TeX$ book* 424, for the IR types, and what is typeset in the result. In  $\backslash\text{vref}$  the markup is inserted as replacement text of  $\backslash\text{next}$ . What is set in the index is governed by the macros which are included after  $\backslash\text{begindoublecolumns}$  in the  $\TeX$ book script.

4. Silent IRs mean that these will appear only in the index, not on the page.

### 3.3 Spaces

Spaces are difficult as always. In the IR they separate parts of the IR and are used in the word part.

- Just typing a space has as an effect that it will be neglected during sorting
- The markup ‘\\_’, a control space, will yield a space subject to sorting, according to the ordering table
- \space as markup will be neglected during sorting. This token is default member of the set of control sequences to be ignored. It will be set in the index as \\_.

**Question** What to do when part of a title should reappear in the index?

**Answer** The naive approach is to enclose that part by braces and precede it by a circumflex. However, that goes wrong because a title is stored and reused in many places. So copy the words and mark them as a silent IR.

**Example** Spaces

Explanation. \space belongs to the set of control sequences to be ignored, ICSs for short. This means that it is skipped with respect to sorting, except when it occurs as the last token of the word part. In that case they are ordered as a space, i.e., according to the lowest value. This explains the position of ‘\space.’ ‘\TeX,’ and ‘\TeX book,’ are subject to the default sorting keys. ‘xyza’ precedes ‘xyz beta,’ because the space is silent. When word ordering is preferred a \\_, a control space, must be included.

<code>^{\space}%an ignored cs</code>	Sorted result in file <code>index.srt</code>
<code>^{\ a} %control space</code>	
<code>^{\aa}</code>	<code>\space {} !0 1.</code>
<code>^{\ a b}</code>	<code>a\ \bf a{} !0 1.</code>
<code>^{\ a \TeX}</code>	<code>a\ a{} !0 1.</code>
<code>^{\ a\ \bf a}</code>	<code>a\ b{} !0 1.</code>
<code>^{\ \TeX book}</code>	<code>aa{} !0 1.</code>
<code>^{\ xyz beta}%space neglected in</code>	<code>a \TeX {} !0 1.</code>
<code>                                  %sorting</code>	<code>space{} !2 1.</code>
<code>^{\ xyza}</code>	<code>\TeX book{} !0 1.</code>
<code>^{\   \space }</code>	<code>xyza{} !0 1.</code>
<code>\sortindex\pasteupindex\bye</code>	<code>xyz beta{} !0 1.</code>

### 3.4 Special tokens

Tokens are either neglected or replaced by another sequence while sorting. `blue.tex` provides two sets of tokens to be ignored while sorting: `\conseqs` and `\consyms`.<sup>5</sup> Replacing a control sequence by another sequence is called associating a sorting key to the control sequence.

Active symbols can’t be part of the IR, for the moment.

5. There are two sets because of the handling of the space after the token in the result.

### 3.5 Tokens to be ignored

In practice I needed things like `\tt` as part of the IR, which must be neglected while sorting.<sup>6</sup> I decided to ignore those tokens while sorting and to include the tokens in the final `index.elm` as such. Default `blue.tex` knows about the following sets of tokens to be ignored.

```
\conseqs{\c\space\bf\it\rm\tt\sub\relax}
\consyms{\'\'\''\^{\~}
```

### 3.6 Sorting keys

In order to extend a set, use the macro `\add`.

**Example** Use of sorting keys

Default `blue.tex` provides the following sorting keys.

```
\srtkeypairs{\AmSTeX{amstex}
              \LAMSTeX{lamstex}
              \LaTeX{latex}
              \TeX{tex}
              \PS{PostScript}}
```

Suppose that we have `\fourtex` and that we like this to be sorted as '4tex.' This can be done by extending the set of `\srtkeypairs`, as follows.

```
\add\fourtex{4tex}to\srtkeypairs
Copy with ^{IR \fourtex}          then the file index.srt will
^{IR 1}                          contain the IRs
^{IR 5}
^{IR a}                            IR 1 !0 <pageno>.
%                                  IR \fourtex{} !0 <pageno>.
\sortindex %with 4tex for \fourtex IR 5 !0 <pageno>.
\pasteupindex%Set 'IR \fourtex{}  IR a !0 <pageno>.
%<pagenumbers>'
\bye
```

with `\fourtex` sorted on 4tex.

**Question** What to do when 'to' is part of the sorting key?

**Answer** Add an extra level of braces.

## 4 Ordering

A fundamental issue with indexes is the ordering. The ASCII table is not suited because lowercase and uppercase letters differ by 32. I decided to rank these as equal, more

<sup>6</sup> The reason is that `<`, `and` `>` are used, and printed wrongly.

precisely to assign the lowercase ASCII values to both. I prefer from the accompanying table the 1<sup>st</sup> column to the 2<sup>nd</sup> one.

Moreover, accented letters are not part of ASCII. How should we order for example e, é, è, ê, ë? I decided to rank accented letters equal to those without an accent, because I prefer from the accompanying table the 3<sup>rd</sup> column to the 4<sup>th</sup> one.

I know that non-letters precede letters, but what about their relative ordering? I decided to stay as close as possible to the ASCII ordering.

Then there is the problem of digits. In IRs they come as part of the word(s) and as page numbers. For the latter I used the numerical ordering. For the former I used the alphabetical ordering.<sup>7</sup>

Furthermore, a user can select the so-called word ordering,<sup>8</sup> by `\u`, T<sub>E</sub>Xnically a control space, as markup for a space. Personally, I like from the accompanying table the 5<sup>th</sup> column better than the 6<sup>th</sup>.

lower vs. upper case		accents vs. unaccented		word ordering	
el	el	el	el	sea lion	seal
Elève	em	élève	em	seal	sea lion
em	Elève	em	élève		

## 5 Typesetting the index

The specifications for typesetting a `blue.tex` index are

- represent the four IR types the same as in the T<sub>E</sub>Xbook
- set in two-columns, balanced, possibly preceded by one-column copy
- set subsidiary entries analogous to the T<sub>E</sub>Xbook
- indent continuation lines by 2em
- indent subsidiary entries by 1em

Users can edit `index.elm` – read: add markup – and provide the necessary macros in for example `\preindex`. In short follow Knuth. To please Frans Goddijn I introduced the tag `\numberstyle`, by default equal to `\oldstyle`.

## 6 Customization

A user might wish to interfere in places

- to include other tokens to be ignored while sorting
- to supply an ordering of his/her own
- to enrich the sorted and compressed file `index.elm`.

7. I could have applied a look ahead mechanism and use numerical ordering throughout. Maybe another time.

8. This means that a space precedes all letters. A space as such is neglected in the ordering.

### 6.1 Adding tokens

What are reasonable requirements to impose upon the handling of markup control sequences (cs for short)? In my opinion

- the cs must be defined
- `\makehref` writes the cs unexpanded
- ordering? unknown, and therefore must be supplied
- `\setupnxtokens` guards that the cs-s are written to `index.srt` and `index.elm`.

As a consequence I decided to neglect the ‘in between’ control sequences while sorting. For those who favour a one-pass job, I have provided the following, though.<sup>9</sup>

The extension of a set of tokens can be done via

```
\add\hfil to\conseqs or \add\'to\consyms
or \add\hfil{hfil}to\srtkeypairs
%with auxiliary \def\add#1to#2{...}
```

Each element from `\conseqs` is redefined in such a way that the control sequence token is written to the file with a space appended.<sup>10</sup>

### 6.2 Modifying ordering

A general way is to ‘copy’ the ordering table and to modify it.<sup>11</sup>

And what about a macro to add to the table? This can be done easily, and superficially looks convenient for an innocent user. At the moment I don’t trust the macros to be worthwhile for an innocent user, unless a very modest index has to be made. And this completes the circle: different ordering is not wanted, I guess.

### 6.3 The process and files involved

Like in `manmac`, `blue.tex` stores the raw IRs in the file `index`. The file `index`<sup>12</sup> is read and stored in an array for internal sorting. After sorting, the number of entries is reduced,<sup>13</sup> and the result is written to the file `index.srt`. Then, `index.srt` is transformed into the file `index.elm`.<sup>14</sup> The result is typeset via `\pasteupindex`. Schematically it comes down to the following.

9. It is simpler to add those control sequences to `index.elm`.

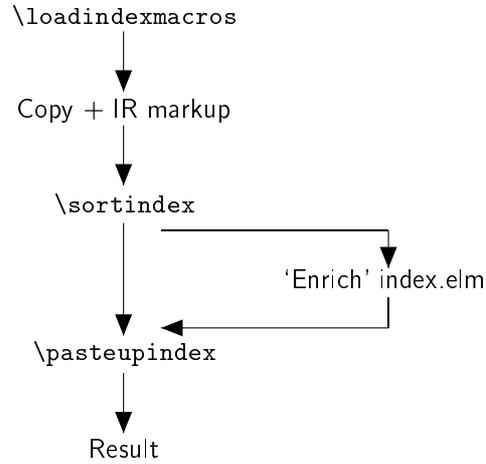
10. `\noexpand` is used instead of `\string`.

11. My `\fifo` is just a shortcut, which also prevents typos in assigning the ASCII values. For `\fifo`, see my ‘FIFO and LIFO sing the BLUES.’

12. Default index is the value of the toks variable `\irfile`, which is used in `\sortindex`.

13. Those which differ by page number are collected in one entry.

14. Default `index.elm` is the value of the toks variable `\indexfile`, which is used in `\pasteupindex`. The transformation abandons the IR syntax. The part which specifies the kind of IR is deleted and the word part marked up accordingly.



`\loadindexmacros` loads the index and sorting macros, and performs initializations. It is safeguarded against double loading.<sup>15</sup>

#### 6.4 Enriching the index

This use is necessary when for example

- control sequences have to be typeset
- special symbols are needed, or
- cross-references within the index are required.

The best way is to start from the `index.elm` file.

#### 6.5 Typesetting the enriched file

When the default name is used – `index.elm` – just say `\pasteupindex`. For another file name assign this name to the `\indexfile` variable, prior to the invocation of `\pasteupindex`.

## 7 Extras

Undoubtedly people favor their own subset of  $\text{T}_{\text{E}}\text{X}$ , or more likely  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ . The good news is you don't have to use BLUE's format system. I gathered the sorting and indexing stuff as an independent self-contained set in the file `plainindex.tpl`.

The bad news is that up till now I did not do much about preventing name clashes.

<sup>15</sup> I introduced this because I start each chapter with `\loadindexmacros`, independent from whether it is run on its own or as part of the total.

### **7.1 T<sub>E</sub>Xnical details**

The details with respect to indexing have been treated in 'BLUe's Indexes,' and the sorting aspects have been treated in 'Sorting in BLUe,' both available from the CTAN, in the directory `/pub/archive/info/pwt/<filename>`



# A Russian style for Babel: problems and solutions

Olga Lapko and Irina Makhovaya

Mir Publishers  
2, Pervyi Rizhskii Pereulok  
Moscow, 129820  
Russia  
irina@mir.msk.su

## Abstract

As with other languages using nonlatin basis there are some typographic features and national peculiarities that must be shown in the style. The paper describes the Russian style with macros `\captionrussian` for four standard Russian documents, `\daterussian`, `\Asbuk` and `\asbuk` for Russian alphabet counters and `\mathrussian` for Russian math operators. Some problems concerning the usage of this style (e.g. usage of different encodings) are described.

## 1 Introduction

As is generally known,  $\TeX$  is based on Latin alphabet and theoretically it is possible to use  $\TeX$  for other alphabets: Greek, Arabic, Cyrillic and so on. But there are a lot of troubles when we try to use  $\TeX$  for other alphabets in practice. Babel package is the first successful attempt to solve the problems of multilingual  $\TeX$ .

In this paper we discuss the concrete difficulties we encountered when creating the `russianb`<sup>1</sup> file for Babel. There are a lot of typographic features in Russian documents that can be separated into 3 classes:

1. the features that were borrowed from European typography, especially German and French;
2. the features that are peculiar for Russian typography only and there are no problems to describe them in the file `russianb`;

---

1. The filename of Babel style for Russian language is `russianb` as an analog of `germanb` to avoid probable confusion with  $\LaTeX$ 2.09 versions. Now the `russianb` is a beta-version – part of *CyrTUG-emTeX* appendix.

3. the features that are peculiar for Russian typography and there are some difficulties in describing and using them.

We also see two very important problems: variety of encoding schemes and necessity of portability of this file to different platforms, which we can solve only partially.

Now we shall describe the macros of the file `russianb` according to the classification.

## 2 Macros that were borrowed from other styles

The file `russianb` was derived initially from the original versions of `german` (for  $\text{\LaTeX}2.09$ ) and `germanb` (for  $\text{\LaTeX}2_{\epsilon}$ ) and `francais` (for  $\text{\LaTeX}2_{\epsilon}$ ). These files have the language-specific macros which Russian typographic rules need:

1. *from* `germanb`
  - macros for French and German double quotes. *Note:* French double quotes are created by METAFONT in Cyrillic font and have their own ligature (e.g. <<)
  - “shorthands” for hyphenation in compound words and words with nonliteral characters (Russian words are not so long as German ones but rather long too). As in `germanb` the sign " was done active, certainly.
  - `\lefthyphenmin–\righthyphenmin`: for the Russian (as well as for the German) language hyphenation patterns are used with values 2–2;
2. *from* `francais`
  - macros for `:`, `,`, `?`, `!` signs: the amount of white space is increased before these signs:  $\text{\TeX}$  looks for a space between word and this sign, then, if it is,  $\text{\TeX}$  “unskips” it and places a little white space about 0.1em. *Note:* in `francais`  $\text{\TeX}$  places such space in case of space between word and sign and the amount of this space is somewhat larger;
  - `\frenchspacing` is switched on;
  - some additional signs (as well as in `francais`) are described in our style, e.g. number sign.

## 3 Macros that are created in Russian style and have no problems in usage

There are some macros borrowed from `russian.sty` (for  $\text{\LaTeX}2.09$ ) of different releases:

1. macros for math operators whose names differ from English ones (e.g. in Russian manuscripts we write `tg` and `ctg` instead of `tan` and `cotan`);
2. there are additional macros for printing counter values with uppercase and lowercase Cyrillic letters (`\Asbuk` and `\asbuk` as analogs to `\Alph` and `\alph`).

We created an additional “shorthand” for Russian emdash (“---”): in our printing documents this sign is shorter a little, and it has spaces about 0.2 of font size (e.g. for size 10pt it is about 2pt) around and never breaks with word before emdash. *Note:* the macro for explicit hyphen sign (“-”) was, certainly, rewritten because of creating macro for Russian emdash.

#### 4 Macros, which have some difficulties or questions in usage

In this section we discuss macros of breaking in text formulas. By Russian typography tradition we must repeat last sign in broken formula on the next line.

There is a package which includes macros for repeating signs in formulas.

This package includes two possible ways:

1. *hand breaking* – in this way `\binoppenalty = \relpenalty = 10000`; for breaking we must use commands `\brokenbin{ }` and `\brokenrel{ }`;
2. *automatic breaking* – in this way `\relpenalty > \binoppenalty > 10000`; some signs are equal to `\mathcode=8000` and divided into two groups: binary and relational signs `+ - < > =` allow break after them, signs `* ( [ / . ,` protect break; also almost all mathematic signs are rewritten using new commands `\brkbin` and `\brkrel` to allow or protect breaking, e.g.:

```
\def\wedge {\brkbin{\mathchar"225E}}
\def\gg      {\brkrel{\mathchar"321D}}
\def\exists  {\mathchar"0239\unbrk}
\def\bigl#1 {\mathopen{\big#1}\unbrk}
\def\bigm#1 {\mathrel {\big#1}\unbrk}
\def\langle  {\delimiter"426830A \unbrk}
```

the command `\not`, for example, must be redefined:

```
\def\not#1 {\brkrel{\mathchar"3236 #1}}
```

and so on. In this case  $\TeX$  breaks formulas by itself but sometimes we must handle breaking using special commands `\unbrk` or `\allowbrk`.

There are the drawbacks in this package but rather exotic:

- one must write `$x \brkbin{\+}^1 y$` instead of `$x +^1 y$`;
- operators `\sin` must be written with arguments in parenthesis;
- in formulas like `x + ... + y` one must write `$x \unbrk + \ldots + y$` to protect first breaking (or breaking signs must look forward);
- in case the signs `^` and `_` are redescrbed, we cannot use  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ 's macros like `\Sb... \endSb` (i.e. `^ \bgroup ... \egroup`) otherwise it is impossible to use something as `^ \leq` and `^ *`.

As you see we must rewrite all definitions for binary and relation signs and some signs which protect the break after them. In other words we must rewrite  $\TeX$  formats.

Now this style exists as additional<sup>2</sup> and is switched on by user.

<sup>2</sup> Now it is beta-version style.

## 5 Encoding and font problems

The file `russianb` was created for “nonlatin user” who uses Cyrillic alphabet. One of the problems is: there are a lot of different encodings in which Cyrillic letters have different codes.

Because of this, our file has some particular features as compared with Babel’s analogs.

To make this style independent of encoding, we have to use macros-names instead of Cyrillic letters themselves. Russian letters and signs in this style are used in macros for date (`\daterussian`) and the strings for four standard styles of L<sup>A</sup>T<sub>E</sub>X (`\captionrussian`), and also in commands for printing counter values with use of Cyrillic uppercase and lowercase letters (`\Asbuk` and `\asbuk`).

The macros-names for Cyrillic letters and some signs are switched on by a file-satellite (e.g. `lhrcod.sty`) which is created for necessary encoding. This file also switches on Cyrillic font family.<sup>3</sup>

In Russia, papers are typesetted by special encodings where the Cyrillic letters are joined with Latin ones and placed by necessary encoding in upper part of the code table, so there is the tradition of using Russian letters in macronames (the usage of familiar, Russian words is more convenient – Russian letters are letters too, aren’t they?). To tell more: there are packages which are based on Russian-word commands. So, in `russianb` Cyrillic letters were made as `\catcode\letter`.

Now we must say that `\mathcode` for Russian letters equals to 70??, i.e. we set class 7 – variable family and use font of `\fam0` (`\rm`) to join the Russian typography tradition.

## 6 What we must do

Now our style is adapted for Russian “8-bit” documentation, in which we sometimes use Latin fragments. In this case a file-satellite `russianb` (in current version we input `lhrcod` for Alternative encoding) simply declares new font family and encoding at the beginning of document<sup>4</sup> and then toggles Russian/Latin hyphenation only – this way takes less memory. For Russian-Latin papers typesetted with transliteration (using Latin letters as Cyrillic) we must declare sometimes to Latin letters that they are Cyrillic, so we have to toggle fonts and encodings too. These two ways should be described and divided in `russianb`.

The main problem of file `russianb` is that we have to use macros-names of Cyrillic letters. Now these letters are described as `\def\CYRa{a}`. In this case the commands `\uppercase` and `\lowercase` don’t work correctly (e.g we can’t use standard style book

---

3. Now the switching on of the Cyrillic family is made for L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  only.

4. Such file now is created for L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  only.

– we will never get uppercase `\chaptername` in running heads).<sup>5</sup> Maybe other definitions for letters or/and strings for styles will solve this problem.

The paragraphs above described some solutions of portability of the file `russianb` to different platforms. Since we have no practice in other platforms (e.g. Unix) we don't know about probable difficulties there.

Our work is the first attempt only and we hope to provoke discussion and further work of our colleagues. We hope also that `Omega` package can help to solve the described problems, especially those that are connected with encoding schemes and portability.

## 7 Acknowledgements

In conclusion we would like to note that a lot of our colleagues make the valuable contributions to creation of the file `russianb`: Mikhail Grinchuk, Eugenii Ivanov, Andrey Slepukhin, Yurii Tyumentsev, and others. We are very much obliged to all of them.

## References

- [1] J. Braams. Babel, a multilingual style-option system for use with L<sup>A</sup>T<sub>E</sub>X's standard document styles. *TUGBoat*, 12.2:291–302, 1991.
- [2] P. Gilenson. *Spravochnik tekhnicheskogo redaktora*. Moscow, Kniga, 1979.
- [3] M. Goossens, F. Mittelbach, and A. Samarin. *The L<sup>A</sup>T<sub>E</sub>X Companion*. Addison-Wesley, Reading, MA, 1994.
- [4] Y. Haralambous and J. Plaice. First application of `Omega`: Greek, arabic, khmer, poetica, iso 10646/unicode, etc. *TUGBoat*, 15.3:344–352, 1994.
- [5] A. Khodulev and I Makhovaya. *On T<sub>E</sub>X experience in Mir Publishers*. Proceedings of the 7th EuroT<sub>E</sub>X Conference, Prague, 1992.
- [6] O. Lapko. *MAKEFONT as part of CyrTUG-em T<sub>E</sub>X package*. Proceedings of the eight European T<sub>E</sub>X Conference, Gdańsk, 1994.

---

5. If Cyrillic letters would be described as `\def\CYRa{\char160}` or `\chardef\CYRa=160` we won't get uppercase/lowercase texts in principle.



# Data with $\delta\alpha\text{T}\text{E}\text{X}$

Andries Lenstra, Steven Kliffen and Ruud Koning

lenstr@sci.kun.nl

## Abstract

The authors explain how to handle data in  $\text{T}\text{E}\text{X}$  documents, in particular, how to avoid ever having to type in – and check! – the same data or text twice. These data may be stored in ordinary (non- $\text{T}\text{E}\text{X}$ ) databases, in ASCII files arranged according to the easy  $\delta\alpha\text{T}$  format, or in the  $\text{T}\text{E}\text{X}$  document itself.  $\delta\alpha\text{T}\text{E}\text{X}$  works in plain  $\text{T}\text{E}\text{X}$  and is supposed to work in  $\text{L}\text{A}\text{T}\text{E}\text{X}$ .

## 1 Introduction

As soon as one uses the same data more than once, or the same text with different data, a classical problem arises: how far should one go in leaving the repetitions to the machine?  $\delta\alpha\text{T}\text{E}\text{X}$ , a collection of  $\text{T}\text{E}\text{X}$  macros for storing and retrieving data, is an engine that enables one to go all the way, so that document source texts are as short as possible and integrity of the data is guaranteed, as well as uniformity in typesetting.

That  $\text{T}\text{E}\text{X}$  is suitable for repetitive tasks such as mail merges, was already shown e.g. in [2]; that a more general approach is also feasible, and that, in fact,  $\text{T}\text{E}\text{X}$  is an ideal word processor for data handling, is what we hope to show here.

The key feature of  $\text{T}\text{E}\text{X}$ , of course, is its programmability in plain, readable text, so that in the first place the manipulating c.q. processing of data and the perfect fine-tuning of text to data are easy and rewarding tasks – certainly also for non- $\text{T}\text{E}\text{X}$ nicians, as practice has shown. One could say that the personalizing of printing, or data handling in general, opens up a whole new realm of applications for the power of  $\text{T}\text{E}\text{X}$  as a programming language. Now this beautiful machinery not only generates the typography, but prior to this the very text. Examples can be found in [2] and [1].

In the second place the existence of expandable, plain text macros has great benefits in the set-up and maintenance of databases. Such macros can be a substitute for real, explicit data, and their expansion can be changed according to different circumstances. Imagine, for instance, someone who uses  $\text{T}\text{E}\text{X}$  for the typesetting of concert programmes in several languages. In his database of music pieces, the contents of the field 'key' belonging to some piece in A major (some music pieces have a coordinate called key; 'A

major' is a key) will not be `A major`, but something like `\Amaj`, which auxiliary files will translate into 'A major', 'A Dur', 'La majeur', or 'A grote tert's', etc., dependent on the desired language. Or think of the printing of a price list: the contents of the field 'price' need not be the explicit digits of the price, but may expand into a formula from which the explicit digits will follow after a calculation with adjustable parameters.

This possibility of filling the contents of a data base with 'meta'data,  $\TeX$  macros that expand to the actual, printed data, can even be exploited further. Consider in the first example the field 'composer' (most of the music pieces have a composer). Instead of putting there a specific name, e.g. Mozart, or W.A. Mozart, or W.A. Mozart (1756--1791), why not put the command

```
\Some Later To Be Specified Set Of Fields
  Belonging To The Entry 'Mozart2'
```

(in  $\delta\alpha\TeX$  abbreviated as `*Mozart2* *`), if there is a second database, of composers, with an entry that is identified by the string `Mozart2`, and  $\TeX$  is told where to find this entry? – Then all questions as to the inclusion of initials and dates can safely be left unanswered until the time of typesetting. Besides, apart from being more versatile and less error-prone, in the presence of this second database such an approach is mandatory because of the problem of data integrity: as soon as the same explicit data are entered more than once, how can one be sure that a change or correction is carried through in all occurrences in all documents?

One will realize that with this method of letting fields of one entry refer to other entries by means of commands built around an entry-identifying string, one can give shape to many relationships between data, in a readable and easily memorized way, so that with minimal effort a coherent, structured set of data is obtained. The workings of  $\delta\alpha\TeX$ , which uses the idea of the identifying string, are now readily sketched.

## 2 Sketch of $\delta\alpha\TeX$

With  $\delta\alpha\TeX$ , the referencing facility also exists in ordinary document source text. Suppose  $\TeX$  has learned  $\delta\alpha\TeX$  and that one writes

```
\Of *Mozart2* * {\bf\Name, \Initials} (\TownOfBirth)
```

in the document. Suppose also that earlier in the document, or in a separate data file, the following lines occur:

```
**Mozart2* \Name Mozart; \Initials W.A.;\TownOfBirth Salzburg;
. . .
**zzzz*
```

Then the typeset result will be

**Mozart, W.A.** (Salzburg)

The string `*Mozart2* *` is an abbreviation for

```
{\Of *Mozart2* * \Type}
```

The `\Type` is a field that every entry should have; for `Mozart2` one could choose `\Type` to be `\Composer`:

```
**Mozart2* . . . \Type \Composer;\YearOfBirth 1756; . . .
```

Then `\Type` will expand to `\Composer` and `\Composer` acts as a template, say

```
\def\Composer{%
  {\bf \Name, \Initials}
  (\TownOfBirth, \YearOfBirth)%
}%
```

With this definition of `\Composer` the typeset result of writing `*Mozart2* *` will be **Mozart, W.A.** (Salzburg, 1756)

Instead of writing

```
\Of *Mozart2* * \TownOfBirth
```

(with a space after the final `*`) in order to get 'Salzburg', one can also write

```
\def\Composer{\TownOfBirth}*Mozart2* *
```

The latter may seem too much if it is used only once, but imagine the possibilities if the referencing is repeated. For instance, with these three templates

```
\def\Mpiece{\Comp}%
\def\Composer{\TownOfBirth}%
\def\Town{\Country}%
```

the result of `*(some music piece)* *` will be the country where one can find the town where the composer was born, assuming that one filled the proper fields in the proper databases with the proper types and the proper three-starred identifying strings c.q. data. In the next section we will do this explicitly as an illustration of the  $\delta\alpha\TeX$  format for storing data.

For repetitive tasks, however, it is immediately clear that the template is the perfect tool. A template is put implicitly in the document by `*(an identifying string)* *` as above; writing three `\Composers`

```
*Bach* *
*Mozart2* *
*Beethoven* *
```

(or as one line `*Bach* **Mozart2* **Beethoven* *`) gives three times the expansion of `\Composer`, in the first case separated by spaces. This repetition is done automatically by `\Filter`. In  $\delta\alpha\TeX$  the `\Filter` command filters a data file according to a certain `\Type`, i.e. it considers every entry in consecutive order and, if the `\Type` corresponds to the given `\Type`, e.g. `\Composer`, it expands the template, in this case `\Composer`. So this is the way to perform mail merges, in which a form letter, the template, is merged with a data file that contains the data of the persons that are supposed to receive the letters. The exact description and an example can be found in Section 7.

Explicitly asking for one or more fields of a certain entry, as in

```
The composer \Of *Mozart2* * \Initials{} \Name{} ...
```

```
At that time, \YearOfBirth, the town \TownOfBirth{} was ...
```

follows the rule that should be obvious from the use of `\Of`: all field names give the contents belonging to the last identified entry, which means that as soon as `\Of` *\*(a different identifying string)\** appears, `\Of` *\*Beethoven\** say, all field contents are of the new entry, *Beethoven*; those of *Mozart2* are forgotten. If *Mozart2* has a field `\YearOfBirth`, but *Beethoven* has not, then after the appearance of `\Of` *\*Mozart2\** the text `\Of` *\*Beethoven\** `\YearOfBirth` will cause the error message

```
Use of \YearOfBirthOf doesn't match its definition.
```

and if the same text is not preceded by any `\Of` *\*(string identifying an entry which has a field \YearOfBirth)\**, then it will cause the error message

```
! Undefined control sequence.
```

With the help of the  $\delta\alpha\text{T}\text{E}\text{X}$  command `\IfField\YearOfBirth\Exists` such embarrassing moments can be avoided. Tools like this one can be found in Section 5.

Under circumstances to be explained later, it will save time to tell  $\text{T}\text{E}\text{X}$  in which database it should look for the desired identifying string. Such a specification may take the place of the space between the second and the third `*`, as in

```
*Mozart2*c:/music/data/comp.dat*
```

or in

```
\def\Dpath{c:/music/data/}%
\Of *Mozart2*\Dpath comp* \YearOfBirth
```

the extension `.dat` being supplied by  $\delta\alpha\text{T}\text{E}\text{X}$  if no extension has been specified.

### 3 Storing data

$\delta\alpha\text{T}\text{E}\text{X}$  has its own way of storing data, the  $\delta\alpha\text{T}$  format. In order for data to be accessible to  $\delta\alpha\text{T}\text{E}\text{X}$ , they should be stored according to this format, for instance in a  $\delta\alpha\text{T}$  file. The user of  $\delta\alpha\text{T}\text{E}\text{X}$  may use his own data base programs as long as they are capable of producing intermediate ASCII files; these are the subject of Section 6. As soon as the composition of the identifying string has been specified, the conversion to the  $\delta\alpha\text{T}$  format can be taken care of by  $\delta\alpha\text{T}\text{E}\text{X}$ .

Data base programs other than  $\delta\alpha\text{T}\text{E}\text{X}$  often offer facilities, such as sorting, that until now for  $\delta\alpha\text{T}$  files only exist in cooperation with such a data base program via an intermediate file, or with the operating system. (Sorting inside  $\text{T}\text{E}\text{X}$  is possible; see [3].) The  $\delta\alpha\text{T}$  format, on the other hand, is very simple, versatile, and easy to learn, while  $\delta\alpha\text{T}\text{E}\text{X}$  has a very powerful sorting-out mechanism. Files according to the  $\delta\alpha\text{T}$  format are as portable as ordinary  $\text{T}\text{E}\text{X}$  document source text files; in fact, they can be integrated, wholly or partially, into the document, as we shall see.

In the  $\delta\alpha\text{T}$  format data are stored in data blocks. A  $\delta\alpha\text{T}$  file has a name with a one, two, or three character extension, but not necessarily with the extension `.dat`, and consists of a number of data blocks. Before and between these data blocks one should put only `\NoDefaults` and `\Default` commands (to be described shortly) or comments,

and after the last data block a  $\delta\alpha T$  file should be empty – at least until one exactly knows what is going on.

A data block consists of a number of entries. An entry starts with **\*(the identifying string)\*** followed by any number of fields and occurrences of this three-starred identifying string. A field consists of optional spaces followed by a control word denoting the field name (a control word is of the form  $\backslash$ (a string of letters without spaces)) followed by the field contents followed by a semicolon,

$\langle$ field name $\rangle$   $\langle$ field contents $\rangle$ ;

and an entry ends with the occurrence of **\*(a different identifying string)\***, with which a new entry starts, except when this last string is **zzzz**. The string **zzzz** is supposed to be different from all strings used for entries; after **\*\*zzzz\*** the data block ends. The fields may be broken by the identifying strings; the entry minus all occurrences of **\*(the identifying string)\*** should be a sequence of fields. So

```
**Mozart2*\Name Mozart;\Type\Composer; \ChristianNames Wolfgang%
**Mozart2* Amadeus ;**Beethoven* \Name Van Beethoven;\Type \Composer
; **Beethoven***Salzburg*\Type\Town;\Country Austria;
**zzzz*
```

is a data block, albeit a somewhat untidy one. The rationale for allowing the identifying strings breaking the fields is that **\*(the identifying string)\*** should occur at the beginning of every line of the entry and nowhere else, so that no two entries share the same line. For such data blocks some manipulations with the data are possible with the help of common outside tools. Before we discuss these possibilities, let us clean up the above data block, adding some data and  $\backslash$ Defaults.

```
\NoDefaults
\Default\Type\Composer;
\Default\Name\Ident;
**Mozart2*\Name Mozart;\TownOfBirth *Salzburg* *;
  \ChristianNames Wolfgang%
**Mozart2* Amadeus ;
**Mozart2* \YearOfBirth1756; \YearOfDeath 1791;
**Beethoven* \Name Van \Ident;
**Bach*
**Salzburg*\Type\Town;\Country Austria;
**KV 488*\Type\Mpiece;\Key \Amaj;\Comp *Mozart2* *;
**zzzz*
```

Now with the definitions from the previous section **\*KV 488\*** will indeed give 'Austria'. The control word  $\backslash$ Ident always expands to the identifying string of the entry. For the entry **Bach** two fields exist (by  $\backslash$ Default), the field with field name  $\backslash$ Name and contents  $\backslash$ Ident, and the field with field name  $\backslash$ Type and contents  $\backslash$ Composer. In the entry **Mozart2** the contents of the field with field name  $\backslash$ ChristianNames are

Wolfgang Amadeus , with a space at the end. Without the %-sign there would have been two spaces between these names.

In practice, one probably would not mix up so many different `\Types` in one data block, and one would add a list of field names for every `\Type`, just to be sure that no such things happen as using together `\Forenames` and `\ChristianNames`. But one has the complete freedom to invent new fields on the spot and to list them in any order, independent from other entries, as long as all fields are closed by the delimiter `;` (semicolon). This should be carefully checked, apart from the explicit data themselves.

Vice versa, as soon as a semicolon appears, chances are high that it terminates the field contents. If one needs the semicolon in the contents of a field, one should use `\Semicolon`. The typeset asterisk, `*`, is available as `\Star`.

The choice of the identifying string is free, as long as it identifies the entry (i.e. all entries have different strings), is not empty, and only contains letters, digits, spaces, and no two spaces in a row.  $\delta\alpha\TeX$  will only check this to a limited extent.

Whole, closed data blocks can be put anywhere in the document. A  $\delta\alpha\TeX$  file can be absorbed, i.e. memorized, searched for an entry, or `\Filtered`; a data block in the document is always absorbed.

### 3.1 Manipulations with data blocks

If all entries of a data block have their `**⟨identifying string⟩*` at the beginning of every line and nowhere else, then sorting the block on the first column means sorting the data on the identifying string. This is useful in its own right and also a means of bringing together the different parts of an entry that is scattered around the block. A way of taking care of broken fields would be by inserting sorting dummies like `\zz1`, `\zz2`, `\defined as {}`.

Moreover, with the help of the `grep` command, well known to `Unix` users, one can sort on field contents by having `\Filter` make a list of these contents paired with the identifying strings,

```
⟨(possibly processed) field contents⟩ ⟨identifying string⟩
```

sorting the list on field contents, and asking `grep` to rearrange the data block according to the new order of the identifying strings.

Finally, `grep` allows preprocessing of `TEX` documents in which  $\delta\alpha\TeX$  is invoked. The searching of files for strings by `grep` will be faster than by  $\delta\alpha\TeX$ , so that having `grep` search the document for three-starred identifying strings, search the  $\delta\alpha\TeX$  files for the lines on which these occur, and putting these lines in the document (taking care of the proper `\Defaults` and the closing of the data block by `**zzzz*`) will sometimes save time.

## 4 System set-up

For simple  $\delta\alpha\text{T}\text{E}\text{X}$  tasks there is nothing left to learn except the use of a few tools, and the fact that the size of data blocks in documents is limited because they are absorbed (how much  $\text{T}\text{E}\text{X}$  can absorb, depends on the local implementation). If  $\delta\alpha\text{T}\text{E}\text{X}$  has to search unspecified  $\delta\alpha\text{T}$  files, however,  $\delta\alpha\text{T}\text{E}\text{X}$  has to know these files and may need some guidance in their treatment. The

`\TheDataFiles`  $\langle$ first file name $\rangle$ ( $x$ ) ... `\InSearchOrder` command tells  $\delta\alpha\text{T}\text{E}\text{X}$  which  $\delta\alpha\text{T}$  files should be absorbed ( $(x)=(a)$ ), which  $\delta\alpha\text{T}$  files should be searched if no search-file has been specified after the second star ( $(x)=(s)$  or  $(x)=()$ ) and in what order they should be searched, and which of these default search-files get a 'search-only' treatment ( $(x)=(s)$ ) under `\DefaultMemoryProtection`.

Whenever

`* $\langle$ an identifying string $\rangle$ * $\langle$ optional  $\delta\alpha\text{T}$  file specification $\rangle$ *`

or

`\Of * $\langle$ an identifying string $\rangle$ * $\langle$ optional  $\delta\alpha\text{T}$  file specification $\rangle$ *  $\langle$ field name $\rangle$`

invoke  $\delta\alpha\text{T}\text{E}\text{X}$  to retrieve data,  $\delta\alpha\text{T}\text{E}\text{X}$  will normally try to remember these. If the data block was absorbed in which the identifying string occurs, or if  $\delta\alpha\text{T}\text{E}\text{X}$  has looked up the data already once before, it will succeed in searching its memory and find the data. Otherwise the data are looked up in the search-file specified after the second star or, if there is no such file, in the default search-files, i.e. all non-absorbed files of the list of `\TheDataFiles`.

So the normal way is that the more strings  $\delta\alpha\text{T}\text{E}\text{X}$  looks up, the more data it will remember. This means that when  $\delta\alpha\text{T}\text{E}\text{X}$  is invoked for many different strings,  $\text{T}\text{E}\text{X}$  may run out of memory. Therefore `\DefaultMemoryProtection` allows for something special: if a  $\delta\alpha\text{T}$  file has been specified in the list as  $(s)$ , 'search-only', and has been specified between the second and third star, as in `* $\langle$ an identifying string $\rangle$ * $\langle$ name of a search-only  $\delta\alpha\text{T}$  file $\rangle$ *`, then  $\delta\alpha\text{T}\text{E}\text{X}$  will not try to remember the data but will look them up immediately in the specified file, use them, and forget them.

In the `\DefaultMemoryProtection` mode the protection of the memory has to be activated by specifying a search-only file after the second star. However, in the `\StrongMemoryProtection` mode the memory is always protected; the only way to have  $\delta\alpha\text{T}\text{E}\text{X}$  search its memory immediately is by specifying an absorbed,  $(a)$ , file after the second star. In this mode the specification of any non-absorbed file makes  $\delta\alpha\text{T}\text{E}\text{X}$  act as after the specification of a search-only file in the default mode. If there is no specification at all,  $\delta\alpha\text{T}\text{E}\text{X}$  will not try to remember the data (this would involve the forming of a control sequence, and the number of control sequences that  $\text{T}\text{E}\text{X}$  can see in a single run is limited), but search the default search-files for them. If it finds the data, it uses them and forgets them; in the absence of success it will conclude that the data must have been absorbed and only then search its memory.

$\delta\alpha\text{T}\text{E}\text{X}$  always starts in `\DefaultMemoryProtection` but the user can alternate between this mode and `\StrongMemoryProtection`.

## 4.1 File specification

When between the second and third star an absorbed file or a search-only file is specified, taking the place of the space, then in order for  $\delta\alpha\text{T}\text{E}\text{X}$  to be able to recognize this file as absorbed or search-only, it is not only necessary that the `\TheDataFiles` command has been executed already, but also that the ‘canonical’ file name of this file on this place in the document is the same as when it was listed in the list of `\TheDataFiles`.

The canonical file name is the result of the following reduction:  $\delta\alpha\text{T}\text{E}\text{X}$  expands all control sequences in the typed-in file name, e.g. `\Dpath` in `\Dpath comp` in Section 2, then it tries to remove possible spaces at the beginning and at the end, and checks if the result has an extension, i.e. ends on `.x` or `.xy` or `.xyz`, with `x`, `y`, and `z` letters. If there is an extension then this is preserved, otherwise the extension `.dat` is attached. So always writing the same name is by no means necessary, but for the same  $\delta\alpha\text{T}$  file one should not switch between file name with full path included, and file name.

If the expansions of the control sequences in the typed-in file names do not begin or end with a space and there are no `+` signs in file names, then there will most probably be no problems with spaces around the file names or around the delimiters `(a)`, `(s)`, and `()`. For instance, one can write `*⟨identifying string⟩* \Dpath comp *` as well as `*⟨identifying string⟩*\Dpath comp*` and

```
\TheDataFiles
  \Dpath comp.dat (a)
  \Dpath mpiece(s)town ()
\InSearchOrder
```

as well as

```
\TheDataFiles
  \Dpath comp.dat(a)\Dpath
  mpiece (s)
  town()\InSearchOrder
```

– `\TheDataFiles`, for that matter, provides a check-list.

## 5 Tools

### 5.1 Default fields

The `\NoDefaults` and `\Default` commands, introduced in Section 3, may only be given outside a data block, so when one wants to change the default contents of a field, or wants to add a default field, then one should first put `**zzzz*`. The syntax of `\Default` is

```
\Default⟨field name⟩ ⟨default field contents⟩;⟨space⟩
```

which is the same as `\Default ⟨default field⟩ ⟨space⟩`. The necessary space after the semicolon could be provided by giving every `\Default` a line of its own.  $\delta\alpha\text{T}\text{E}\text{X}$  puts the default fields immediately behind the first `*⟨identifying string⟩*` of an entry in the given

order (new default fields behind the old ones), before the fields that are explicitly typed in. The contents of a field are overridden by those of a subsequent field with the same field name. This holds for all fields, default or explicitly typed in, so that by

```
\Default<same field name> <new default field contents>; <space>
```

one can change the default contents of a field.  $\delta\alpha\text{T}\text{E}\text{X}$  starts with an empty list and will not change this list unless it is told to do so by a `\Default` or `\NoDefaults` command. In order to know the exact contents of this list at every moment and to avoid surprise results when  $\delta\alpha\text{T}\text{E}\text{X}$  absorbs or searches a sequence of  $\delta\alpha\text{T}$  files, one should have it emptied often by the `\NoDefaults` command – for instance at the beginning of every  $\delta\alpha\text{T}$  file.

## 5.2 Conditionals

We introduce three `\If...` commands. Like  $\text{T}\text{E}\text{X}$ 's ordinary `\if...`s, they have an optional `\else` part, are closed with `\fi`, and may be nested.  $\delta\alpha\text{T}\text{E}\text{X}$  allows `\Fi` instead of `\fi`. All examples refer to the second data block of Section 3.

The definition of a control word can be inspected by the command

```
\IfCs<control word>\IsDefinedAs{<a string>}
```

After `\Of *<an identifying string>* *` the control word `\Ident` is defined as `<this identifying string>`, so that the typeset result of

```
\Of *Mozart2* *
  \IfCs\Ident\IsDefinedAs{Mozart2}%
    {\bf\Name}%
  \Fi
```

is **Mozart**. With `\Of *<any other identifying string>* *` the result will be nothing.

The possibility of checking on the existence of fields was announced already in Section 2. The condition

```
\IfField<field name>\Exists
```

is true for all entries for the field names `\Type` and `\Name`, e.g.

```
'\Of *Bach* * \IfField\Name\Exists' is true;
```

```
'\Of *Bach* * \IfField\TownOfBirth\Exists' is false.
```

Finally, here is a facility for testing if an existing field looks like a given field:

```
\IfExistingField\LooksLike<field name> <given field contents>;
```

In this comparison the field contents are left untouched by  $\text{T}\text{E}\text{X}$ , i.e. they are not expanded or processed otherwise. So the typeset result of

```
\Of *Mozart2* *
  \IfField \TownOfBirth \Exists
  \IfExistingField\LooksLike \TownOfBirth*Salzburg* *;%
    {\bf\Name}%
  \Fi
\Fi
```

is **'Mozart'**. With `\Of *{any other identifying string from our data block}* *` the result will be nothing. Furthermore:

```
\Of *Mozart2* * \IfExistsField\LooksLike \Name Mozart; is true,
\Of *Beethoven* * \IfExistsField\LooksLike\Name Van \Ident; is true,
\Of *Beethoven* * \IfExistsField\LooksLike\Name Van Beethoven; is false,
\Of *Bach* * \IfExistsField\LooksLike \Type \Composer ; is true.
```

The comparison of a given string with processed field contents is a different matter. Consider, for instance, the string `Van Beethoven`. This string is equal to the result of processing the field contents `Van \Ident` in the following way: expand all that is expandable until there is nothing expandable left. After `\Of *Beethoven* *` the process of producing out of `\Name` the field contents `Van \Ident` themselves, also only involves expansion. Therefore, for the control word `\nAme` the effect of

```
\Of *Beethoven* * \edef\nAme{\Name}%
```

is the same as

```
\def\nAme{Van Beethoven}%
```

All other fields `\Name` in our data block also have contents that can be expanded completely, i.e. until there is nothing expandable left. This means that in this case we are back to the `\IfCs` technique learnt above. The typeset result of

```
\Of *Beethoven* * \edef\nAme{\Name}%
\IfCs\nAme\IsDefinedAs{Van Beethoven}%
{\bf\nAme}%
\Fi
```

is **'Van Beethoven'**, and with `\Of *{any other identifying string}* *` the test will work but the result will be nothing. (We could have used `\Name` itself instead of `\nAme`, but then the protection against misuse of the field name `\Name`, as exposed in Section 2, would have disappeared.)

Field contents of the form `*{an identifying string}* *` *cannot* be expanded completely; they cannot be put in an `\edef` without  $\TeX$  having to stop and complain, or behaving in some other undesirable manner. If in our data block there had been an entry of which the `\Name` had contents of this form, the above test would not work properly and the `\IfExistsField\LooksLike` test should be preferred.

Now for processed field contents where the processing involves *more* than expansion. The comparison of

1. field contents that refer to other entries, but of which the stage that is to be compared is completely expandable, with
2. strings in which nothing expandable is left,

is possible with `\xdef` ( $=\global\edef$ ). Consider, for instance, the string `Austria`, the result of `*KV 488* *` with the templates of Section 2. After `\Of *Salzburg* *`, the process of producing `Austria` out of `\Country` only involves expansion, but after `\Of *KV 488* *` the process of producing `Austria` out of `\Comp` involves *more* than sheer expansion, and

```
\Of *KV 488* * \edef\cOmp{\Comp}%
```

is not recommended. The `\Comp` field contents `*Mozart2*` refer to other entries, but the stage `\Country`, that is to be compared, is completely expandable. The solution is to have  $\TeX$  do its work, but put the result aside for later testing, wrapped in a control word: when

```
\def\Town{\xdef\cOuntry{\Country}}*KV 488* %
```

hence

```
\IfCs\cOuntry\IsDefinedAs{Austria}%
```

is true. Of course, more would be needed to make the test work for arbitrary `\Mpieces`. Now the `\Comp` and all other fields have to exist down to `\Country`.

One can also test immediately and export the result. This will be shown for strings and field contents that should not be touched. If the `\Country` field contents are `*Austria*`, then after

```
\newif\ifTheRightOne
\def\Town{\global\TheRightOnefalse
\IfExistingField\LooksLike\Country*Austria* *;%
\global\TheRightOnetrue
\Fi
}%
*KV 488* %
```

it will be seen that `\ifTheRightOne` is true.

### 5.3 Wrapping

In the last test only the result of the comparison was available, not a control word like `\cOuntry` for further testing or processing. As an analogue to `\xdef`, the `\WrapIn` command provides this facility whenever  $\TeX$  should not touch the field contents to be compared. After `\Of` *\*(an identifying string)\**

```
\WrapIn (pseudo field name) (field name)
```

is equivalent to

```
\gdef (pseudo field name){(field contents)}
```

Here is the penultimate example again; the `\Country` field contents are assumed to be `*Austria*`. The condition in

```
\def\Town{\WrapIn\cOuntry\Country}*KV 488* %
\IfCs\cOuntry\IsDefinedAs{*Austria* *}%
```

is true, and `\cOuntry` is available for other purposes.

## 6 Conversion

Most data base programs are capable of reading and producing ASCII data files.  $\delta\alpha\TeX$  offers two utilities that facilitate the cooperation with such programs. The first reads almost all ASCII data files that the latter may produce, and converts these data files

into  $\delta\alpha T$  files. The second converts, for almost every data base program, a  $\delta\alpha T$  file into the particular ASCII format that is readable for the program. These utilities are currently under construction.

## 7 Filtering

The `\Filter` command, announced in Section 2, filters a  $\delta\alpha T$  file according to a certain `\Type`:

```
\Filter < $\delta\alpha T$  file specification>\Type <the filter type>
```

for instance,

```
\Filter \Dpath comp \Type\Composer
```

considers every entry of `\Dpath comp` in consecutive order and, if the `\Type` is equal to `\Composer`, it expands `\Composer`.  $\delta\alpha T\text{E}\text{X}$  reduces the  $\delta\alpha T$  file specification to the canonical file name, as explained in Section 4. A space before '`\Type`' should not cause any problems.

The restriction to one `\Type` is not essential. Suppose one has a big  $\delta\alpha T$  file `mail.dat` with many entries of different `\Types` sorted on some postal code. If all entries should receive a letter, then `\Filtering` the  $\delta\alpha T$  file once for every `\Type` would destroy the ordering. The solution is to replace, in `mail.dat`, all occurrences of '`\Type`' by '`\ProType`' (i.e. to change the field name `\Type` into `\ProType`), to add, at the top of `mail.dat`, the line

```
\Default\Type\FormLetter;
```

if there is one template, `\FormLetter`, or the line

```
\Default\Type\ProType;
```

if every old `\Type` has its own template, and to

```
\Filter mail.dat \Type\FormLetter
```

or to

```
\Filter mail.dat \Type\ProType
```

respectively. For detailed examples of form letters we refer to [1].

We conclude with an example that filters a  $\delta\alpha T$  file of `\Composers`. If there is a field `\TownOfBirth`, its contents refer to a  $\delta\alpha T$  file of towns, i.e. are of the form `*(a town)* *`. Every `\Town` has `\Default \Name \Ident;`, and a `\Country` with contents that are macros to be translated by a translation file:

```
\def\Town{\xdef\name{\Name}\WrapIn\cOuntry\Country}%
\def\Composer{%
\IfField \TownOfBirth \Exists
\TownOfBirth
\IfCs\cOuntry\IsDefinedAs{\Austria}%
{\bf\Name}, from \name, \cOuntry\par
\Fi
\Fi
```

```
}%
```

```
\Filter c:/dat/comp \Type\Composer
```

The result is a list of all composers in `c:/dat/comp.dat` that are born in Austria, with their towns of birth.

## References

- [1] A. Lenstra, S. Kliffen, R. Koning, and K. Aardal. Tips and tricks with  $\delta\alpha\TeX$ . *to appear*, 1995.
- [2] M. Piff. *Text merges in  $\TeX$  and  $\LaTeX$* . Taken from the file `textmerg.dtx` provided with the program source code, April 21, 1995.
- [3] K. van der Laan. Sorting in BLUe. *MAPS*, 10, 1992.



# Modifying L<sup>A</sup>T<sub>E</sub>X<sup>1</sup>

L<sup>A</sup>T<sub>E</sub>X3 Project Team

## Abstract

This is an updated version of a document that was first written to be part of the distribution of the new standard L<sup>A</sup>T<sub>E</sub>X. It was produced in response to suggestions that the modification and distribution conditions for the files in our system should be similar to those implied by Version 2 of the GNU General Public Licence, as published by the Free Software Foundation.

Although we are by now convinced that the principles described here are sound, the detailed consequences of these for the distribution and modification conditions are still evolving. Thus this article should not be treated as a definitive version of these conditions, even at the date of its publication.

## 1 Introduction

This article describes the principles underlying our policy on distribution and modification of the files comprising the L<sup>A</sup>T<sub>E</sub>X system. It has been produced as a result of detailed discussions of the issues involved in the support and maintenance of a widely distributed document processing system used by diverse people for many applications. These discussions have involved users, maintainers of installations that support L<sup>A</sup>T<sub>E</sub>X and various types of organisations that distribute it. The discussions are continuing and we hope that the ideas in this article will make a useful contribution to the debate.

Our aim is that L<sup>A</sup>T<sub>E</sub>X should be a system which can be trusted by users of all types to fulfill their needs. Such a system must be stable and well-maintained. This implies that it must be reasonably easy to maintain it (otherwise it will simply not get maintained at all). So here is a summary of our basic philosophy:

We believe that the freedom to rely on a widely-used standard for document interchange and formatting is as important as the freedom to experiment with the contents of files.

---

1. Copyright 1995; all rights reserved

We are therefore adopting a policy similar to that which Donald Knuth applies to modifications of the underlying T<sub>E</sub>X system: that certain files, together with their names, are part of the system and therefore the contents of these files should not be changed unless the following conditions are met:

- they are clearly marked as being no longer part of the standard system;
- the name of the file is changed.

## 2 The system

In developing this philosophy, and the consequent limitations on how modifications of the system should be carried out, we were heavily influenced by the following facts concerning the current widespread and wide-ranging uses of the L<sup>A</sup>T<sub>E</sub>X system.

1. L<sup>A</sup>T<sub>E</sub>X is not just a document processing system; it also defines a language for document exchange.
2. The standard document class files, and some other files, also define a particular formatting of a document.
3. The packages that we maintain define a particular document interface and, in some cases, particular formatting of parts of a document.
4. The interfaces between different parts of the L<sup>A</sup>T<sub>E</sub>X system are very complex and it is therefore very difficult to check that a change to one file does not affect the functionality of both that file and also other parts of the system not obviously connected to the file that has been changed.

This leads us to the general principle that:

with certain special exceptions, if you change the contents of a file then the changed version should have a different file name.

We certainly do not wish to prevent people from experimenting with the code in different ways and adapting it to their purposes. However, we are concerned that any distribution of modifications to the code should be very clearly identified as not being a part of the standard distribution. The exact wording and form of the distribution conditions is thus something that is flexible, but only within the constraint of keeping L<sup>A</sup>T<sub>E</sub>X as a standardised, reliable product for the purposes described above: the exchange and formatting of documents.

## 3 Some examples

Here we elaborate the arguments that have led us to the above conclusion.

*Separate development considered harmful!*

In many fields, the use of L<sup>A</sup>T<sub>E</sub>X as a language for communication is just as important as its capacity for fine typesetting; this is a very important consideration for a large population of authors, journal editors, archivists, etc.

Related to this issue of portability is the fact that the file names are part of the end-user syntax.

To take a real example, the L<sup>A</sup>T<sub>E</sub>X 'tools' collection contains the package 'array.sty'. A new user-level feature was added to this file at the end of 1994 and a document using this feature can contain the line:

```
\usepackage{array}[1994/10/16]
```

By supplying the optional argument, the document author is indicating that a version of the file `array.sty` dated no earlier than that date is required to run this document without error.

This feature would be totally worthless if we were to allow an alternative version of the array package to be distributed under the same name since it would mean that there would be in circulation files of a later date, but without the new feature. If the document were processed using this 'alternative array' then it would certainly produce 'undefined command' errors and would probably not be processable at all.

#### *What's in a file-name?*

In a pure markup language, such as SGML, it is reasonably clear that control over the final presentation lies with the receiver of a document and not with the author.

However, the way that L<sup>A</sup>T<sub>E</sub>X is often used in practice means that most people (at least when using the standard classes and packages) expect the formatting to be preserved when they send the document to another site.

For example, suppose, as is still the most common use of L<sup>A</sup>T<sub>E</sub>X in publishing, you produce a document for 'camera-ready-copy' using the class 'article' and that you carefully tune the formatting by, for example, adding some explicit line breaks etc, to ensure that it fits the 8 page limit set by the editor a journal or proceedings.

It then gets sent to the editor or a referee who, without anyone knowing, has a non-standard version of the class file 'article' and so it then runs to 9 pages. The consequence of this will, at the least, be a lot of wasted time whilst everyone involved works out what has gone wrong; it will probably also lead to everyone blaming each other for something which was in fact caused by a misguided distribution policy.

It should also be noted that, for most people, the version of the class file 'article' that gets used is decided by a site maintainer or the compilers of a CD-ROM distribution. To most users, the symbols `article` in:

```
\documentclass{article}
```

are just as much part of L<sup>A</sup>T<sub>E</sub>X's syntax as are the symbols `12pt` in:

```
\hspace{12pt}
```

Thus they should both define a standard formatting rather than sometimes producing 1 more page or a 5pt larger space.

Users rely on the fact that the command (or menu item) 'LaTeX' produces a completely standard L<sup>A</sup>T<sub>E</sub>X, including the fact that 'article' is the 'standard article'. They would not be at all happy if the person who installed and maintains L<sup>A</sup>T<sub>E</sub>X for them were

allowed to customise ‘article’ every second day so as (in her or his opinion) to improve the layout; or because another user wanted to write a document in a different language or typeset one with different fonts.

### *T<sub>E</sub>X itself*

We have modelled our policies on those of the T<sub>E</sub>X system since this has for some time now been widely acknowledged as a very stable and high quality typesetting system.

The distribution policy set up by Donald Knuth for T<sub>E</sub>X has the following features:

- There is a clearly specified method for changing parts of the software by the use of ‘change files’.
- Although arbitrary changes are allowed, the resulting program can be called T<sub>E</sub>X only if its functionality is precisely the same as that of T<sub>E</sub>X (i.e. neither less nor more) in all important areas.
- There are many files in the system that cannot be changed at all (without changing the name): examples are the file `plain.tex` and the files associated with fonts, including the Metafont source files.

### *Maintaining complexity*

Our experience of maintaining L<sup>A</sup>T<sub>E</sub>X has shown us just how complex are the interactions between different parts of the system.

We have therefore, with lots of help from the bug reports you send in, developed a large suite of test files which we run to check the effects of every change we make. A non-negligible percentage of these test runs give unexpected results and hence show up some unexpected dependency in the system.

## **4 Some assurances**

We are certainly not attempting to stop people reformatting L<sup>A</sup>T<sub>E</sub>X documents in any way they wish. There are many ways of customising incoming documents to your personal style that do not involve changing the contents of L<sup>A</sup>T<sub>E</sub>X’s standard files; indeed, this freedom is one of the system’s many advantages. The simplest way to achieve this is to replace

```
\documentclass{article} by \documentclass{myart}
```

Nor do we wish to discourage the production of new packages improving on the functionality or implementation of those we distribute. All we ask is that, in the best interests of all L<sup>A</sup>T<sub>E</sub>X users, you give your superbly improved class or package file some other name.

## 5 Configuration possibilities

The standard L<sup>A</sup>T<sub>E</sub>X system format can be configured in several ways to suit the needs and resources of an installation. For example, the loading of fonts and font tables can be customised to match the font shapes, families and encodings normally used in text mode. Also, by producing the appropriate font definition files, the font tables themselves can be set up to take advantage of the available fonts and sizes. The loading of hyphenation patterns can be adjusted to cover the languages used; this has to be done as part of making the format since this is the only stage at which patterns can be loaded.

A complete list of these configuration possibilities can be found in the distributed guide *Configuration options for L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>*  (`cfgguide.tex`). However, as it says there, the number of configuration possibilities is strictly limited; we hope that having read this far you will appreciate the reasons for this decision. One consequence of this is that there is no provision for a general purpose configuration file, or for adding extra code just before the `\dump` of the format file.

This was a deliberate decision and we hope that everyone (yes, that includes you!) will support its intent. Otherwise there will be a rapid return to the very situation, of several incompatible versions of L<sup>A</sup>T<sub>E</sub>X 2.09, that originally prompted us to produce L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub> : the new, and *only*, standard L<sup>A</sup>T<sub>E</sub>X. This will make L<sup>A</sup>T<sub>E</sub>X unmaintainable and, hence, unmaintained (by us, at least).

Therefore you should not misuse the configuration files or other parts of the distribution to produce non-standard versions of L<sup>A</sup>T<sub>E</sub>X.

## 6 Modification conditions

It is possible that you need to produce a version of L<sup>A</sup>T<sub>E</sub>X which is sufficiently distinct from standard L<sup>A</sup>T<sub>E</sub>X that it is not feasible to do this simply by using the configuration options we provide or by producing new classes and packages.

If you do produce such a version then, for the reasons described above, you should ensure that your version is clearly distinguished from standard L<sup>A</sup>T<sub>E</sub>X in every possible way, including the following.

- Ensure that it contains no file with a name the same as that of a file in the standard distribution but with different contents.
- Ensure that the method used to run your version is clearly distinct from that used to run standard L<sup>A</sup>T<sub>E</sub>X; e.g. by using a command name or menu entry that is clearly not `latex` (or `LaTeX` etc).
- Ensure that, when a file is being processed by your version, the use of non-standard L<sup>A</sup>T<sub>E</sub>X is clearly proclaimed to the user by whatever means is appropriate;
- Ensure that what is written at the beginning of the log file clearly shows that a non-standard L<sup>A</sup>T<sub>E</sub>X has been used.

## **7 What do you think?**

We are interested in your views on the issues raised in this document. The best way to let us know what you think, and to discuss your ideas with others, is to join the LaTeX-L mailing list and send your comments there. To subscribe to this list, mail to:

`listserv@vm.urz.uni-heidelberg.de`

the following one line message:

`subscribe LATEX-L <your-first-name> <your-second-name>`

## The proposed T<sub>E</sub>X Directory Structure

Joachim Schrod

The past has seen many discussions why T<sub>E</sub>X is conceived to be so difficult. Four years ago, at EuroT<sub>E</sub>X '91 in Paris, we even had a panel on "Why is T<sub>E</sub>X unusable?" A basic criticism that came up in almost all these discussions was

- a. the difficulty to install T<sub>E</sub>X and maintain the installed system afterwards,
- b. that there is no agreement what components belong to an installed T<sub>E</sub>X system, and
- c. that the structure of T<sub>E</sub>X installations is too different from site to site, thereby making it difficult to maintain a T<sub>E</sub>X installation.

Over the last 15 months, a TUG working group has been busy preparing a draft for a standard T<sub>E</sub>X Directory Structure (TDS). We hope to serve the T<sub>E</sub>X community by attacking item (c) mentioned above. In fact, when the draft is accepted, we hope that item (a), the difficulty to install and maintain T<sub>E</sub>X systems, will be reduced as well.

The TDS draft addresses primarily the T<sub>E</sub>X system administrator at a site and people preparing T<sub>E</sub>X distributions. It explicates where files of a package will reside in a final installation, thus making it easier for the administrator to find his or her way around. If someone is responsible for T<sub>E</sub>X installations on more than one platform, it will also reduce the needed time to find files as the structures will all be the same.

One T<sub>E</sub>X system can be used (e.g., via NFS mount or mounted from a CD-ROM) for both Unix-based workstations and DOS-based PCs, thereby reducing the maintenance time again. To support that aim, only the location of implementation-independent files are fixed; locations for implementation- and platform-dependent files are only recommended.

If developers of a package can assume a common directory structure, the package's installation can be automated, or at least the instructions can be made very specific. Last, but not least, many users will be interested in a defined installation structure, as they want to have a look at the system they are using.

The basic idea behind the TDS is that the files from a distributed package may fall in different categories: macro files for one (or even more) T<sub>E</sub>X formats, fonts and font metrics, auxiliary files for utility programs, etc. For each category, a package gets

assigned a set of directories where its files are placed. If more than one file exists for a category, a whole (exclusive) directory is allocated for that package. Otherwise this file is placed in a directory named `misc`.

When an update for such a package arrives, the current files in the assigned directories (or the one file in `misc`) may be thrown away and the new ones may be installed. (It's as – or even more – important to know which files to remove on update, as to know which files to install. Everybody who has maintained any system for some time has stumbled over that problem.)

This distribution of files over a directory tree implies that both  $\TeX$  ports and utility programs (like DVI drivers) must be able to search a file recursively in a directory tree. A survey among developers showed that most widely used  $\TeX$  software supports subdirectory searching already; other implementations will get it soon. Actually, the majority of developers were not willing to spend much work in sophisticated cache and search strategies, so the proposed layout pays attention to that restriction. As always, one had to make compromises.

Members of the working group are Barbara Beeton, Karl Berry, Vicki Brown, David Carlisle, Alan Jeffrey, Pierre MacKay, Rich Morin, Sebastian Rahtz, Joachim Schrod, Elizabeth Tachikawa, Ulrik Vieth, and Norman Walsh (chair). These members have either years of experience in maintaining  $\TeX$  systems or they are active in preparing distributions for important  $\TeX$  packages or they are engaged in the preparation of complete  $\TeX$  distributions (or all of these points). So we are reasonably confident that our proposal is not hot air; it is in use already and we hope that it will be utilized by all important  $\TeX$  distributions in the future.

The current TDS draft is available on any CTAN host, in various formats ( $\LaTeX$ , DVI, PostScript, etc.) It is placed in subdirectories of `/tex-archive/tds/`. Any feedback to that draft should be sent by email to `twg-tds@shsu.edu` or by paper mail to the chair of the working group (Norman Walsh, O'Reilly & Associates, Inc., 90 Sherman Street, Cambridge, MA 02140, USA).

# Occam's Razor and macro management

Laurent Siebenmann

Université de Paris-Sud  
Matématiques, Bâtiment 425  
F-91405, Orsay, France  
lcs@topo.matups.fr

## Abstract

The philosophical principle known as Occam's Razor asserts that entities should not be multiplied beyond necessity. The  $\TeX$  utility OCCAM is a tool to eliminate from a collection of supporting  $\TeX$  macros (composite commands) those that are unnecessary in a given typescript. Hopefully, it will serve to (a) let Plain  $\TeX$  users produce typescripts which can be electronically posted in a compact form that is nevertheless autonomous and perfectly archival, and (b) to simplify a macro package before making modifications for a special purpose.

The OCCAM utility will ultimately be programmed entirely in  $\TeX$  language to assure that it is universally available. Today it is just an evolving prototype implemented with a bit of help from an editor (on Macintosh) that has a programmable control language based on GREP.

To achieve reasonably *automatic* functioning of OCCAM, not requiring surveillance by a  $\TeX$  programmer, it is necessary to maintain a carefully structured master version of each macro package involved; this `.occ` version can double as the documented source version of the package.

*Keywords:* Occam's Razor, macro management, Plain  $\TeX$ , `.tex` typescripts, electronic publication, docstrip,  $\LaTeX$ .

## 1 The aphorism

At this European congress, let me remind you that the English philosopher William of Occam (or Ockham) was, like Abelard or Erasmus, a consummate European; he worked successively in Cambridge, Avignon, and Munich.

Occam's Razor is

*entia non sunt multiplicanda praeter necessitudinem*  
*entities should not be multiplied beyond necessity*

*William of Occam 1285-1349(?)*

Experts believe that Occam did not formulate it in exactly these famous words, but rather as

*What can be done with fewer assumptions*  
*is done in vain with more.*

or

*Plurality is not to be assumed without necessity.*

The Razor is sometimes called the ‘Principle of Parsimony’.

## 2 Introduction

Have you ever felt guilty about burdening a friend with macros that are not really necessary for composing your typescript? I certainly have; and would ideally like to follow Knuth’s example of using macro files that define exactly what is necessary for a document and nothing more.

However, pruning a macro file that has served for other purposes is a pain. Most of us respond to this pain by adopting a rather messy maximalist approach in which all the macros that have a genealogy related to the necessary macros are transmitted.

But there is another approach! One can seek efficient mechanisms to ease the task of weeding out unnecessary macros.

One such mechanism is `auditor.tex`, which makes a list of names of those macros of macro file that turn out to be unnecessary in a given typescript.

A complementary tool is DEFSTRIP. This utility exploits a specially arranged OCCAM version of the macro file to be cleaned up, in conjunction with the list of unused macros provided by `auditor.tex` in order to delete the macros listed (and some annexed material).

Ultimately, DEFSTRIP will hopefully be a `.tex` program `defstrip.tex` resembling the `docstrip.cmd` utility of L<sup>A</sup>T<sub>E</sub>X fame. Today, there exists only a QUED/M command script called “DefStrip-QUEDCmds”. (QUED/M is an inexpensive editor with convenient composite command capabilities, its own ‘macros’. It is available on Macintosh computers, distributed by Nisus Software Inc. of Solano Beach Calif.) Methods suitable for programming `defstrip.tex` are described in [2].

AUDITOR and DEFSTRIP together make up the system called OCCAM. Let us consider two classes of situation where the OCCAM system will be useful.

## 2.1 Weeding one's personal macro files

Many T<sub>E</sub>X users build up a cumulative personal macro file through composing many articles with T<sub>E</sub>X. A time inevitably comes when it is embarrassing, cumbersome, or confusing to submit (or post electronically) the whole macro file along with the article. The OCCAM system makes the pruning of the macro file painless. It is necessary to tidy up the total macro file and maintain it with 'OCCAM structure', which is usually distinguished by the extension `.occ`; this structure will be described in Section 4. Then, and only then, will `auditor.tex` and DEFSTRIP collaborate to *automatically* produce a minimal version of the macro file suitable for the article at hand.

## 2.2 Preparing autonomous and archival ".tex" postings

Suppose that one proposes to post in electronic `.tex` form an article prepared using a remarkable but not really standard package such as the `harvmac.tex` macro package of Paul Ginsparg for Plain T<sub>E</sub>X.<sup>1</sup> Such a macro package is not immune to alteration with time, and unfortunately the principles of upward compatibility are just pious hopes, not laws. Consequently, one is well advised to post, along with the `.tex` version, the macros necessary to compile it – especially if modifications to the macros have been used. Unfortunately, the `harvmac.tex` macros are as voluminous as a 10 page article. This is an unfortunate obstacle to electronic posting of shorter `.tex` typescripts.

The solution proposed requires `harvmac.occ`, which is `harvmac.tex` macros set out in a form designed for use with OCCAM. Then the necessary macros for a given article can be extracted by OCCAM. This is illustrated (in the OCCAM distribution) for a famous article by Edward Witten posted electronically in November 1994. The resulting archival posting (Plain based) requires only 6 Ko of macros rather than the original 20 Ko. The 68 Ko body of Witten's article is unmodified.

Recently `harvmac.tex` has been enhanced by inputting `hyperbasics.tex`, the hyper-reference macros of Tanmoy Bhattacharya, (and the new name is `lanlmac.tex`).

The archival nature of the T<sub>E</sub>X postings that have been minimized using OCCAM still depends on Knuth's Plain format being archival. Plain T<sub>E</sub>X will probably forever remain unchanged, or at least be upwardly compatible in the best sense. If this does not seem a sure bet to you, your posting could be made archival on the scale of the life of T<sub>E</sub>X by subjecting the Plain macros to standard OCCAM discipline to produce `plain.occ`. An article might require half the macros of `plain.occ` (or 20 Ko) to have its own format built from INITEX. This may seem needlessly radical to an English speaking user; however, for longer works in the many other languages that in any case require a special compilation from INITEX, I consider bootstrapping from INITEX the best approach to fully archival `.tex` postings.

---

1. The alternative `.dvi` form is undeniably convenient, but also less flexible. For instance, the `.tex` version can be redimensioned and then retypeset to be read in comfort on any computer screen whereas a `.dvi` version (or anything derived from it) often has lines too long for the viewer. Other output formats like `.ps`, being derived from the `.dvi`, are similarly inflexible.

There are many other macro packages that might benefit from OCCAM's `.occ` structuring. The 'picture' macro package embedded in  $\text{\LaTeX}$  is an example; interestingly these macros run on Plain. The `amssym.tex` math symbol definition package for  $\mathcal{AMS}$  fonts is another; it defines hundreds of control sequences, of which precious few are used in any given article. Both will be included in `.occ` form in the OCCAM distribution.

Would it be reasonable to convert  $\mathcal{AMS}\text{-}\text{\TeX}$  into a Plain macro package with `.occ` structuring, much as it has been converted by the  $\mathcal{AMS}$  into a documented  $\text{\LaTeX}$  package? This would be a move toward abandoning  $\mathcal{AMS}\text{-}\text{\TeX}$ 's status as a full-fledged format. In particular the  $\mathcal{AMS}\text{-}\text{Plain}$  package would have no influence outside of math mode.

In the long term, the structuring of a macro package for use with OCCAM will be the responsibility of the author of the package. Clearly such structuring will catch on only if OCCAM performs well as a *fully* portable  $\text{\TeX}$  utility.

### 3 How Auditor operates

The first utility, AUDITOR, of the OCCAM package is already a perfectly portable  $\text{\TeX}$  program `auditor.tex`. On a small scale, AUDITOR is independently useful, so it should help you learn about OCCAM interactively.

Suppose you have a Plain  $\text{\TeX}$  typescript `x.tex` that inputs a macro file `x.sty` whose size and history lead you to suspect it to be full of unused macros. AUDITOR serves to provide a list of macros defined in `x.sty` that are *not* used in `x.tex`.

The basic idea used by AUDITOR is easy to grasp if described in a simplified form as follows. Auditor changes the definition of many a new (top level) macro `\mymacro` in `x.sty` so that its expansion includes an 'auditing' device able to report whether `\mymacro` has been used in `x.tex`. If the expansion of `\mymacro` is originally `blablabla` then during use of AUDITOR it becomes:

```
\global\let\_mymacro\_@Used blablabla
```

Here `@Used` is a macro with some arbitrary expansion such as `@@Used`. With a bit of luck, a use of `\mymacro` with this new definition will cause an artificial macro whose name string is `_mymacro_` to become defined and have expansion `@@Used` – and do so without disturbing the normal functioning of `\mymacro`.<sup>2</sup>

The artificial macro `_mymacro_` can clearly be polled after typesetting `x.tex`; that will record whether `\mymacro` has been used. Of course, AUDITOR must somehow find out which macros `_mymacro_` should be polled. That is easy, and is done as follows, along with the above redefinition of `\mymacro`. Before the audit, `\def \mymacro` is replaced by `\Def \mymacro`; this `\Def` first stores `_mymacro_` in a token sequence for later polling, and then makes the modification of the expansion of `\mymacro` we have exhibited above.

---

2. Variants are possible. To identify macros that are little used, one could count how many times `\mymacro` is used.

We have seen the simple idea behind `auditor.tex`. Programmers will have also noticed that the idea can fail to work in unfortunate cases; we shall return to that.

### 3.1 How to make “audit.tex” list unused macros

Here now is a user-oriented recipe to get a list of unused macros – they can often be quickly eliminated by hand. Recall that the typescript is `x.tex` and its macro file is `x.sty`.

- make a copy `x.occ` of `x.sty`.
- temporarily have `x.tex \input x.occ` in place of `mflogo.dtx.sty`.
- at the top of `x.occ` add: `\input auditor.tex`
- wherever an ‘outer’ (top level) definition `\def\mymacro...` occurs in `x.occ`, replace `\def` by `\Def`. The latter is a special auditing version of `\def`, which is defined in `auditor.tex`.
- typeset `x.tex`; this will produce a file `audit.lst` containing a list of all the macros defined using `\Def`; in it the macros that are unused by `x.tex` are specially marked.

Using `audit.lst` to eliminate from `x.sty` the unused macros is sometimes tiresome to do by hand; the chief role of the `DEFSTRIP` utility described in the next section 4 is to automate this.

`AUDITOR` is a bit more general than indicated so far: `\Def` has a number of cohorts that behave similarly:

```

\Def          (variant of \def)
\gDef        (variant of \gdef or \global\def)
\Let         (variant of \let)
\gLet        (variant of \global\let)
\Mathchardef (variant of \mathchardef)
\Newsymbol   (variant of \newsymbol)

```

These replace corresponding uncapitalized control sequences in the file `x.occ`. Here, `\newsymbol` (from `amssym.def`) is a macro used copiously for declaration of symbols from the  $\mathcal{AMS}$  math fonts `msam*` and `msbm*`, as in `amssym.tex`. Note that `\mathchardef` and `\newsymbol` define a ‘mathchar’ not a macro; but the capitalized versions define macros.

As has been mentioned, `auditor.tex` is not bullet-proof. Any change whatever in the expansion of a macro can *in principle* alter its behavior. For example,  $\TeX$  can use `\ifx` and many other means to examine the expansion of a macro; a ‘perverse’ `x.tex` can always be constructed that stops compilation if there is any tampering with definitions.

However, if one exercises prudence this is *unlikely*. Here is some cautionary advice:

- Use `\Let` only with macros; e.g. `\Let\mymacro\thymacro` is allowable only when the control sequence `\thymacro` is a macro; the test command `\show\thymacro` will tell you if it really is one.
- Do not modify `\def`'s etc. within other definitions. This would often be pointless and perhaps dangerous. (But see section 5.)
- Avoid definitions involving “`\outer`” and “`\long`” macros; (But perhaps `\outerDef` and `\longDef` will be introduced to handle them.)

If there is trouble in using `x.occ`, then `opt` (by dialog) to compose *without* an audit. There should be no change from the original behavior of `x.tex`. Correct any misbehavior – often simply arising from a typing error in constructing `x.occ`. Sometimes, here and there in `x.occ`, one has to change `\Def` back to `\def` etc; (in that case the macro in question is clearly used!).

## 4 Occam structuring for macro packages, and the action of DefStrip

As described in the last section 3 above, the AUDITOR half of OCCAM uses both the macros `x.sty` and the typescript `x.tex` to establish a list `audit.lst` indicating macros unused in `x.tex`. OCCAM's second half, DEFSTRIP, serves to delete them from `x.sty` in a quite automatic way. The file `x.tex` is not further used but the specially structured version `x.occ` of `x.sty` is required. In addition, more structuring must be added to `x.occ` than was necessary for AUDITOR – the fully structured macro file is said to be OCCAM structured.

The goal of this section is to specify the syntax of the more basic OCCAM structuring, and indicate how DEFSTRIP should interpret it.

This structuring is less simple than the easy description of the action of AUDITOR in the last section 3 above might lead one to expect. It is true that DEFSTRIP ultimately merely serves to delete selected lines of the file `x.sty`. However, the result would be rather messy and less than minimal if only the lines occupied by the unused definitions were deleted. For example, one wants to delete comments attached to deleted macros, and possibly some auxiliary commands like `\newif...` not mentioned in `audit.lst`.

We now make this more precise through describing the syntax by which these blocks are unambiguously specified, and the rules for block deletion.

### 4.1 Main specifications of the OCCAM syntax

Two composite symbols `%^` and `%_` are employed in conjunction with `\Def` etc. to delimit possible deletions; the percentage sign `%` makes them invisible to  $\TeX$ . On its line, `%^` is always preceded by spaces only (zero or more); similarly `%_` is always followed by spaces only.

*Unconditionally deleted material*

```
%%^_ <delete me>
```

The block of lines from %%^\_ to the end of file is then deleted. To delete just a segment use

```
%^ <delete me> %_
```

The block of  $\geq 1$  lines deleted must include no blank line. Note that it may well contain \Def etc. but not %^, %\_.

The unconditional deletions will occur as if done in the order described, and before conditional deletions (described below) are considered.

*Conditionally deleted material*

```
\Def \somemacro ...
      ...%_
```

may cause deletion of the block of lines beginning with \Def etc. and ending with %\_. This material is really deleted, precisely if the macro \somemacro is marked for deletion in the the file audit.lst.

The material <maybe delete me> must contain no blank line nor %^, %\_, \Def etc; but it is otherwise arbitrary; in particular, macro arguments, comments, and auxiliary definitions are permissible.

Along with this material some additional preceding material is deleted, namely contiguous preceding lines (if any) that (a) are nonempty and (b) contain no %\_ (but \Def etc; are allowed). Typically, such preceding material might be comments or commands 'owned' by the macro being deleted. For example the whole block

```
%_
\ifx\undefined\eightpoint
  \Def\eightpoint{}
\fi %_
```

will be deleted, precisely in case \eightpoint is marked as unused in audit.lst. (The first %\_ could be replaced by a blank line.)

Note that %\_ is not really a closing delimiter since it can exist in arbitrary numbers without belonging to a matching pair. For another example, consider:

```
\Def\amacro ...%_
\newtoks\btoks %_
\Def\cmacro ...%_
```

Here, the the first two %\_ prevent \newtoks\btoks being deleted – in all circumstances.

The example

```
\Def\amacro ...
```

```
\Def\bmacro ...%_
```

is incorrect because the block beginning with `\Def\bmacro ...` contains `\Def\bmacro`.

### Sentinels

There is a second type of conditional deletion. Suppose `\amacro` is not used and is so designated in `audit.lst`. It often occurs that several *disjoint* blocks of lines should be deleted along with the block containing the definition of `\amacro`. These blocks should each be designated as follows:

```
%/\amacro
  <stuff>
%/_
```

`\amacro` is called the *sentinel (watchman)* for the block. The sentinel's line `%/\amacro` must contain nothing more than `%/\amacro` and blank space. The initial and terminal lines will vanish along with `<stuff>`.

### Summary of primary deletions by DefStrip

Any block `%^...%_` is unconditionally deleted, while a block signalled by `\Def`, `\gDef`, etc. with the help of `%_` and/or blank lines is deleted or not according as the macro following `\Def` etc. is or is not marked for deletion in `audit.lst`. Similarly for blocks with sentinel macro. None of these blocks for conditional or unconditional deletion is allowed to contain an empty line nor any extraneous `%^`, `%_`, `%/\amacro`, `%/_`, `%%^_`, `\Def`, `\gDef`, etc. The blocks introduced by `\Def`, `\gDef`, etc. include material extending backward as far as (but not including) a preceding line that is blank or terminated by one of `%_`, `%/_`. No such extension for blocks introduced by `%^`, `%/\amacro` is allowed – nor would it be helpful.

### Secondary cleanup by DefStrip

Beyond these primary deletions, the utility DEFSTRIP performs a few auxiliary tasks:

- All remaining `\Def`, `\gDef`, etc. are converted to `\def`, `\global\def`, etc. Also, if a remaining `%_` is alone on its line (spaces ignored), the whole line disappears. And each remaining `%_` *not* alone on its line becomes `%` (this is the only deletion that can affect a line that survives, and `TEX` is unaffected).
- Any empty line sequence (usually created by the deletion of blocks of lines) is reduced to a single empty line.
- Residual appearances in `x.occ` of macros marked for deletion in `audit.lst` will be marked by `%%[VESTIGE]` (on a new following line).

Such an occurrence of `%%[VESTIGE]` should be considered an error warning concerning the OCCAM structuring of `x.occ`. Users may find such vestiges hard to deal with. Thus the programmer should enquire whether (for example) the vestige could be automatically

deleted using the sentinel mechanism. For their part, users should report vestiges to the programmers along with the involved `audit.1st` file from `auditor.tex`.

In most cases, anyone who programs  $\TeX$  macros at an intermediate level will find it an easy task to provide OCCAM structuring for any simple macro file. However, as soon as the macro file has interdependent macros, and definitions of control sequences other than macros<sup>3</sup>, attention from a programmer will be needed – plus testing.

The author of the macro package himself is the most appropriate person to introduce Occam structuring:

- The author has the opportunity to add extra documentation to the `.occ` version and make of it a fully documented master copy of the package. Using conditional deletions, one has flexible control of which documentation is passed on by DEFSTRIP to the user.
- The author can often remodel the macro package to allow OCCAM to do a better job more simply.
- The author protects his work against the erosion of time; it becomes unimportant that large parts of a package become antiquated; only the parts that remain viable will be exported by OCCAM into users typescripts.
- Once committed to OCCAM (or similar means of macro distillation) the author can flout the usual rules of upward compatibility, provided it is made clear from the outset that the package is to be used exclusively to produce autonomous typescripts.

## 5 Nested macros

Unused macros whose definitions are nested within those of other macros that *are* used can often be eliminated, although that would be very difficult on the basis of features of OCCAM described thus far.

We now describe suitable additional syntax. It was implemented in 1995 through modifying both `auditor.tex` and DEFSTRIP. This is probably more subject to change than features in earlier sections.

Where DEFSTRIP is concerned, the new syntax is currently implemented quite trivially by making several passes, and the example below is conveniently explained in this way. However, the  $\TeX$  version `defstrip.tex` will almost certainly reduce this to a single pass; the `audit.tex` utility already acts in a single pass.

Here is a generic 'example'. The original macro file contains:

```
\def\MACRO{<stuff1>%
<stuff2>
\def\macro{<stuff3>}%
<stuff4>%
<stuff5>}
```

---

3. For example, unused fonts are hard to eliminate.

An OCCAM structured version is:

```
\Def\MACRO{<stuff1>%#_
<stuff2>
\DDef\macro{<stuff3>}%
<stuff4>%#_
<stuff5>}%_
```

The macro `\DDef`, defined in `audit.tex`, behaves much like `\Def` except that the associated sign distinguishing *unused* macros in `audit.lst` is `*#` in place of `*`.

Note that if `\MACRO` is *unused* then the whole block vanishes.

We are interested in the case where `\MACRO` is *used*. Then, on first pass of the macro file through `DEFSTRIP`, `\DDef` is converted to `\Def`; and the marks `%#_` therein are converted to `%_`. At the same time, the file `audit.lst` undergoes one wave of changes `*#\ ⇒ #*\` and `*\ ⇒ #\`, i.e. asterisks move right or die on backslash. On the second pass through `DEFSTRIP`, one is treating:

```
\def\MACRO{<stuff1>%_
<stuff2>
\Def\macro{<stuff3>}%
<stuff4>%_
<stuff5>}%
```

and, in response to an entry `#*\macro` in the current `audit.lst`, `DEFSTRIP` will delete the block

```
<stuff2>
\Def\macro{<stuff3>}%
<stuff4>%_
```

i.e. this block is deleted precisely if `\macro` is *unused*. (Only a programmer can guess whether this elimination is safe!)

### 5.1 Nested Sentinels

The macros in such nested definitions are allowed to be *sentinels* for blocks, as follows. Often, one wants to delete other material along with the block surrounding `\macro`. The syntax for a block to be eliminated along with `\macro` is:

```
%#/\macro
<stuff6>
%#/_
```

It is permissible to use `_` in place of `/_` on the above syntax. But not `^` in place of `/^` since that would give an unconditional deletion.

## 5.2 Nested unconditional deletions

There is also a notion of nested *unconditional* deletion. The syntax is:

```
%#^
<stuff6>
%#_
```

The developments of this section first proved desirable in `harvmac.occ`, which is a 'worked example' in the OCCAM package.

Until the T<sub>E</sub>X program `defstrip.tex` is built, I would particularly welcome suggestions for improvements of OCCAM structuring.

## 6 Afterthoughts

### 6.1 Is Occam's Razor now dull?

Occam's Razor was one of the guiding principles of scientific thought for several hundred years before the coming of age of computers. However, I suspect the philosophy of Aristotle or Descartes is far more likely to appeal to computer scientists. One might go so far as to say that programmers have discarded Occam's Razor. Indeed, in object oriented programming they consciously cultivate the art of multiplication of entities, and even L<sup>A</sup>T<sub>E</sub>X users do this sort of thing with commands such as `\newheading`. What can the minimalism of Occam's Razor offer T<sub>E</sub>X users at this late date? Probably just a few things.

- *Friendliness to human beings.* Unnecessary entities that cost a microprocessor only microseconds can cost the human mind a significant amount of time.
- *Extra storage space and computing power.* Both are in a period of exponential growth. But so is the T<sub>E</sub>X related software we use. Thus performance in a fixed task can occasionally decline. When this happens, the old-fashioned minimalism of Occam's Razor can prove essential to derive pleasure and profit from progress.

### 6.2 Plain versus L<sup>A</sup>T<sub>E</sub>X

The goals of OCCAM do not make much sense in the L<sup>A</sup>T<sub>E</sub>X world. The L<sup>A</sup>T<sub>E</sub>X group is building official L<sup>A</sup>T<sub>E</sub>X macro modules that cover more and more territory, and are universally available. If upward compatibility is fully maintained, there will be little reason to use OCCAM in the everyday L<sup>A</sup>T<sub>E</sub>X world since all macros used will be standard. This is L<sup>A</sup>T<sub>E</sub>X's greatest strength. It assures L<sup>A</sup>T<sub>E</sub>X an important and possibly dominant role in electronic scientific publication based on `.tex` typescript format. The role OCCAM seeks to play in this sort of electronic publication lies mostly within the realm of Plain T<sub>E</sub>X and INITEX.

However, I have been alarmed by the growing difficulties facing an individual when programming  $\text{\LaTeX}$  beyond its current capabilities – I mean more than routine reparameterizing and renaming. It is not just  $\text{\LaTeX}$ 's exponentially growing internal complexity that is alarming, nor the fact that  $\text{\LaTeX}$  runs slower and slower relative to Plain. The most debilitating problem is that internal macros should probably not be reprogrammed since (unlike the user macros) they are open to change. Thus the results of such individual programming efforts risk being electronically 'unpublishable' even if the macros involved are posted along with the article they produce – indeed the version of  $\text{\LaTeX}$ 's internal macros on which they were based may vanish from the next  $\text{\LaTeX}$  update, and that may well invalidate the posting. A related fundamental difficulty is that  $\text{\TeX}$  primitives are not guaranteed to be/remain accessible from within  $\text{\LaTeX}$ . The logical conclusion would seem to be that all programming of  $\text{\LaTeX}$  sufficiently deep to modify  $\text{\LaTeX}$  internals should be left to the official  $\text{\LaTeX}$  group.

Don Knuth seems to have had similar premonitions and takes a possibly more doubtful attitude. I repeat what he said on the occasion of the 10th anniversary celebration of  $\text{\TeX}$  82.

Suppose you were allowed to rewrite all the world's literature; should you try to put it all into the same format? I doubt it. I tend to think such unification is a dream that's not going to work.

[TuGboat, vol 13 (1992), page 424]

I (and Knuth?) may be unduly alarmed. OCCAM is nevertheless to some extent my attempt to stop the decline of Plain. (There is nothing significant I know how to do for this real or imaginary malady of  $\text{\LaTeX}$ .)<sup>4</sup>

$\text{\LaTeX}$  continues to perform well for an expanding palette of routine tasks, and I am happily using it to prepare this article! The most favorable outcome would be for both approaches to work well. If this comes about, I expect the "look and feel" of Plain and  $\text{\LaTeX}$  to nevertheless steadily diverge.

In summary, the  $\text{\TeX}$  utility OCCAM offers a novel response to the most debilitating problem of Plain  $\text{\TeX}$ , namely the confusion and incoherence that come from continual and uncoordinated accretion of macros. Naturally, this weakness of Plain is clearly perceived by Leslie Lamport who stated (on the net in 1994):

Because Plain  $\text{\TeX}$  is fixed, it seems likely that the Plain  $\text{\TeX}$  community will fragment into numerous small islands in a sea of incompatibility.

The  $\text{\LaTeX}$  programming group deals with the accretion of macros by constantly improving infrastructure while selectively enlarging  $\text{\LaTeX}$ , whereas my partial answer for Plain  $\text{\TeX}$  is to use OCCAM to cull out inessential macros, restoring simplicity in the macro

---

4. An author who absolutely needs many of the basic talents of  $\text{\LaTeX}$  and new performance, might consider using a frozen but hopefully permanent format like  $\text{\LaTeX}$  or 'classic'  $\text{\LaTeX}$  2.09 and macros distilled by OCCAM from personal or public packages whose permanence is in doubt.

set actually used in each document, and thereby making new infrastructure and standardization largely unnecessary. OCCAM's action has a parallel in classical programming, namely the use of a compiler – whereas the L<sup>A</sup>T<sub>E</sub>X approach is parallel to the use of a big and constantly evolving interpreter.

My hope is that, with OCCAM's help, Plain T<sub>E</sub>X will regain vigor and prove as viable as L<sup>A</sup>T<sub>E</sub>X, and indeed complementary to it.

## References

- [1] L. Siebenmann. Elementary text processing and parsing in T<sub>E</sub>X – the appreciation of tokens. *TUGboat*, 13:62–73, 1992.
- [2] L. Siebenmann. *OCCAM, a T<sub>E</sub>X utility*. the electronic master posting in 1995 is on <ftp://matups.math.u-psud.fr//the CTAN archives>, 1993-5.



# A package for Church-Slavonic typesetting

Andrey Slepuhin

pooh@shade.msu.ru

## 1 Introduction

The multilingual ability of T<sub>E</sub>X is one of its most important properties. Due to T<sub>E</sub>X it became possible to produce high-quality books in many different languages (sometimes with very exotic grammatic rules). For more than 10 years of its existence T<sub>E</sub>X became a real polyglot and it seems that it doesn't want to stop evaluating. In this paper one more, may be rather exotic, example of practical usage of T<sub>E</sub>X is considered, and also many ideas and solutions which result from 5-year experience of T<sub>E</sub>X using.

## 2 General solutions

How does a language-specific package have to look like from the point of view of a computer publishing system? It must include at least the following components:

- quality fonts;
- tools for simplifying the text formatting;
- hyphenation table;
- punctuation or some other poligraphic rule description;

These requirements became a basis to C<sub>h</sub>TeX development. The two first items got quite satisfactory realization. As to the realization of the third one – it depends on the volume of the dictionary, which is not sufficiently complete yet. The fourth item is absent because the Church-Slavonic language has no precise rules of punctuation or whatever similar things.

## 3 Fonts

Designing the quality-fonts is, in general, a very hard task and moreover the author's knowledge on this subject at the beginning of this work were minimal. So, the designing

of the base version of fonts took more than half a year, and different improvements are still under development. Among the factors that made the work more complicated, a large number of symbols (only letters – 44) in the Church-Slavonic alphabet should be noted. Also the glyphs of symbols have a very few similar elements. The typeface, which had a wide spreading at the beginning of the XX century, was taken as a model of created fonts. The following technology was applied for the fonts developing: the symbols were magnified and separate elements were extracted, then base and control points of outline curves were placed manually and the METAFONT macros were designed; the obtained symbols were finally improved using the METAFONT graphic output.

#### 4 The diacritical signs problem

The main problem, that occurred during the  $\TeX$  development, was connected with the fact, that every word in a Church-Slavonic texts has at least one diacritical sign. None of computer publishing systems (except  $\TeX$ , of course), known to the author, contains any convenient tools for typesetting a text with accents.  $\TeX$  uses `\accent` macro for this purpose, but this macro seems to be designed for rather rare usage, because it gives the following undesirable effects:

- the kern between accented and previous symbols disappear;
- $\TeX$  doesn't make any hyphens in the remainder of the word after accented symbol and can make invalid hyphens in the initial part of the word;

These effects are arising because  $\TeX$  uses explicit kern while expanding `\accent` macro. So, it seems, that the best solution of the diacritical signs problem (realized for many European languages, for example) is a method, when a letter together with an accent is represented by a single character in the font. However, in the case of the Church-Slavonic language this solution cannot be applied in proper form, because there are too many possible pairs 'letter – accent', and a limit of 256 symbols will be exhausted.

It would be wonderful if the following idea works: several pairs 'letter – accent' have positions equal modulo 256, and their metrics are identical. Unfortunately, it's impossible to force  $\TeX$  to put a symbol with character code greater than 256 into DVI-file. Such restriction is especially misunderstanding, because the DVI-file format supports the usage of symbols with character codes up to  $2^{32} - 1$ . One more well-known method to deal with the accents is their realization as strongly shifted left characters of zero width. Such a variant is unsatisfactory too, because it does not solve the kerning problem and significantly complicate the hyphenation table constructing.

To solve the accent problem we need to understand how the diacritical signs in Church-Slavonic language are placed. It can be found, that some of them can be placed only over the first letter in the word, and some can be placed only over the last letter. These two cases are realized by special macros `\fcaccent` and `\lcaccent`. The last macro can be written in a very simple way, because it needs only to locate the accent

with the help of kerns. The macro `\fcaccent` has `\nobreak\hskip0pt` construction in addition, which enables the hyphenation of the word after the diacritical sign.

As for the accents in the middle of a word, some of them are realized together with the corresponding letters, and other are representing symbols, used, in general, for abbreviation of certain words (in Church-Slavonic language they are called 'titlo'). The words, containing these symbols, as a rule, cannot be hyphenated, so it became possible to write a special macro, placing an accent and preserving kern both before and after the symbol. It is a quite sophisticated macro, which use such powerful T<sub>E</sub>X tool as `\futurelet`. The text of this macro is given below:

```
\def\ccaccent#1#2{%
#2\setbox2=\hbox{#2}\setbox1=\hbox{#1}%
\dimen0=\ht2\advance\dimen0 by -1ex%
\dimen1=\wd1\advance\dimen1 by \wd2%
\divide\dimen1 by 2%
\kern-\dimen1\raise\dimen0\hbox{#1}%
\advance\dimen1 by -\wd1%
\kern\dimen1%
\def\tmp{\explkern{#2}\next@}%
\futurelet\next@\tmp%
}%
```

The `\explkern` macro simply adds the kern, that must be placed between its arguments:

```
\def\explkern#1#2{%
\def\next@{ }%
\ifcat#2a%
\explkern@#1#2\else%
\ifcat#2.%
\explkern@#1#2\else%
\ifx#2\-%
\explkern@#1#2\else%
\fi\fi\fi%
}%
\def\explkern@#1#2{%
\setbox0=\hbox{#1#2}%
\setbox1=\hbox{{#1}{#2}}%
\dimen1=\wd0\advance\dimen1by-\wd1%
\kern\dimen1%
}%
```

For the convenience of the text typesetting, the symbols ' , " , ' , ~ , \_ , | and < are made active and are expanded to the corresponding macros. The selection of Church-Slavonic

mode is realized by the `\beginslav` macro, and return to usual mode is realized by the `\endslav` macro.

## 5 Slide making

Another problem, that come into consideration during package development is the problem of slide making. Almost all Church-Slavonic texts are two-colored. For the implementation of the color separation and for obtaining separate slides for each color, the `SlitEX`'s idea of using 'invisible' fonts was applied. However, kerning problems make impossible the usage of pure `SlitEX`. Indeed, having a word with a first letter emphasized by another color (in Church-Slavonic texts it occurs very often), `SlitEX` loses the required kern between the first letter and the remainder of the word when switching to another font. So, for such cases we need special macros. To implement the color separation a special font selection scheme was designed, slightly similar to NFSS. After including the font description file and appropriate macros, user can declare usage of any color via the macro `\newcolor<color>`. This macro induces the macros `\<color>g{<any text>}` and `\<color>`. The first of them switches the color, preserving kern, and the second switches it without preserving any implicit kern (`TEX` interprets this macro in the simplest way, so its usage is approved). Now, typing `\showcolor<color>` or `\hidecolor<color>` in the input file, we can make any selection by a specified color visible or invisible in output. The text before the first usage of `\<color>g{<any text>}` or `\<color>` will be always visible.

## 6 Numeration

In Church-Slavonic language a literal numeration is accepted, which can be described by the following algorithm:

Given an integer  $n \geq 0$ . Let  $S(n)$  be its representation in Church-Slavonic language. See also Table 1.

The representation of zero is absent in Church-Slavonic language, but let it be empty for conveniency.

If  $10 \leq n < 20$ , then  $S(n) = S(n \bmod 10)S(10)$ . If  $20 \leq n < 100$ , then  $S(n) = S(n - (n \bmod 10))S(n \bmod 10)$ . If  $100 \leq n < 1000$ , then  $S(n) = S(n - (n \bmod 100))S(n \bmod 100)$ . If  $1000 \leq n < 10000$ , then  $S(n) = S(n - (n \bmod 1000))S(n \bmod 1000)$ .

There are disagreements about representation of numbers greater than 9999, and by this reason it is not implemented yet. The macro `\slnum<number>` automatically generates the number representation in the Church-Slavonic language. For example, `\slnum(1995)` gives `Ⲡⲩⲱⲓⲉ`. One would be careful, because this macro is valid only in Church-Slavonic mode.

n	$S(n)$	n	$S(n)$	n	$S(n)$
1	ā	10	ī	100	ř
2	ķ	20	ķ	200	ř
3	ř	30	ā	300	ř
4	ā	40	ā	400	ř
5	ē	50	ī	500	ř
6	š	60	š	600	ř
7	š	70	ō	700	ř
8	ī	80	ī	800	ř
9	ā	90	ī	900	ř

Table 1: Numeration in Church-Slavonic language

## 7 T<sub>E</sub>X without encoding

During the work on  $\text{\textcircled{C}}\text{\textcircled{A}}\text{\textcircled{I}}\text{\textcircled{K}}\text{\textcircled{T}}\text{\textcircled{E}}\text{\textcircled{X}}$  an idea appeared which allows to solve the compatibility problem while transferring any package to another platform. This problem is especially actual in Russia, because Russian letter encodings on different platforms do not coincide. A version of T<sub>E</sub>X cyrillisation made by CyrTUG is specific for PC-compatible computers under MS-DOS. It causes, in particular, the disgust of numerous UNIX users in big research institutes, which need T<sub>E</sub>X most of all.

The idea of easy transferring any T<sub>E</sub>X package to different platforms is given below:

- The encoding table containing a map between character codes and their symbolic names (for example, like PostScript names) must be defined for each specific platform and font family.
- The certain utility must be written (it can be done even by T<sub>E</sub>X!) which generates two files: T<sub>E</sub>X encoding table and METAFONT encoding table, from the original one.
- The set of METAFONT macros must be added to redefine `beginchar` macro; it must allow the usage of symbolic names instead of character codes by declaring `usenames:=1` or whatever like this.
- Hyphenation table must be written, using the symbolic names; when generating base file, firstly T<sub>E</sub>X should read encoding, then it should convert original hyphenation table into temporary file using current encoding and then it should read the file obtained.
- `\catcode`, `\lccode` and `\uccode` should be defined using symbolic names.

This idea is implemented in the last version of the package represented and is now in the process of testing. It should be hoped that new CyrTUG's cyrillisation versions will be written in the form described above. It would facilitate the work for many T<sub>E</sub>X users in Russia and for people who need to typeset Russian (or other Cyrillic) texts.

## 8 Type 1 from Metafont?

The one more idea, implemented as a part of  $\text{\LaTeX}$  project, came from the article[3]. Its realization was forced by appearing a PostScript-printer on the author's table. There was written a set of METAFONT macros which allow to obtain a text representation of Type 1 fonts from METAFONT sources and, furthermore, a downloadable font by L. Hetherington's Type 1 utilities. This macro package initially was designed to solve the specific problem of representation Church-Slavonic in Type 1 format, but the conversion of Computer Modern fonts (and others) is also possible. This work requires a special consideration and is not described in the paper.

## 9 Problems and plans

The most important of the update problems is the adaptation of  $\text{\LaTeX}$  package to  $\text{\LaTeX}2\epsilon$ . When this paper was being written, the author had the distribution of  $\text{\LaTeX}2\epsilon$  for a month, and this distribution was not installed due to the lack of the time. Another problem is connected with the fact that the current font version contains only symbols of modern Church-Slavonic language, whereas symbols from ancient versions of language are often needed. The author also plans to develop a package for typesetting music in non-linear notation (so-called 'krjuki') being in use before XVIIIth century. The following problems connected to Church-Slavonic typesetting also would be noted: designing a font of initial caps and a special font for headings. In this font different combinations of letters must have specific glyphs (this task seems to be a little fantastic, because such font should have a monstrous number of symbols and ligatures).

It would be hoped that somebody shares the author's interest in the problem of Church-Slavonic and ancient texts. May be sometime a multilingual edition of Bible (in Church-Slavonic, Greek, Latin, Hebrew ... what else?) made by  $\text{\TeX}$  come in appearance.

## 10 Examples

A simple example of a Church-Slavonic text:

```
г|сди <i_исе х|срт'е, с_не б_жій,  
пом'илуй м'я гр'ешнаго
```

and the result of its compilation:

ГѠи ѡсе хрѣте, сѡе вѣѡй, помѡлѡи мѡ грѣшнаго

An example of color separation: a sequence of macros

```
\beginslav\family(slav)\size(12)%
\def\pray{%
\redg Г|с{ди} <i_исе х|срт'е, с_не
б_жій, пом'илуй м'я гр'ешнаго
}%
\black%
\showcolor(red)%
\par\noindent\pray
\hidecolor(red)%
\showcolor(black)%
\par\noindent\pray
\showcolor(red)%
\par\noindent\pray
\endslav
```

gives the result

Г  
ГѠи ѡсе хрѣте, сѡе вѣѡй, помѡлѡи мѡ грѣшнаго  
ГѠи ѡсе хрѣте, сѡе вѣѡй, помѡлѡи мѡ грѣшнаго

This example shows that accents can be placed over a group of symbols, not only over a single symbol.

## References

- [1] Donald E. Knuth. *The T<sub>E</sub>Xbook*. Addison Wesley, Reading, MA, 1990.
- [2] Donald E. Knuth. *The METAFONTbook*. Addison Wesley, Reading, MA, 1990.
- [3] Bogusław Jackowski, Marek Ryćko. Labyrinth of METAFONT paths in outline. *EuroT<sub>E</sub>X Proceedings, 1994: 18–32*.
- [4] Ieromonakh Alipiy (Gamanovich). *Grammatika tserkovno-slavjanskogo jazyka*. Palomnik, Moscow, 1991.
- [5] *Slovar' russkogo jazyka XI–XVII vv.* Nauka, Moscow, 1975.



# The W95 environment

Antonín Strejc

strejc@vc.cvut.cz

## Abstract

Since 1992 'WORKSHOP 9x' has taken place at the Czech Technical University (CTU) each year. The aim of this broadly-based seminar is to give all CTU researchers and research teams an opportunity to present their research projects in twenty minutes of spoken presentation and two pages of seminar proceedings. I have dealt with the technical problem of making the proceedings using L<sup>A</sup>T<sub>E</sub>X. As the number of contributions has increased year by year and the time for making the book is limited, some automation of the typesetting process was and still is necessary. W94 and W95 are attempts to transfer part of the typesetting work from the final typesetter to the authors. W9x is a simple single-purpose user-interface between L<sup>A</sup>T<sub>E</sub>X and the MS-DOS user, who may know nothing about T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X. Some experience (both technical and psychological) of using this system in the two last years is discussed in this paper and may be useful for organizers of seminars, conferences etc. where contributors are not T<sub>E</sub>X users and the proceedings are to be made with T<sub>E</sub>X.

## 1 Introduction – the history

The Czech Republic has passed through a great political, economic and social transformation process in the last five years. Of course, great changes have also affected the CTU in Prague. The old system of state support for education and research, based on long-term central planning has changed into a grant system based on competition between research projects. At present, 607 research and educational projects are supported by grants from various Czech and foreign grant agencies.

More and more projects have brought more and more need for publishing the results of research work. In 1991, the first ideas of establishing an annual broadly-based university seminar – Workshop 9x – appeared at the CTU. Workshop 92 was the first in the series and took place at the CTU in January 1992 and subsequent Workshops have been held regularly every year. The seminar gives an opportunity to present research projects in spoken form (20 minutes) and in two pages of proceedings.

In 1991, I was given a job at the CTU Computing Centre, where a small group of young people working with T<sub>E</sub>X was set up. The Centre was asked to co-operate with

the Workshop 92 organizing committee and the small group to make the proceedings with  $\text{\LaTeX}$ .

As the conditions have changed rapidly year by year (see Table 1), it has been necessary to find and improve our methods of work. Our experience is described in the following sections.

Year	Number of contributions	Number of pages	Number of typesetters	Time for completion	pg/man per day
	Method of work				
1992	210	484	4	30 days	4.03
	Manual conversion from text editor				
1993	293	662	3	30 days	7.35
	Conversion from text editor, partially automated				
1994	303	688	2	20 days	17.2
	W94 Environment				
1995	436	928	2	15 days	30.9
	W95 Environment				

Table 1: Conditions, elapsed time and methods of work

### 1.1 1992 – no experience, great confusion...

A desperate situation... In 1991 someone announces a seminar for which contributions are to be word processed. Even this fact was quite a shock for many people at a time when PC 386 SX was a great hit for a small number of lucky people. Many department offices still preferred typewriters and many people were still starting to learn how to use a personal computer. Most computers were XT's and 286's.

Authors were asked to prepare their contributions with the simple text editor Text602, which is a WordStar-type editor without mathematical fonts. Mathematical equations were therefore written by hand into empty spaces in the text and camera-ready figures were enclosed. The organizing committee collected all contributions (diskettes, Text602 prints and enclosures) and this complete pile of papers was given to us. First we made a  $\text{\LaTeX}$  style file and started to convert ASCII text files into the source form that  $\text{\LaTeX}$  needs. Reading and typesetting mathematical formulas was particularly hard work. Very often it was impossible to find out what the formula should have been and it was necessary to consult with the authors (by phone or in person).

As we had no experience we did not know how to organize our work, how to sort the contributions, where and how to store the ready prints and files, how to divide the work optimally among a number of people, etc., etc. Due to these problems we were very short of typographic time and so the final typographic standard of the proceedings

was low, though much higher than it would have been in the case of direct processing of the Text602 prints. The proceedings were therefore well received.

## **1.2 1993 – first rules of the game**

A year later I became the leading typesetter in our group and also a member of the organizing committee of Workshop 93. I had to bear all the responsibility for making the proceedings. My situation was a bit easier as I had about six months in which to prepare all the strategy.

I started by analyzing the previous year's proceedings, and I found the structure of all the contributions quite similar. At that time I got the idea to turn this similarity into an identity of structure, to set up an obligatory structure for contributions. I supposed that text files of this obligatory structure could be pre-processed with some tool (a short Pascal program). I wrote this tool to read data from some known parts of the file and to add some formatting commands to them. I anticipated that this would save much typesetting time.

This involved telling authors (very exactly) what and in which way I wanted them to write. So I wrote detailed 'Instructions for Authors' in which the following sentences appeared:

'The first line of your text file should contain the title of your article. Use all upper-case letters. Then leave one blank line. The next line should contain the initial letter of your first name, a point, one space and your surname. Then leave a line again. Then... etc., etc.' These detailed instructions were intended to ensure the output uniformity of all articles in the proceedings.

The output should have looked something like Figure 1. Seven obligatory parts can be seen in this fictitious example of an article:

- the title
- names of the authors
- addresses of the authors
- key words
- main article body
- references
- information about grant support

This structure became a standard not only for Workshop 93 but also for subsequent years. This is the basis from which the idea of the W94 and W95 environment has grown, as we will see in the following sections.

In 1993 the processing was much easier than it had been a year before, though not everything went according to my plans. The problem was that most authors did not follow my instructions exactly. This led to the fact that most files could not be pre-processed directly.

All the file-by-file processing had the following stages:

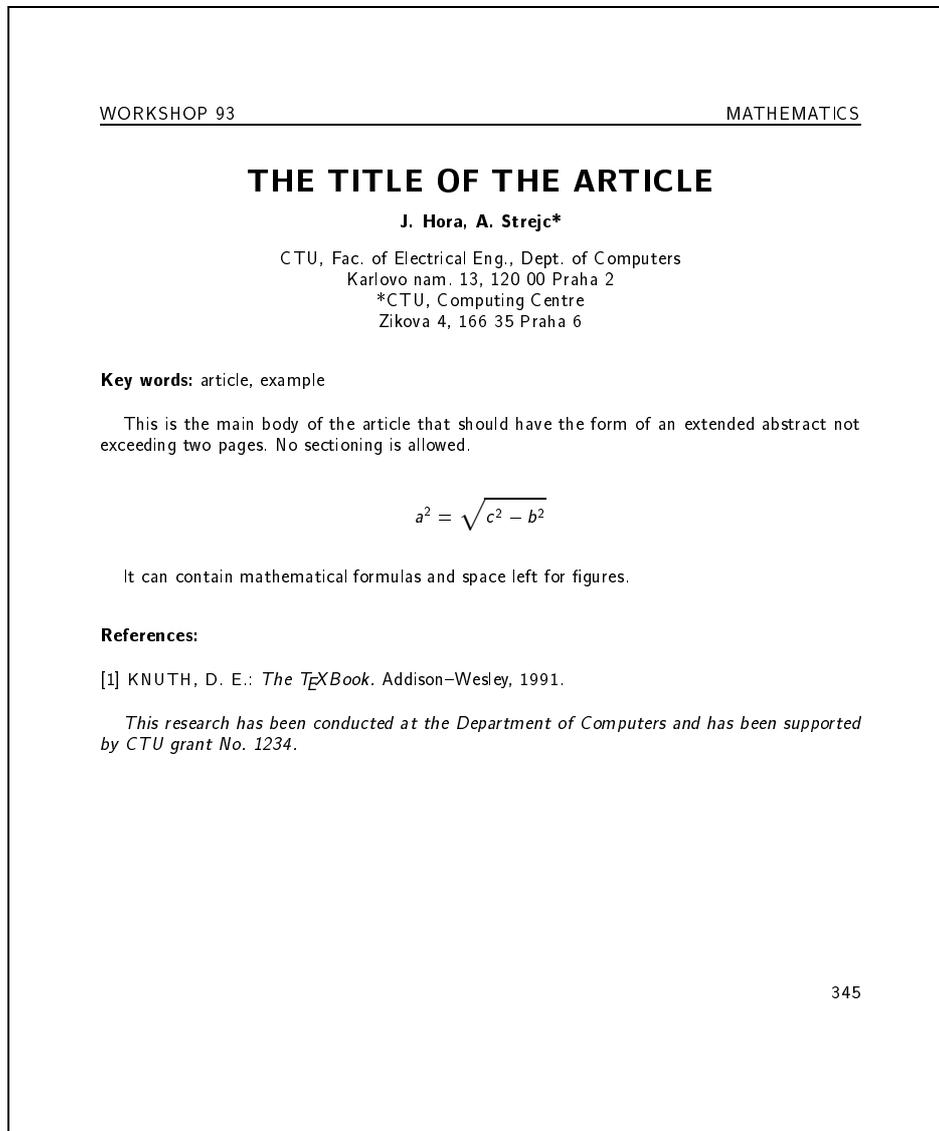


Figure 1: A fictitious Workshop contribution

1. checking the incoming files and editing them to get the exact form for pre-processing
2. pre-processing with a specially written tool
3. L<sup>A</sup>T<sub>E</sub>X typesetting itself, i.e. editing/L<sup>A</sup>T<sub>E</sub>X compiling/viewing the pre-processed file until the definitive output form.

The results were not bad. As can be seen in Table 1, fewer people were needed to process more contributions than in 1992. Indirectly, the typographic level of the proceedings was higher because more time was spent on actual typesetting. The proceedings were received very well. All this filled me with the feeling that the method was right but that more steps would have to be taken to make the process more automated and to keep the idea alive in the case of even more contributions. One more fact can be seen in Table 1. The number of team members decreases year by year. Some people have left and some are still leaving the University for better conditions offered in the private sector of the economy. . .

### 1.3 1994, 1995 – the environment is in action

More exactly, the child was born in 1993, several months after the Workshop 93 seminar. 1994 was the year of the Workshop 94 seminar and the first use of the W94 Environment. I will not write about this system in detail here but will describe only the improved version W95 in the text that follows.

Experience from the previous year had led me to the following opinions:

- the idea of central L<sup>A</sup>T<sub>E</sub>X processing non-L<sup>A</sup>T<sub>E</sub>X contributions is vital, and this method leads to a higher typographic standard of the proceedings that cannot be reached by simple camera-processing
- an obligatory structure for contributions (Figure 1) is good for this special purpose and there is no need to change it
- in the case of an obligatory structure the user-interface can be written to collect part of the data in a user-friendly manner and convert it automatically into L<sup>A</sup>T<sub>E</sub>X source code.

## 2 The W95 environment

The whole system can be divided into two basic parts, as follows:

1. a very reduced installation of emT<sub>E</sub>X (as minimal as possible) as a kernel of the system
2. a user-friendly interface based on the menu system controlling all the operations concerned with writing the article, i.e. collecting the data, and running all necessary applications (editor, T<sub>E</sub>X, dvi-drivers, packing programs etc.)

It was not difficult to deal with the kernel of the system. The following elements of a usual T<sub>E</sub>X installation were taken:

- T<sub>E</sub>X compiler (emT<sub>E</sub>X version 3.141)
- the latex.fmt format file
- Mattes dvi-drivers version 1.5a (dviscr, dvihpj)
- only the necessary fonts (in 300dpi) that can appear in the article for both the screen previewer and laser printer

The main goal was to write Pascal program w95.exe – the interface. It has two basic kinds of functions – to collect user data and to run applications working with it.

Figure 2 shows the structure of the whole system. Here arj.exe is used as a packer/unpacker of user data, keybcs2.com is a resident keyboard driver, dbedit.com is an ASCII text editor, \*.bats are batch files running T<sub>E</sub>X, screen previewer (dviscr) and laser printer driver (dvihpj) respectively, USER DATA is all data concerning the logged user and COMMON DATA is all data common to all users (helps, examples, bases, address lists etc.)

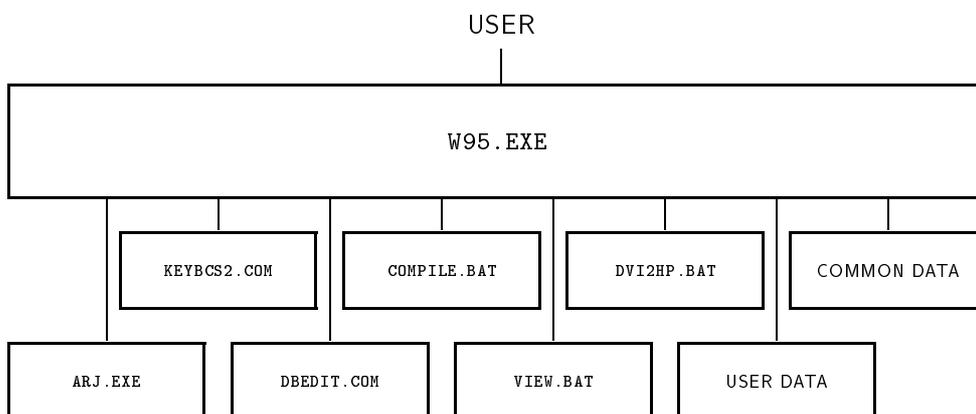


Figure 2: The W95 general structure

## 2.1 User data

As was explained above, we can divide each Workshop article into seven parts. Six of them (all except the body of the article) correspond to files: title.bin, authors.bin, address.bin, keywords.bin, refer.bin and this\_res.txt. The first five are binary files (Pascal records) keeping information about the title, authors etc., the sixth file is a text file storing the final paragraph, which begins with the words ‘This research...’ and contains information about grant support (see Figure 1).

The seventh part of the article – the article body – is the most complex part. It consists of one or more files that contain subparts (let us call them ‘blocks’) of the body of the article. These files have user-specified names with one of the eight valid extensions (.txt, .equ, .dis, .eqa, .lst, .fig, .2fi or .tab). Here the extensions

specify the 'type of block' i.e. what type of information the block contains. The block types will be described later.

The last file, which is accounted as a user data file, is `body_map.bin`. This is a very important file that says what blocks are defined (it keeps their filenames), which of them are 'active' i.e. which should be included in the body of the article) and the order in which the blocks are to be taken.

## 2.2 Logging into W95 – unpacking user data

W95 is a multi-user system, i.e. it can store the user data of more than one user. When it is started it asks you for your *username* (if you are a new user you can choose a name). Each user has all his user data stored in a file called `username.arj`. After logging as *username* ARJ unpacks the user data (Figure 3) and puts it into the working subdirectory.

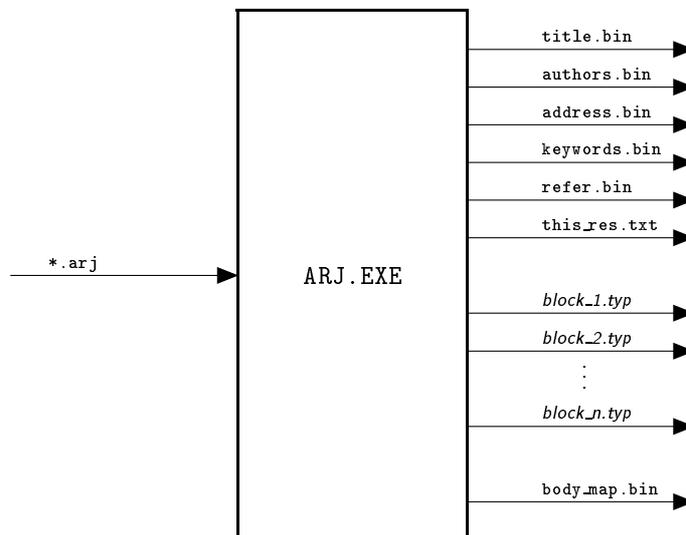


Figure 3: Unpacking the user data

## 2.3 Editing user data

All binary user data files (6 files `*.bin`) are edited directly within the W95 menu system. The file `this_res.txt` and all blocks are edited using the built-in text editor. All these operations are done through the W95 main menu by selecting the proper item. Since binary data consists of Pascal records of constant length, some limitations of data size must be given (e.g. a maximum of 4 lines of title, 12 authors, 4 addresses etc.). Editing these things is very easy and user-friendly. As for addresses, the complete list of all CTU

faculty names, department names and locations is part of the menu system. This kills two birds with one stone: it saves typing time and guarantees error-free and uniform structure of the addresses.

The body of the article can be edited block by block. The number of blocks can be from 0 (theoretically) to 16. People who are accustomed to writing in  $\text{T}_{\text{E}}\text{X}$  or  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  will probably write everything in a single block without problems. Making more blocks is better for those who know little or nothing about the system. The following section tells you more about blocks.

#### 2.4 The philosophy of blocks

A user of W95 may or may not be familiar with  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ . This is the fundamental problem I had to solve. The idea of blocks as elements of the article body seemed to be a solution. A  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  user just opens one block and writes the body of his article in  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  without any problem, because he knows  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ . Someone who is not familiar with these things will create the body of his article in the following way:

1. The user divides his article into several 'homogeneous' parts. He will write each part separately in a block of corresponding type. W95 offers the following block types:
  - `TXT` for writing a piece of plain text
  - `LST` for lists
  - `DIS` for equations without number
  - `EQU` for numbered equations
  - `EQA` for equation arrays
  - `TAB` for tables
  - `FIG` for figures 'per extension'
  - `2FI` for a pair of figures
2. He opens a block. This means that he runs the editor that opens a text file called *name.typ* where *name* is the user-defined name of a block and *.typ* is an extension specifying the block type.
3. Though the opened file is new it is not empty, since it is prepared for writing appropriate things. For example if the file `*.equ` is opened, one can see the lines `\begin{equation}` and `\end{equation}` and more commentary lines with instructions on how to type equations. This is like a context help, but here helps are parts of files so no special keys for their activation are necessary.
4. Swapping to a file with examples concerning the proper block type is available after pressing a hot key. The editor makes it possible to copy these examples into the user's block. So all examples can be  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ ed and the user can compare their source form with the screen output. This can be considered as a quick tutorial (I took this idea from the Borland Turbo Pascal environment).
5. When a block is ready it can be 'turned off' and the next one can be dealt with. So the blocks can be  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ ed either separately or together, or any subset of them can

be taken. In this way the user can, for example, create text parts in the first stage, then all equations etc.

Generally, the philosophy is based on a fact that is well-known from programming. You may write simple programs before you become a real programmer. Indeed, you may write programs and never become a programmer. Similarly, you may write an article in  $\text{\LaTeX}$  and never become a  $\text{\LaTeX}$  user. If you have good helps and ready examples and an average I.Q. you should be able to create the correct  $\text{\LaTeX}$  input. You certainly do not need to learn all the  $\text{\LaTeX}$  commands. Only a little subset of them and a few specially defined macros are necessary for writing the article. The philosophy of  $\text{\LaTeX}$  is to reduce the great number of  $\text{\TeX}$  commands and provide macros for several document styles. Only one style is needed for the Workshop article. The obligatory article structure is given. Therefore a next-step reduction of  $\text{\LaTeX}$  commands can be performed.

## 2.5 Pre-processing user data

Pre-processing is always done automatically when the 'Compile' function is called from the W95 main menu and it is followed immediately by the  $\text{\LaTeX}$  compilation. In the stage of pre-processing a compact input file called `article.tex` is generated from the various-format user data.

1. A standard preamble is generated.
2. Binary data from `title.bin`, `authors.bin`, `address.bin` and `keywords.bin` are transformed into parameters of special defined macros `\title{\{\}\{\}\{\}}`, `\authors{\{\}\{\}\{\}}` etc.
3. Blocks that are set 'active' are appended to `article.tex`. Filenames of all blocks and their activity flags are stored in `body_map.bin`.
4. Binary data from `refer.bin` are transferred into parameters of specially defined macros `\refitem{\{\}\{\}}` generating a list of references.
5. The final paragraph `this_res.txt` is appended to `article.tex`
6. The `\en` macro is generated (end of article).

Let us come back to Figure 1. This is the example article. Figure 4 now shows the appropriate input file `article.tex` as a result of W95 pre-processing.

## 2.6 Other operations

$\text{\LaTeX}$  compilation, previewing and printing the article are standard functions of all  $\text{\TeX}$  installations and there is no need to describe them here.

Other functions are saving user data, copying the `article.tex` file to diskette and exiting the environment.

```

\documentstyle[12pt,w95]{book}
\pagestyle{wpage}
\nofiles
\begin

\title
{THE TITLE OF THE ARTICLE}
{}
{}
{}

\authors
{J. Hora, A. Strejc*}
{}
{}

\addresses
{{}CTU, Fac. of Electrical Eng.,
  Dept. of Computers}
{Karlovo nám 13, 120 00 Praha 2}
{{}*CTU, Computing Centre,
  }
{Zikova 4, 166 35 Praha 6}
{}{}
{}{}

\keywords
{article,
example}

This is the main body of the article that should have the form of an
extended abstract not exceeding two pages. No sectioning is allowed.

\begin{displaymath}
a^2=\sqrt{c^2-b^2}
\end{displaymath}

It can contain mathematical formulas and space left for figures.

\references
\refitem
{KNUTH, D. E.:}
{The \TeX Book.}
{Addison--Wesley, 1991.}

\vspace{4mm}
{\it This research has been conducted at the Department of
Computers and has been supported by CTU grant No.~1234.}

\end

```

Figure 4: Example of article.tex

### 3 Experience of using W95

This section is about processing the proceedings of the most recent Workshop. I prepared the W95 installation packet consisting of an installation program, the packed W95 system and a 'readme' file in Czech. The size of the packet was about 1 Mbyte. I uploaded it on our ftp archive and prepared installation diskettes for people not familiar with ftp or not connected to the network.

A hotline number and e-mail address were announced where any problems could be consulted. Both were often used and many authors received quick hints from me.

The results were very good. Only a few typographic corrections had to be made and many contributions were error-free. This fact allowed us to process such a great number of contributions within 15 days (Table 1).

#### 3.1 Psychological problems

While writing W95 I realized this operation could be quite dangerous and I was sure some people would be against it. In the event, many people did not like W95 but they succeeded in writing their contributions very well. I received only two letters strongly rejecting W95. Most opponents changed their opinion when they received the ready proceedings. Everybody appreciated the typographic level of the proceedings.

Here are some fundamental negative statements that everyone must expect if preparing such a Workshop.

1. Why must I use W95? I have my WordPerfect (or something else) and I want to use it. And you should convert it into your  $\text{T}_{\text{E}}\text{X}$ ...
2. W95 is based on  $\text{T}_{\text{E}}\text{X}$  and nobody will make me study this terrible system (i.e.  $\text{T}_{\text{E}}\text{X}$ ).
3. I do not have time to install W95 and to study your instructions.

All these arguments were very awkward for me. I tried to explain to those people that it would be impossible for me to convert more than 400 contributions from various tools. I explained that we wanted the proceedings in  $\text{T}_{\text{E}}\text{X}$  and not a camera-processed patchwork. I explained that nobody needed to know  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  or  $\text{T}_{\text{E}}\text{X}$  and nobody would have to learn it. I myself installed W95 in the computers of those people who 'did not have time' (the installation takes about 20 seconds) and showed them how to work with it.

I can add that one very positive result also occurred. Some people found the way from W95 to 'full'  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  and  $\text{T}_{\text{E}}\text{X}$  and have become regular users of these systems.

### 4 Conclusions

This is some advice for organizers of similar seminars who would like to follow the described way of processing the proceedings in  $\text{T}_{\text{E}}\text{X}$  and for  $\text{T}_{\text{E}}\text{X}$ -typesetters who would like to participate in such work.

1. Dealing with the idea on such an environment is possible only if an obligatory (or at least very similar) structure of contributions is given.
2. Close cooperation between the organizing committee and typesetters is necessary. The best results can be obtained if one typesetter (head of the team) is a member of the committee.
3. General information and instructions should be announced at least six months before the deadline. The environment installation packet must be available to all authors at least two months before the deadline.
4. All necessary helps must be part of the program. On the other hand helps must not be too detailed for psychological reasons. People do not have time to study excessively long helps.
5. During the time when contributions are being written, a hotline and hot e-mail must be available for authors. It is often more efficient to make a call and get a quick answer than to search in helps or manuals.
6. Personal consultations are a good method for people in trouble who are not far from the typographic center. Expect about 1–2% of all authors to be helped in person.
7. The organizing committee must collect all contributions and decide the order in which they are to be included in the final book. Then (and not earlier) the complete pile of contributions is to be transported to the typographic center. Any other channels from authors to typesetters must be strictly forbidden.
8. Expect most negative statements just after the announcement of the seminar. The number will decrease slightly after installations and after getting familiar with the environment, and it will fall almost to zero when authors receive the proceedings.

# MusiX $\TeX$ , even more beautiful than Music $\TeX$ for music typesetting

Daniel Taupin

Laboratoire de Physique des Solides  
Centre Universitaire  
F-91405 ORSAY Cedex  
France  
taupin@rsovax.lps.u-psud.fr

## Abstract

MusiX $\TeX$  is a new music typesetting package derived from Music $\TeX$ , but it provides more beautiful scores than Music $\TeX$  did. While Music $\TeX$  was a single pass package, MusiX $\TeX$  is a three pass system: the first pass performs a rough  $\TeX$ ing which reports the spacings of each music section, the second pass is a computation of the best note spacings, and the third one is the final  $\TeX$ ing process.

The beauty of single notes does not significantly differ from Music $\TeX$ , but slurs are much more beautiful, and notes are regularly spaced instead of being irregularly spaced with glue.

## 1 History

Music $\TeX$  is now well known and widely spread over the world for music typesetting. It is mostly used by highly skilled amateurs, but even sometimes by music typesetting professionals.

Nevertheless, most connoisseurs actually regretted the questionable aesthetic of its slurs and ties; this ugliness was due to the fact that only horizontal lines (`\hrule`) would resist the glue inserted by  $\TeX$  to achieve line<sup>1</sup> justification. This would lead to something like:

---

1. We use the word 'line' to meet  $\TeX$ ers' way of thinking, but the correct musical word describing a synchronous set of staves tied together with braces or bar rules, is 'system'.



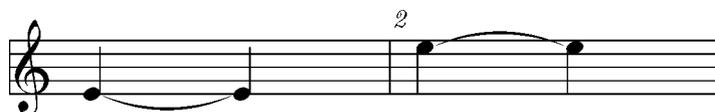
Various suggestions were proposed, all of them resulting in a several pass system, for example inserting `\specials` in the DVI, analyzing this DVI byte after byte to compute the accurate size of needed slurs, and eventually invoking metafont when needed to generate the final text with the slurs/ties of the exact required length.

In 1992, Ross Mitchell<sup>2</sup> proposed another package (initially called 'Muflex') in which  $\text{\TeX}$  explicitly writes in a file the spacings consumed – regardless of an arbitrary unique scale factor – by each group of notes.

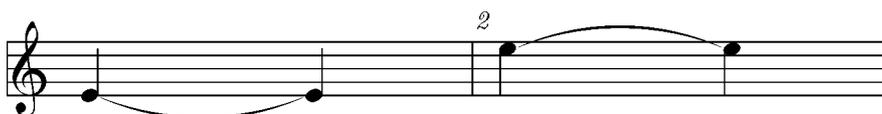
At the next pass, this file read by a small program – `musicflx` initially in Fortran, now in C – which determines the optimal value of the elementary spacing (`\elemskip`) so that each score line exactly fits in a  $\text{\TeX}$  line (i.e. one `\hsize`) without any additional glue to be inserted.

Then, at final pass,  $\text{\TeX}$  reads the brainstorming results of this small Fortran or C program, and it readily knows which spacing it must assign to the various notes in order to avoid any glue inclusion to fill the line. Thus, if the unit length `\elemskip` is known to be 14.25 pt, while a given slur is 13 units long<sup>3</sup>, it is then easy to choose the convenient sequence of symbols to build a smart curve of the right length, with an accuracy of one point.

Thus, using `MusiX $\text{\TeX}$`  the previous sequence becomes:



and when the spacings are increased, one obtains:



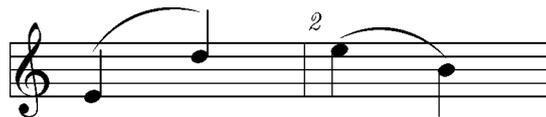
After this preliminary trial `MusiX $\text{\TeX}$`  was created by Andreas Egler<sup>4</sup> and Daniel Taupin, tuning the Muflex by Ross Mitchell and 'negotiating' some features of `Music $\text{\TeX}$` .

---

2. CSIRO Division of Atmospheric Research, Mordialloc, Victoria – Australia.  
 3. In fact this is seldom an integer, due to additional spaces for bars and special insertions.  
 4. Ruhr-Uni-Bochum, D-44793 Bochum.

## 2 The characteristics of MusiX $\TeX$

Most commands are taken from Music $\TeX$ , sometimes with name changes such as `debutmorceau` becoming `startpiece`. Some people may smile about this point, but as a matter of fact the existence of keywords taken from the French language sometimes triggers allergy reactions<sup>5</sup>... In addition, MusiX $\TeX$  provides specific macros to achieve slurs whose final altitude is different from the initial, for example:



Here is another example – Intermezzo op. 117,1 by Brahms – according to data provided by Miguel Filgueiras

Two systems of piano music for Intermezzo op. 117,1 by Brahms. Each system consists of a grand staff with a treble and bass clef. The first system shows a slur over a sequence of notes in the right hand, with a '2' above it. The second system shows a slur over a sequence of notes in the right hand, with a '3' above it, and another slur over a sequence of notes in the right hand, with a '4' above it.

## 3 MusiX $\TeX$ 's advantages and difficulties

### 3.1 The glue problem

Obviously, the *glue* notion is essential to  $\TeX$ , since it enables justifications equally spread over the text lines. Music $\TeX$  also reasonably uses this feature, in order to approximately

5. Werner Licking humoristically suggested that German people use something like `\HierbeginntmeinePartitur` instead of `\debutmorceau`.

justify the lines of music scores, with that specific difficulty that music ‘paragraphs’ may not finish with partial lines.

However, the Music $\TeX$  experience was that glue imposed slurs and ties with a wide horizontal section, in order to enable overlappings or extensions of the `\hrule` form. Moreover, if the user was poorly careful, glue would introduce wide empty spaces between compact sequences of notes.<sup>6</sup>

MusiX $\TeX$  solves all these problems, but the unfortunate counterpart is that the least parasitic space:

- forgotten % at end of line when not ended with a keyword,
- boxes containing text (lyrics) spilling out of the horizontal length allocated to the group of notes,

results in, at best some `Underfull` boxes filled *in extremis* with glue, at worst catastrophic `Overfull` boxes.

As a consequence, when getting these messages, hunting spilling boxes or parasitic spaces becomes sophisticated operation, of the competences of a skilled  $\TeX$ pert only.

### 3.2 Compatibility with MusiX $\TeX$

Our idea – as well as of another musician, Werner Icking – was to build a new package using the same commands as Music $\TeX$  except hard impossibility, and offering in addition a more automatic page layout, and an aesthetic slur generation by means of a set of additional commands.

But later, one of the co-authors – Andreas Egler – wanted to do the other way, namely a distinct *new* package, obviously taking advantage of the bases of Music $\TeX$  but providing different internal and external commands:

- unification of certain distinct commands whose choice could have been automatized... with deletion of the old ones;
- compilation speed enhancement replacing many `\def` by `\let`, admittedly faster but badly encapsulated;
- replacement of command names taken from French and Italian with English looking keywords;<sup>7</sup>
- locking some internal identifiers by inserting @ ;
- ambiguous shortening of command names, which were really long but self-explanatory, probably due to memory problems in his computer.

Nevertheless, although he claimed that music typesetters should give up using Music $\TeX$  to definitely move to MusiX $\TeX$ , with no possible backstepping, he accepted

6. This the unfortunate experience of most Music $\TeX$  beginners.

7. However Andreas Egler is German, and I would have definitely preferred that he replace `\barre` with the German word `\Takt` rather than `\bar` which is confusing with basic  $\TeX$ /L $\TeX$ .

to develop an optional set of macros, `musiccpt`, which superposes the fundamental MusiX $\TeX$  most of the commands commonly used by the 'old-fashioned' Music $\TeX$ -ers.

And all this works... except some details presently under revision.

Thus, it is presently possible to have a unique source file which can be compiled with both Music $\TeX$  or MusiX $\TeX$ . To do that it suffices changing the `\inputs` at the beginning or, even better, creating two formats:

1. a format with Plain  $\TeX$ +`musicnft+musictex`,
2. a format with Plain  $\TeX$ +`musixtex+musiccpt`.

### 3.3 The remaining problems

1. A persisting controversy between A. Egler on one side, and D. Taupin and W. Icking on the other, about the developing strategy for MusiX $\TeX$ .
2. Clean lyrics insertion, since `musixflx/Muflex` is unable to handle and shrink text lengths to meet zero glue justification.

## 4 Availability

- Original version (presently T.396) supported by D. Taupin at `ftp://rsovax.lps.u-psud.fr/[anonymous.musixtex]` and at `ftp://hprib.lps.u-psud.fr/pub/musixtex`;
- Copies in the various CTANs (directory `macros/musixtex/taupin`);
- Andreas Egler's version is also available now in the various CTANs (directory `macros/musixtex/egler`). Note that Egler's version is not compatible with sources designed for Music $\TeX$ .

## 5 Two examples

On page 356 is the beginning of Charles Gounod's 'Ave Maria', transcribed for organ and soloist (violin and/or singer), typeset with format `musictex.fmt` (Music $\TeX$ ); in page 357 is the output of the same source file using format `musixtex.fmt` (MusiX $\TeX$ ). Note the different slur shapes, and the unfortunate exceeding glue at bars 10–12 with Music $\TeX$ .

## Méditation – Ave Maria

G.O.: flûte 8' [+4']  
 Positif : flûtes 8'+4' [+2']  
 Pédale : 16', acc. positif

Charles Gounod & J.-S. Bach  
 Transcription Orgue+soliste Daniel Taupin  
 & Markus Veittes

The score is divided into four systems, each containing three staves: Positif (organ), Chant (voice), and Violon (violin). The music is in common time (C) and features a complex, rhythmic accompaniment in the Positif parts. The Chant part includes the following lyrics:

5 A — ve Ma —  
 6  
 7 ri — a,  
 8  
 9 gra — ti — a  
 10 ple — na,  
 11 Do — mi — nus te — cum,  
 12

## Méditation – Ave Maria

G.O.: flûte 8' [+4']  
 Positif : flûtes 8'+4' [+2']  
 Pédale : 16', acc. positif

Charles Gounod & J.-S. Bach  
 Transcription Orgue+soliste Daniel Taupin  
 & Markus Veittes

The musical score is arranged in four systems. Each system contains three staves: Positif (organ), Chant (voice), and Violon (violin). The Positif part features a complex rhythmic pattern of eighth and sixteenth notes. The Chant part includes the lyrics: "A-ve Ma-ri-a, gra-ti-a ple-na, Do-mi-nus te-cum,". The Violon part provides harmonic support with sustained notes and melodic lines. Measure numbers 2, 3, 5, 6, 7, 8, 9, 10, 11, and 12 are indicated above the staves.



# $\varepsilon$ - $\text{T}\text{E}\text{X}$ : a 100%-compatible successor to $\text{T}\text{E}\text{X}$

Following humbly in the footsteps of the Grand Wizard

Philip Taylor

Royal Holloway & Bedford New College  
University of London  
United Kingdom  
p.taylor@vms.rhnc.ac.uk

## 1 Introduction

$\varepsilon$ - $\text{T}\text{E}\text{X}$  is the first concrete result of an international research & development project, the  $\mathcal{N}\mathcal{T}\mathcal{S}$  Project, which was established under the ægis of DANTE during 1992. The aims of the project are to perpetuate and develop the spirit and philosophy of  $\text{T}\text{E}\text{X}$ , whilst respecting Knuth's wish that  $\text{T}\text{E}\text{X}$  itself should remain frozen.

The group were very concerned that unless there existed some evolutionary flexibility within which  $\text{T}\text{E}\text{X}$  could react to changing needs and environments, it might all too soon become eclipsed by more modern yet less sophisticated systems. Accordingly they agreed to investigate a possible successor or successors to  $\text{T}\text{E}\text{X}$ , successors which would enshrine and encapsulate all that was best in  $\text{T}\text{E}\text{X}$  whilst being freed from the evolutionary constraints which Knuth had placed on  $\text{T}\text{E}\text{X}$  itself. To avoid any suggestion that it was  $\text{T}\text{E}\text{X}$  which the group sought to develop against Knuth's wishes, a working title of  $\mathcal{N}\mathcal{T}\mathcal{S}$  (for New Typesetting System) was chosen for the project.

During the initial meetings of the  $\mathcal{N}\mathcal{T}\mathcal{S}$  group, it became clear that there were two possible approaches to developments based on  $\text{T}\text{E}\text{X}$ : an evolutionary path which would simply continue where Knuth had left off, and which would use as its basis the source code of  $\text{T}\text{E}\text{X}$  itself (i.e. `TeX.Web`); the other a revolutionary path which would be based on a completely new implementation of  $\text{T}\text{E}\text{X}$ , using a modern rapid-prototyping language which could allow individual components of the system to be modified or replaced in a simple and straightforward manner. The group agreed that the latter (revolutionary) approach had much greater potential, but were aware that the re-implementation would be non-trivial, and would require external funding to bring it to fruition in finite time;

accordingly they agreed to concentrate their initial efforts on the former (evolutionary) path, and set to work to specify and implement a direct derivative of  $\text{T}\text{E}\text{X}$  which became known as  $\varepsilon\text{-T}\text{E}\text{X}$ . The  $\varepsilon$  of  $\varepsilon\text{-T}\text{E}\text{X}$  may be read as *extended*, *enhanced*, *evolutionary* or *European* at will (!), and is also an acknowledgement of the parallel developments which have led the  $\text{L}\text{A}\text{T}\text{E}\text{X}3$  team to modify their initial goal and to release an interim  $\text{L}\text{A}\text{T}\text{E}\text{X}$ ,  $\text{L}\text{A}\text{T}\text{E}\text{X}2\varepsilon$ , which is directly derived from the earlier  $\text{L}\text{A}\text{T}\text{E}\text{X}$  sources.

The group took as starting point for the development of  $\varepsilon\text{-T}\text{E}\text{X}$  the many contributions which had been made on  $\text{N}\text{T}\text{S-L}$  (the open mailing list on which discussions pertinent to  $\varepsilon\text{-T}\text{E}\text{X}$  &  $\mathcal{N}\mathcal{T}\mathcal{S}$  take place), together with the extremely interesting list of ideas which Knuth gives at the end of  $\text{T}\text{e}\text{X}82.\text{B}\text{u}\text{g}$ , and which he describes as '*Possibly nice ideas that will not be implemented*' (and which he contrasts with '*Bad ideas that will not be implemented*'). Individual members of the group also contributed ideas of their own which had not necessarily been discussed publicly. All proposals were then subjected to a rigorous vetting procedure to ensure that they conformed to the  $\varepsilon\text{-T}\text{E}\text{X}$  philosophy, which may be summarised as follows:

$\varepsilon\text{-T}\text{E}\text{X}$  will in all ways demonstrate its affinity to, and derivation from, Knuth's  $\text{T}\text{E}\text{X}$ ; it will be implemented as a change-file to  $\text{T}\text{e}\text{X}.\text{W}\text{e}\text{b}$ , and will not exploit features which could only be achieved by using a particular implementation, operating system or language; it will be capable of being used successfully on a machine as small as an 80286-based PC or similar.

At format-generation time, a user will have the option of generating either a  $\text{T}\text{E}\text{X}$ -compatible format or an  $\varepsilon\text{-T}\text{E}\text{X}$  format; if the  $\text{T}\text{E}\text{X}$ -compatible format is subsequently used in conjunction with  $\varepsilon\text{-T}\text{E}\text{X}$ , the result will be *Trip-compatible*<sub>*j*</sub> (i.e. indistinguishable from  $\text{T}\text{E}\text{X}$  proper). If an  $\varepsilon\text{-T}\text{E}\text{X}$  format is generated and used in conjunction with  $\varepsilon\text{-T}\text{E}\text{X}$ , then provided that none of the new  $\varepsilon\text{-T}\text{E}\text{X}$  primitives are used, the results will be identical to those which would be produced using  $\text{T}\text{E}\text{X}$  proper. If an  $\varepsilon\text{-T}\text{E}\text{X}$  format is used in conjunction with  $\varepsilon\text{-T}\text{E}\text{X}$  and if one or more of the new  $\varepsilon\text{-T}\text{E}\text{X}$  primitives are used, then those portions of the document which are affected by the new primitive(s) may be processed in a manner unique to  $\varepsilon\text{-T}\text{E}\text{X}$ ; other portions of the document will be processed in a manner identical to that of  $\text{T}\text{E}\text{X}$  proper. Only if an  $\varepsilon\text{-T}\text{E}\text{X}$  format is used in conjunction with  $\varepsilon\text{-T}\text{E}\text{X}$  and if an explicit assignment is made to one of the *enhanced-mode* variables to enable that particular enhanced mode will  $\varepsilon\text{-T}\text{E}\text{X}$  behave in a manner which may be distinguishable from that of  $\text{T}\text{E}\text{X}$  even if no other reference to an  $\varepsilon\text{-T}\text{E}\text{X}$  primitive occurs anywhere in the document. (These modes of operation are referred to as *compatibility-mode*, *extended-mode* and *enhanced-mode* respectively.)

All new  $\varepsilon\text{-T}\text{E}\text{X}$  primitives will be syntactically identical to existing  $\text{T}\text{E}\text{X}$  primitives: that is, they will be either *control-words* or *control-symbols* within a normal cat-code regime. Where an analogous primitive exists within  $\text{T}\text{E}\text{X}$ , the corresponding  $\varepsilon\text{-T}\text{E}\text{X}$  primitive(s) will occupy the same syntactic niche. Every effort will be made

to ensure that new  $\epsilon$ -T<sub>E</sub>X primitives fit into the existing set of T<sub>E</sub>X datatypes; no new datatype will be introduced unless it is absolutely essential.

In brief, this implies that  $\epsilon$ -T<sub>E</sub>X will follow the principle of least surprise: an existing T<sub>E</sub>X user, on using  $\epsilon$ -T<sub>E</sub>X for the first time, should not be surprised by  $\epsilon$ -T<sub>E</sub>X's behaviour, and should be able to take advantage of new  $\epsilon$ -T<sub>E</sub>X features without having either to unlearn some aspects of T<sub>E</sub>X or to learn some new  $\epsilon$ -T<sub>E</sub>X philosophy.

## 2 Installation

It is intended that  $\epsilon$ -T<sub>E</sub>X be available ready-compiled for those systems for which pre-compiled binaries are the norm (e.g. MS-DOS, VMS, ...); for other systems such as Unix<sup>TM</sup>,  $\epsilon$ -T<sub>E</sub>X is supplied as a change-file which will need to be applied to TeX.Web in the normal way. However, since there will already be an implementation-specific change-file for the system of interest, some means will be required of merging TeX.Web with not one but (at least) two change-files; possibilities include PatchWeb, Tie, etc. , but if none of these is available then WebMerge, a T<sub>E</sub>X script, is supplied and can be used as a slower but satisfactory alternative. In practice, two or three change-files may be needed: the  $\epsilon$ -T<sub>E</sub>X system-independent change-file, the T<sub>E</sub>X system-dependent change-file, and perhaps a small  $\epsilon$ -T<sub>E</sub>X system-dependent change-file. The system-independent  $\epsilon$ -T<sub>E</sub>X change-file is supplied as part of the  $\epsilon$ -T<sub>E</sub>X kit, and sample system-dependent  $\epsilon$ -T<sub>E</sub>X change-files are also supplied which may be used as a guide to those places at which system-dependent interactions are to be expected: an experienced implementor should have little difficulty in modifying one of these to produce an  $\epsilon$ -T<sub>E</sub>X system-dependent change-file for the system of interest. Once  $\epsilon$ -T<sub>E</sub>X has been tangled and woven, it should be compiled and linked in the normal way.

Once a working binary (or binaries, for those systems which have separate executables for IniTeX and VirTeX) has been acquired or produced, the next step will be to generate a suitable format file or files. Whilst  $\epsilon$ -T<sub>E</sub>X can be used in conjunction with Plain.TeX to produce a Plain *e-format*, it is better to use the supplied e-Plain.TeX file which supplements the  $\epsilon$ -T<sub>E</sub>X primitives with additional useful control sequences. When generating the format file, and regardless of the format source used, one fundamental decision must be made: is  $\epsilon$ -T<sub>E</sub>X to generate a *compatibility mode* format, or an *extended mode* format? If the former, *all*  $\epsilon$ -T<sub>E</sub>X extensions and enhancements will be disabled, the format will contain only the T<sub>E</sub>X-defined set of primitives, and any subsequent use of the format in conjunction with  $\epsilon$ -T<sub>E</sub>X will result in completely T<sub>E</sub>X-compatible behaviour and semantics, including compatibility at the level of the Trip test. If the latter option, however, is selected, then all extensions present in  $\epsilon$ -T<sub>E</sub>X will automatically be activated, and the format file will contain not only the T<sub>E</sub>X-defined set of primitives but also those defined by  $\epsilon$ -T<sub>E</sub>X itself; any subsequent use of such a format in conjunction with  $\epsilon$ -T<sub>E</sub>X will result in  $\epsilon$ -T<sub>E</sub>X operating in *extended mode*; documents which contains no references to any of the  $\epsilon$ -T<sub>E</sub>X-defined primitives will continue to generate results identical to those

which would have been produced using  $\text{T}\text{E}\text{X}$ , but compatibility at the `Trip`-test level can no longer be accomplished, and of course any document which makes reference to an  $\varepsilon\text{-T}\text{E}\text{X}$  primitive will generate results which could not have been accomplished using  $\text{T}\text{E}\text{X}$ . It should be noted that neither a *compatibility mode* format nor an *extended mode* format may be used in conjunction with  $\text{T}\text{E}\text{X}$  itself; they are only suitable for use in conjunction with  $\varepsilon\text{-T}\text{E}\text{X}$ , since formats are not in general portable. Finally it should be emphasised that even if an *extended mode* format is generated, any document processed using such a format but not referencing any  $\varepsilon\text{-T}\text{E}\text{X}$ -defined primitive will produce results identical to those which would have been produced had the same document been processed using  $\text{T}\text{E}\text{X}$ ; only if the document makes an explicit assignment to one of the *enhanced mode* state variables (`\TeXXeTstate` is the only instance of these in V1 of  $\varepsilon\text{-T}\text{E}\text{X}$ ) will compatibility with  $\text{T}\text{E}\text{X}$  be compromised:  $\varepsilon\text{-T}\text{E}\text{X}$  is then said to be operating in *enhanced mode* rather than *extended mode*.

The choice between generating a *compatibility mode* format and an *extended mode* format is made at the point of specifying the format source file: assuming that the operating system supports command-line entry with parameters, then a normal  $\text{T}\text{E}\text{X}$  format-generation command would probably resemble:

```
IniTEx Plain \dump
```

or if the more verbose interactive form is preferred:

```
IniTEx
**Plain
*\dump
```

With  $\varepsilon\text{-T}\text{E}\text{X}$ , exactly the same command will achieve exactly the same effect, and the format generated will be a *compatibility-mode* format; thus assuming that the inversion of  $\varepsilon\text{-T}\text{E}\text{X}$  is invoked with the command `eIniTEx`, the following will both generate *compatibility-mode* formats:

```
eIniTEx Plain \dump
```

and

```
eIniTEx
**Plain
*\dump
```

In order to generate an *extended mode* format, the file-specification for the format source file must be preceded by an asterisk (\*); whilst this may seem an inelegant mechanism, it has the great advantage that it avoids almost all system dependencies (GUI systems excepted, of course), and the asterisk as a component element of a filename is a very remote possibility (most filing systems reserve the asterisk as a 'wild card' character,

which can therefore not form a part of a real file name *per se*). Thus to generate an *extended mode* Plain format, the following dialogue may be used:

```
IniTeX *Plain \dump
```

or

```
eIniTeX  
***Plain  
*\dump
```

and to generate an *extended mode* e-Plain format, the following instead:

```
eIniTeX e-Plain \dump
```

or

```
eIniTeX  
***e-Plain  
*\dump
```

Once suitable formats have been generated, they can then be used in conjunction both with e-IniTeX and e-VirTeX without further formality: in particular, no asterisk is needed (nor should be used!) if a format is specified, since the format implicitly defines (depending as its mode of generation) in which mode (compatibility or extended)  $\epsilon$ -T<sub>E</sub>X will operate. Thus, for example, if a Plain format had been generated in *compatibility mode*, and an e-Plain format had been generated in *extended mode*, then both:

```
eIniTeX &Plain
```

and

```
eVirTeX &Plain
```

will cause  $\epsilon$ -T<sub>E</sub>X to process any subsequent commands in *compatibility mode*. On the other hand, both

```
eIniTeX &e-Plain
```

and

```
eVirTeX &e-Plain
```

will cause  $\epsilon$ -T<sub>E</sub>X to process any subsequent commands in *extended mode*, but only because the e-Plain format was generated in *extended mode*: it is not the *name* of the format, nor is it the contents of the *source* of the format, which determine the mode of operation – it is the *mode of operation* which was used when the format was generated. Any format generated in *compatibility mode* will cause  $\epsilon$ -T<sub>E</sub>X to operate in *compatibility*

*mode* whenever it is used, whilst the same format generated in *extended mode* will cause  $\varepsilon$ -TeX to operate in *extended mode* whenever it is used.

Although  $\varepsilon$ -TeX is completely TeX-compatible, and there is therefore no real reason why any system should need both TeX and  $\varepsilon$ -TeX, it is anticipated that until complete confidence exists in the compatibility of  $\varepsilon$ -TeX many sites and users will prefer to retain instances of each. For this reason the supplied change-files and binaries will ensure that both TeX and  $\varepsilon$ -TeX can happily co-exist on any system by a careful choice of non-overlapping name-spaces. This might, for example, be achieved by changing the default extension for *e-format* files to (say) `.efm` rather than `.fmt`, or by referencing a different format directory and/or environment variable (for example, `eTeX_formats` rather than `TeX_formats`).

### 3 The new features

Bearing in mind the constraints outlined in the introduction, the group identified approximately 30 new primitives which they believed would give added functionality to  $\varepsilon$ -TeX without compromising its compatibility with TeX; of the 30 new primitives, 25 are extensions (which by definition do not affect the semantics of existing TeX documents), whilst just six (all concerned with the implementation of TeX-XET) are associated with an enhancement. In addition to the new primitives, additional functionality was added to some existing primitives, and TeX's behaviour in some unusual boundary conditions was made more robust (this last has been subsumed in the most recent version of TeX, so this is no longer  $\varepsilon$ -TeX-specific).

The new features are listed and briefly described below, clustered together to indicate related functionality; it is intended that a full description of each together with appropriate examples will be published in *The  $\varepsilon$ -TeX Manual*, which it is hoped will become the definitive reference manual for  $\varepsilon$ -TeX.

#### 3.1 Additional control over expansion

- `\protected`
- `\detokenize`
- `\unexpanded`

`\protected` is a prefix, analogous to `\long`, `\outer`, and `\global`; it associates with the macro being defined an attribute which inhibits expansion of the macro in expansion-only contexts (for example, within the parameter text of a `\write` or `\edef`); if, however, the parser or command processor (TeX's 'oesophagous' and 'stomach', in Knuth's alimentary paradigm) is currently demanding a *command*, then the `\protected` macro will expand in the normal way. This behaviour is identical to that displayed by the explicit expansion of a token-list register through the use of `\the`; the same model is used elsewhere in  $\varepsilon$ -TeX to achieve a consistent paradigm for *partial expansion*.

`\detokenize`, when followed by a *general text*, expands to yield a sequence of character tokens of catcode 10 (*space*) or 12 (*other*) corresponding to a decomposition of the tokens of the *balanced text* of the unexpanded *general text*; c.f. `\showtokens`. The effect is rather as if `\scantokens` (q.v. ) were applied to the *general text* within a regime in which only `\catcodes` 10 and 12 existed. Note that in order to preserve the boundaries between *control words* and any following *letter*, a *space* is yielded after each control word including the last.

`\unexpanded`, when followed by a *general text*, expands to yield the *balanced text* of the unexpanded *general text*. No further expansion will occur if  $\epsilon$ -T<sub>E</sub>X is currently performing a `\write`, `\edef`, etc. , but further expansion will occur if the parser or command processor is currently demanding a *command*. The effect is as if the *general text* were assigned to a token list register, and the latter were then partially expanded using `\the`, but no assignment actually takes place; thus `\unexpanded` can be used in expansion-only contexts.

### 3.2 Provision for re-scanning already read text

- `\readline`
- `\scantokens`

`\readline` is analogous to `\read`, but treats each character as if it were currently of catcode 10 (*space*) or 12 (*other*); the text thus read is therefore suitable for being scanned and re-scanned (using `\scantokens`, q.v. ) under different catcode regimes.

`\scantokens`, when followed by a *general text*, decomposes the *balanced text* of the *general text* into the corresponding sequence of characters as if the *balanced text* were written unexpanded to a file; it then uses T<sub>E</sub>X's `\input` mechanism to re-process these characters under the current catcode regime. As the `\input` mechanism is used, even hex notation ( $\hat{xy}$ ) will be re-interpreted. Parentheses and a single space representing the pseudo-file will be displayed if `\tracingscantokens` (q.v. ) is positive and non-zero.

### 3.3 Environmental enquiries

- `\eTeXrevision`
- `\eTeXversion`
- `\grouplevel`
- `\grouptype`
- `\ifcsname`
- `\ifdefined`
- `\lastnodetype`

`\eTeXrevision`: an primitive which expands to yield a sequence of character tokens of catcode 12; these represent the minor component of the combined version/revision

number. Pre-release versions will be characterised by an initial minus sign (-), whilst post-release versions will be implicitly positive; both will contain an explicit leading decimal point, which will follow any minus sign present.

`\eTeXversion`: an internal read-only integer representing the major component of the combined version/revision number.

`\grouplevel`: an internal read-only integer which returns the current group level (i.e. depth of nesting).

`\grouptype`: an internal read-only integer which returns the type of the innermost group as an integer in the range 0..16. Textual definitions of these types are provided through the an associated macro library, but it is intended that these definitions shall be easily replaceable by national language versions in environments within which English language texts are sub-optimal.

`\ifcsname`: similar in effect to the sequence `\unless \expandafter \ifx \expandafter \relax \csname` but avoids the side-effect of the `cs-name` being ascribed the value `\relax`, and also does not rely on `\relax` having its canonical meaning. No hash-table entry is used if `cs-name` does not exist. (`\unless` is explained below.)

`\ifdefined`: similar in effect to `\unless \ifx \undefined`, but does not require `\undefined` to actually be undefined, since no explicit comparison is made with any particular control sequence.

`\lastnodetype`: an internal read-only integer which returns the type of the last node on the current list as an integer in the range  $-1..15+$  (only values  $-1..15$  are defined in the first release, but future releases may define additional values). Textual definitions of these types are provided through an associated macro library, but it is intended that these definitions shall be easily replaceable by national language equivalents for use in environments within which the use of English language texts is sub-optimal.

### 3.4 Generalisation of the `\mark` concept: a class of `\marks`

- `\marks`
- `\botmarks`
- `\firstmarks`
- `\topmarks`
- `\splitfirstmarks`
- `\splitbotmarks`

`\marks`: whereas  $\text{\TeX}$  has only one `\mark`, which has to be over-loaded if more than one class of information is to be saved (e.g. over-loading is necessary if separate information for recto and verso pages is to be maintained),  $\varepsilon\text{-}\text{\TeX}$  has a whole class of `\marks` (16, in the first release); thus rather than writing `\mark general text` as in  $\text{\TeX}$ , in  $\varepsilon\text{-}\text{\TeX}$  one writes `\mark 4-bit number general text`. For example, `\marks 0` could be used to retain information for the verso page, whilst `\marks 1` could retain information for the recto.

There are equivalent classes for the five  $\backslash\text{marks}$  variables  $\backslash\text{botmarks}$ ,  $\backslash\text{firstmarks}$ ,  $\backslash\text{topmarks}$ ,  $\backslash\text{splitfirstmarks}$  and  $\backslash\text{splitbotmarks}$ .

### 3.5 Bi-directional typesetting: the $\text{\TeX-X}\text{\LaTeX}$ primitives

- $\backslash\text{TeXXeTstate}$
- $\backslash\text{beginL}$
- $\backslash\text{beginR}$
- $\backslash\text{endL}$
- $\backslash\text{endR}$
- $\backslash\text{predisplaydirection}$

$\text{\TeX-X}\text{\LaTeX}$  was developed by Peter Breitenlohner based on the original  $\text{\TeX-X}\text{\LaTeX}$  of Donald Knuth and Pierre MacKay; whereas  $\text{\TeX-X}\text{\LaTeX}$  generated non-standard DVI files,  $\text{\TeX-X}\text{\LaTeX}$  generates perfectly normal DVI files which can therefore be processed by standard DVI drivers (assuming, of course, that the necessary fonts are available). Both systems permit the direction of typesetting (conventionally left-to-right in Western documents) to be reversed for part or all of a document, which is particularly useful when setting languages such as Hebrew or Arabic.

$\backslash\text{beginL}$ : indicates the start of a region (e.g. a section of text, or a pre-constructed box) which should be set left-to-right;

$\backslash\text{beginR}$ : indicates the start of a region which should be set right-to-left;

$\backslash\text{endL}$ : indicates the end of a region which should be set left-to-right;

$\backslash\text{endR}$ : indicates the end of a region which should be set right-to-left;

$\backslash\text{TeXXeTstate}$ : an internal read/write integer, its value is zero or negative to indicate that  $\text{\TeX-X}\text{\LaTeX}$  features are not to be used; a positive value indicates that they may be used. As the internal data structures built by  $\text{\TeX-X}\text{\LaTeX}$  differ from those built by  $\text{\TeX}$ , and as the typesetting of a document by  $\text{\TeX-X}\text{\LaTeX}$  may therefore differ from that performed by  $\text{\TeX}$ ,  $\backslash\text{TeXXeTstate}$  defaults to zero, and even if set positive during format creation will be re-set to zero before the format is dumped. Explicit user action is therefore required to enable  $\text{\TeX-X}\text{\LaTeX}$  semantics, and  $\text{\TeX-X}\text{\LaTeX}$  is thereby classed as an *enhancement*, not simply an *extension*.

### 3.6 Additional debugging features

- $\backslash\text{interactionmode}$
- $\backslash\text{showgroups}$
- $\backslash\text{showtokens}$
- $\backslash\text{tracingassigns}$
- $\backslash\text{tracinggroups}$
- $\backslash\text{tracingifs}$
- $\backslash\text{tracingscantokens}$
- Additional detail for  $\backslash\text{tracingcommands}$

`\interactionmode`: whereas in  $\text{\TeX}$  there exist only explicit commands such as `\scrollmode`, `\errorstopmode`, etc. , in  $\varepsilon\text{-}\text{\TeX}$  read/write access is provided via `\interactionmode` (an internal integer); assigning a numeric value sets the associated mode, whilst the current mode may be ascertained by interrogating its value. Symbolic definitions of these values are provided through an associated macro library, but it is intended that these definitions shall be easily replaced by national-language equivalents in environments within which the use of English is sub-optimal.

`\showgroups`: (e-)TeX has many different classes of group, which should normally be properly balanced and nested; if a nesting or imbalance error occurs, it can be *very* difficult to track down the source of the problem. `\showgroups` causes  $\varepsilon\text{-}\text{\TeX}$  to display the level and type of all active groups from the point within which it was called.

`\showtokens`, when followed by a *general text*, displays a sequence of characters corresponding to the decomposition of the `balanced text` of the unexpanded *general text*; c.f. `\detokenize`.

`\tracinggroups`: a further aid to debugging runaway-group problems, the command `\tracinggroups` (an internal read/write integer) causes  $\varepsilon\text{-}\text{\TeX}$  to trace entry and exit to every group while set to a positive non-zero value.

`\tracingscantokens`: an internal read/write integer, assigning it a positive non-zero value will cause an open-parenthesis and space to be displayed whenever `\scantokens` is invoked; the matching close-parenthesis will be recorded when the scan is complete. If a traceback occurs during the expansion of `\scantokens`, the first displayed line number will reflect the logical line number of the pseudo-file created from the parameter to `\scantokens`; thus enabling `\tracingscantokens` can assist in identifying why an seemingly irrational line number is shown as the source of error (the traceback always continues until the line number of the actual source file is displayed).

If `\tracingcommands` is greater than 2, additional information is displayed.

### 3.7 Miscellaneous primitives

- `\everyeof`
- `\middle`
- `\unless`

`\everyeof`: this is one of Knuth's 'possibly good ideas', listed at the end of `\TeX82.Bug`; analogous to the other `\every...` primitives, it takes as parameter a *balanced text*, the tokens of which are inserted when the end of a file (either real or virtual, if `\scantokens` is used) is reached. This allows `\input` statements to be used within the replacement text of `\edefs`, and allows totally arbitrary files to be `\input` within a  $\varepsilon\text{-}\text{\TeX}$  conditional, since the necessary `\fi` can be inserted before  $\varepsilon\text{-}\text{\TeX}$  complains that it has fallen off the end of the file.

`\middle`: analogous to  $\text{\TeX}$ 's `\left` and `\right`, `\middle` specifies that the following delimiter is to serve both as a right and left delimiter; it will be set with

spacing appropriate to a right delimiter w.r.t. the preceding atom(s), and with spacing appropriate to a left delimiter w.r.t. the succeeding atom(s).

`\unless`: T<sub>E</sub>X has, by design, a rather sparse set of conditional primitives: `\ifeof`, `\ifodd`, `\ifvoid`, etc. , have no complementary counterparts. Whilst this normally poses no problems since each accepts both a `\then` (implicit) and an `\else` (explicit) part, problems occur when one is used as the final `\if...` of a `\loop ... \if ... \repeat` construct, since no `\else` is allowed after the final `\if...`. `\unless` allows the sense of all Boolean conditionals to be inverted, and thus (for example) `\unless \ifeof` yields `true` iff end-of-file has *not* yet been reached.

## 4 What next?

At the time of writing,  $\epsilon$ -T<sub>E</sub>X version 1 is ready to go to `TeX-Implementors`, although work remains to be done on the `eTrip` test. Whether it will have been released to the implementors before the conference cannot be predicted, since I discovered today that I have only four working days left before I leave for Europe, and it will not be possible to release it once I am away. The version being prepared for the implementors is termed Version 1 $\beta$  (the `NTS` team themselves acted as  $\alpha$ -testers). Once the implementors have given us the go-ahead and said that in their opinion  $\epsilon$ -T<sub>E</sub>X is a viable alternative to T<sub>E</sub>X (by which I mean that it is completely compatible, and functions according to the accompanying documentation), we will release it to the T<sub>E</sub>X world as a whole. We will react as quickly as possible to any bug reports (we sincerely hope that there will be few!), and we will then concentrate on new features for version 2. We certainly intend to work as closely as possible with the `LATEX2 $\epsilon$`  team, not because we believe that `LATEX2 $\epsilon$`  is necessarily right for everybody, but because (a) we respect the intellect and knowledge of the members of the `LATEX2 $\epsilon$`  team, and (b) because it might be possible to enable them to achieve things with `LATEX` and  $\epsilon$ -T<sub>E</sub>X which would either be impossible or extraordinarily difficult with `LATEX` and T<sub>E</sub>X. We have a very long list of suggestions from Nelson, we still have many of Knuth's 'possibly good ideas' to consider, and we have an enormous number of suggestions made on `NTS-L`: we are unlikely to run out of ideas for many years yet!

## 5 Acknowledgements

I would like to thank many people without whom both the project and this paper would simply never have come to fruition. I would like to thank above all Don Knuth, for creating T<sub>E</sub>X and for making it so freely available; for sparing us the time to discuss the project when we met last year at Stanford; and for his willingness to allow us to base  $\epsilon$ -T<sub>E</sub>X on T<sub>E</sub>X. I would also like to thank all those who have contributed ideas, either via personal communication or via the `NTS` list, and I would single out Nelson

Beebe for presenting us with an extremely well thought through and well presented set of proposals. I would like to thank DANTE, without whose financial support we could never have afforded to meet (and experience has shewn that *unless* we meet fairly regularly, very little gets done!). And finally I would like to thank all the members of the team, who have contributed ideas, time, enthusiasm and expertise: they are Joachim Lammarsch (Managing Director), Peter Breitenlohner (project leader,  $\epsilon$ -TEX), Jiří Zlatuška (project leader,  $\mathcal{N}\mathcal{T}\mathcal{S}$ ), Bernd Raichle (2-i/c,  $\epsilon$ -TEX &  $\mathcal{N}\mathcal{T}\mathcal{S}$  projects), and Friedhelm Sowa (color, and user interfaces). And I would like to single out for a special vote of thanks Peter Breitenlohner: without his expertise in TeX.Web, there is absolutely no doubt whatsoever in my mind that this project would *never* have been possible: thank you Don, thank you Peter, thank you everyone.

# Adobe™ Acrobat 2.0™

## Beyond the bounds of paper

Wiegert Tierie

Adobe Systems Benelux B.V.  
Europlaza, Hoogoorddreef 54a  
1101 BE Amsterdam Z.O.  
The Netherlands  
wtierie@adobe.com

### 1 Introduction

Witness an exciting new development in the computer revolution. No longer content to *create* documents better, faster and cheaper, we've begun to look for ways to *use* documents better, faster and cheaper. We are in the midst of a shift from the paper trail to the information superhighway. While concepts like just-in-time and on-demand are familiar in referring to hard goods, we are just beginning to apply them to our truly liquid assets – information. Today, if you can move it, use it or manage it more efficiently than the next guy, you've got a competitive edge.

Consider the following:

- June 1993. The first copies of Acrobat are just rolling off the assembly line. Kenneth Grant and W. David Schwaderer wrote in the Adobe Acrobat Handbook, 'If the IRS used Acrobat, widely accessible income tax preparation forms would print on low-cost home printers, averting taxpayers' frantic trips to copy forms for last-minute filings.'
- Now roll ahead eight months: On February 21, 1994, *Business Week* reported, 'Say the Internal Revenue Service hasn't sent you an 8841 to request a deferred payment. Now, instead of scrambling to the nearest post office or IRS office, you can get digital copies of the forms sent directly to your home computer. CompuServe in Columbus, Ohio has arranged with the IRS and Adobe Systems to provide electronically any of the 450 federal tax forms over its on-line information service. First,

subscribers to CompuServe<sup>®</sup>, which is owned by H&R Block Inc., have to download a copy of Adobe Acrobat software from the network – offered at no charge during an introduction period. Then, by searching for keywords, such as ‘charitable gifts,’ subscribers can find just the form they want and have it sent to their computer. Acrobat then reproduces the form on any printer – including dot-matrix models – almost as they appear on paper from the IRS.’

- President Clinton presented the FY1995 United States Budget to Congress in Acrobat Portable Document Format (PDF) on CD-ROM.
- Tandem Computer sends price lists, data sheets, product brochures and standard forms to field offices in PDF on CD, providing instant access to all product marketing and sales information.
- Over 100 World Wide Web sites offer PDF documents on their Web server in sheets, product information, newspapers, forms, magazines etc.

These real-life scenarios illustrate the growing interest in electronic document software, products that let users create, view and print documents regardless of the computer, software or fonts used by the creator or available on the recipient’s machine. Adobe Acrobat software strikes the best balance between screen and paper versions of documents, with its ability to navigate, link, search, crop and rotate; and with its PostScript™ language-based resolution independence and high-quality output.

## **2 What is Adobe Acrobat?**

Acrobat software puts documents to work electronically. Anybody can create, use, store, share, view or print them. With any computer, any application and any printer.

Acrobat guarantees that the document you create is the document everybody sees. The Portable Document Format (PDF) has the flexibility to describe the content and appearance of documents created with virtually any application for virtually any computer or printer. Acrobat Exchange and Reader are powerful tools for accessing and managing information.

Adobe Acrobat is a family of products – Acrobat, Acrobat Pro, Acrobat for Workgroups and Acrobat Catalog – designed to bring the power of electronic documents to a broad spectrum of users. Each product contains the right collection of components for a specific user’s needs (see ‘Getting Started with Adobe Acrobat’ later in this document).

The Acrobat Reader is the perfect tool for corporate and commercial publishers who distribute documents to large audiences. Acrobat Reader is included with all Adobe Acrobat products and can be freely distributed without charge to others. It allows recipients to view and print any PDF document they receive, and gives them access to all the annotations, bookmarks and links that are part of the PDF file.

The Acrobat Exchange program gives users the power to exchange documents with other Acrobat users. They can create, view, collate, navigate and print PDF documents. PDF Writer creates PDF files from any application by ‘printing’ to a file when a user

selects Print from the application's File menu. Included in Acrobat 2.0 is Acrobat Search, which provides full-text search capabilities for collections of PDF documents. Using Search software, users can quickly search indexed PDF documents for single words or terms, phrases and arbitrary character patterns specified with wildcard characters. They can also perform Boolean searches for documents that contain combinations of words and phrases.

Acrobat Distiller™ software lets individual and network users transform PostScript language files into PDF documents.

The Acrobat Catalog program creates full-text indexes for collections of PDF documents. For each word or term, the full-text index lists the documents and page numbers where the word or term is used. The Acrobat Search software uses full-text indexes to find words and terms in the documents quickly, without having to open the documents.

## **2.1 The flexibility to use any document, anywhere**

Electronic document programs facilitate document distribution for users who are working on other computers, without requiring the recipient to have access to the applications used to create the documents. Not only the text, but formatting, fonts, page layout and graphics appear on-screen precisely as they would if printed on paper.

Electronic documents should be a natural extension of the existing work flow. To an everyday business user, that means working with any word processor, spreadsheet or presentation package and creating an electronic document as easily as selecting the Print command. It means adding value by easily combining elements from multiple applications into a single document, regardless of page orientation, fonts used and so on.

Commercial publishers, writers, graphic artists and engineers create more complex documents using applications and fonts that make use of industry-standard Adobe PostScript technology. Their work flow is different and demands an easy way to turn documents created with the PostScript language into shareable electronic documents, with the ability to fine-tune the file size and quality parameters.

Consumers look for the ability to view and print any document – not just documents they create, but those they receive from a multitude of sources – at the best quality possible. The process should fit seamlessly into the existing work flow and should accommodate any document.

Who needs electronic documents? Anybody for whom the communication, time value or management of information is critical.

Commercial publishers distribute materials electronically with all the visual enhancements that in the past required printed media.

Business organizations exchange proposals and reports electronically, enhanced with charts, diagrams, bold text, bullets and indentation. Price lists, data sheets, contracts and administrative forms are distributed days sooner. Technical documentation, blueprints and research are stored, managed and accessed more efficiently.

Information is central to most organizations, and making information more useful creates a competitive advantage.

## **2.2 More direct access to information**

The power of information lies in the ability to use it. A fundamental benefit of electronic documents is enhanced access. Acrobat provides unparalleled facilities for finding and using information:

- Sophisticated full-text search allows users to find detailed information in thousands of documents – from the desktop, without assistance from reference librarians.
- Cross-document links add a new dimension to information access, linking disparate documents across the network and allowing information to be referenced and updated dynamically.
- Article threads automatically follow the story for ease of on-screen viewing.
- Bookmarks and links take the user to topics of interest.
- Thumbnails provide visual browsing and navigation.

## **2.3 Better ways to manage information**

These days, it seems to be a given that hard disk use and network traffic will expand to exceed whatever capacity is allocated. Adobe Acrobat provides an opportunity to ease some of the strain of managing the varied collections of information that any organization must deal with:

- Documents can be indexed on existing networks without changing the way they are organized or stored.
- Document security controls access and reduces the need for multiple copies of documents in different locations.
- Multi-user notes allow workgroups to review and comment on material electronically.
- Compressed, platform-independent files can be archived on a central file server or distributed on a multi-platform CD-ROM.

## **2.4 An extensible architecture to customize and add value**

Adapting electronic documents to an organization's work flow means tailoring products for unique environments. Acrobat Exchange 2.0 contains a complete set of Application Programming Interfaces (APIs) for creating plug-in modules, customizing the user interface and integrating with other products. Leading systems integrators are incorporating Acrobat products into a wide variety of information systems.

## 2.5 More ways to integrate with other products

In today's resource-conscious market, businesses are looking for ways to create custom plug-and-play solutions from off-the-shelf components. In addition to the comprehensive APIs, Acrobat supports industry-standard interfaces to provide integration and customization features to the broadest set of customers, even nonprogrammers:

- Lotus Notes<sup>®</sup> F/X support to seamlessly share field-level information with today's most popular workgroup application. The information in PDF Document Info fields is used by Lotus Notes to organize their display in Views, giving users a powerful way to browse and find information across the network.
- DDE, OLE 2.0 and OLE automation support. As an OLE server, the Acrobat program integrates with important Windows<sup>™</sup> applications.
- AppleEvent support to connect Acrobat with HyperCard<sup>®</sup>, FileMaker<sup>®</sup>, Microsoft<sup>®</sup> Excel, AppleScript<sup>™</sup>, and many other popular Macintosh<sup>®</sup> applications.

## 2.6 A platform for evolution

The PostScript page description language is a de facto standard that is the foundation of the Acrobat program's Portable Document Format. Device and resolution-independent Adobe PostScript technology ensures that the most important investment – content – will migrate effortlessly into the future. Adobe published the Portable Document Format just as it did the PostScript language definition, to guarantee that information is not locked into a proprietary and transitory format. Choose any platform, monitor or printer today – or ten years from today – and Acrobat documents will take advantage of the benefits they offer.

## 3 Technical overview

The goal of Adobe Acrobat is to reproduce any document (authored yesterday, today or tomorrow) as faithfully as possible, in a form that can easily be viewed, stored and distributed. Simple objective, many obstacles.

First, consider what's necessary to deliver on the goal:

- Create documents that handle any font or graphic, from applications that may no longer be marketed or may not yet be developed.
- Work with the hardware and software that users have or will have.
- Create small files.
- Perform well in networked workgroup environments.
- Assist users in navigating documents and finding the information they need.
- Help manage electronic content.
- Print to any printer – ImageWriter<sup>®</sup> to imagesetter.
- Provide an open, extensible and flexible file format

Now, a brief look at some of the problems and solutions in the areas of fonts, graphics, content independence and compression.

### 3.1 Platform coverage

Adobe Acrobat software is the only electronic document solution that works seamlessly in the DOS, Windows, Macintosh, and UNIX<sup>®</sup> environments. Because all Acrobat products are based on core code that describes a document's content independent of hardware platform or imaging model – and because the format is available as an open, published standard – users are assured that they will always have solutions tailored to their current computing environment.

### 3.2 The key to quality and longevity – content independence

The Adobe Acrobat program's file format is PostScript language-base – not a bitmap image of the page. Text is maintained as actual text characters in a specific font; graphics are maintained as lines and Bézier curves; images are maintained as monochrome, grayscale or color, just as they were in the original document.

Portable Document Format (PDF) is a published standard<sup>1</sup> and does not rely on QuickDraw<sup>™</sup> or GDI.

Why is this important? The market moves too rapidly to track improvements to hardware and software in a timely way – device and resolution independence enable documents to be displayed or printed at the full resolution of next year's device. If you print to an 800 dpi laser printer, your text and graphics will print at 800 dpi. If you zoom in to 538%, the text and graphics will render at exactly that resolution and give you the best quality possible. In addition, a published file format ensures the openness of the solution and enables third-party vendors to embrace Adobe Acrobat and the PDF file format standard to deliver a wide range of solutions based on Adobe Acrobat.

Sure, flexible document handling covers common office documents, the stuff we churn out daily on our trusty word processors and empty into dumpsters weekly. But the flexibility to handle any document means legacy data and 'high-end' graphics, too. The PostScript language is the turf of both power graphics users and technical specialists using sophisticated equation fonts. Some kinds of documents can only be printed to Adobe PostScript output devices. Complex graphic artwork and images in Encapsulated PostScript (EPS) language format must be processed by a PostScript interpreter to reproduce at their full resolution.

How does Acrobat deal with PostScript language files?:

- 'fi' and 'fl' ligatures in PostScript Type 1 fonts
- Handling of PostScript Type 1 fonts included in the PostScript language file
- PostScript Level 2 files, even where PostScript Level 2 is not supported

---

1. Bienz, Tim and Cohn, Richard. *Portable Document Format Reference Manual*. Adobe Systems Incorporated/Addison-Wesley Publishing Company, 1993. ISBN 0-201-62628-4.

- Search for text within an EPS graphic
- Print without loss of quality
- Compare the size of the original PostScript language file to the PDF document
- Recognition of PostScript page-size operators or commands; you shouldn't have to manually change the page size in the Page Setup dialog box each time you want to convert a PostScript language file that is a different size or orientation (Acrobat handles this task automatically)

How does Acrobat handle these experiments? Flawlessly. Not only are all characters in fonts correctly captured and displayed on Macintosh, Windows, DOS and UNIX systems, but they are not stored as bitmaps, so you can display and print them at any resolution. The following screen shot shows a zoomed-in view at 410% (Acrobat is not limited to discrete magnification percentages).

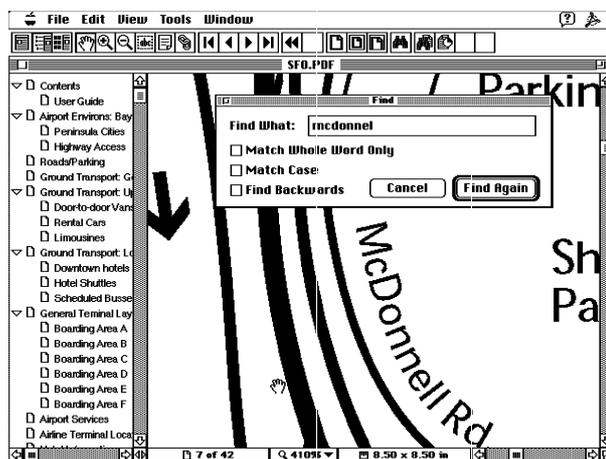


Figure 1: How does Acrobat handle any document you throw at it? Flawlessly.

### 3.3 Font handling

Fonts – their size, style and spacing – are the anchor of a document's appearance. Proper display of text is the toughest single problem in electronic document software.

Even users who have standardized on cross-platform applications and typefaces stumble over fonts when they try to share documents. Characters in fonts are referenced in platform-specific ways. Some characters are unique to a platform, such as 'fi' on the Macintosh. Acrobat handles font encoding intelligently to ensure that all characters are displayed properly regardless of platform. Further, Acrobat is the only electronic document program to offer full support for embedding both TrueType™ and Type 1 fonts. This document's Glossary contains useful information on font terminology.

There are three main approaches to representing fonts in electronic documents:

- Using local fonts
- Creating synthetic fonts (font substitution)
- Embedding the original font

Each approach involves tradeoffs among fidelity, performance and ease of use. Acrobat provides the full spectrum of options to meet the broadest set of user needs.

PDF files contain the names and metric information (called font descriptors) for all the fonts in the document. Adobe Type Manager™ (ATM™) or the native TrueType rasterizer rasterizes the requested font if it is available to the operating system; otherwise it creates substitute fonts based on font descriptor information in the PDF file. The layout, character spacing, graphics and general look of a PDF document are always preserved.

If a source document contains only the fonts installed with the Acrobat viewers, you don't have to worry about whether your readers have the fonts used in your document. But if your documents use other fonts, you can choose to let Acrobat create substitute fonts or to embed the fonts in the PDF file.

#### *Font substitution – for smaller files*

The Acrobat program's font substitution strategy creates synthetic fonts that closely simulate the appearance of unavailable fonts. It preserves exactly the spacing, alignment, stroke weights and other characteristics of the original font metric information stored within the PDF file. During the creation of the PDF file using the Acrobat PDF Writer or Distiller, the font metric information is extracted directly from the font itself (not the ATM font database, as people sometimes assume). Once the PDF has been created, the file itself is completely self-sufficient in terms of font metric information. The font metrics occupy only about 1K of space in the file, while embedding the entire font itself would use 25–30K.

ATM uses two special multiple master fonts to do font substitution: Adobe SansMM and Adobe SerifMM. Not available in application font menus, they were designed specifically to substitute for other fonts. Users can set preferences for substitution fonts to tune resource use versus quality. The default is both serif and sans serif, which uses both Adobe SerifMM and Adobe SansMM for font substitution.

#### *Font embedding – for absolute fidelity*

What about decorative fonts, such as Adobe Wild Type™, which preserve the document look but lose their communication punch when replaced by substituted fonts? Or TrueType fonts, which vary between Macintosh and Windows, and aren't available for UNIX? By embedding the actual font program into the PDF file, Acrobat Exchange/Reader can display the font exactly the same way on Macintosh, Windows, UNIX and DOS.



Figure 2: Acrobat uses the original font metrics and a multiple master font to preserve the characteristics of the original. On top is the original ITC Cheltenham<sup>®</sup> Condensed Bold font; below is the Acrobat substituted font, Adobe SerifMM.

The result? Type 1 or TrueType, simple or elaborate – not only are all characters in the font correctly captured and displayed on Macintosh, Windows, DOS, and UNIX systems, but they can be displayed and printed at any resolution (they are not stored as bitmaps). The following is a zoomed view at 410%.



Figure 3: Acrobat Exchange/Reader for Macintosh. Zoomed-in view (410%) of a document that was converted from a PostScript language file to PDF using Acrobat Distiller. Notice the 'fi' character and the high-quality appearance of the text, even at this odd magnification.

As with font substitution, several options allow users to balance file size with fidelity. New in Acrobat 2.0, the Acrobat Distiller embeds font subsets – only the characters used in the document – to reduce file size and improve viewing performance.

It's useful to understand how Acrobat handles symbolic fonts. During conversion to PDF, the outlines or actual font itself for the symbolic characters are placed into the PDF file. These character outlines are then used for on-screen display in Acrobat

Exchange or Reader. Below are 300-dpi scanned images of the printed output from Microsoft Word and Acrobat Exchange:

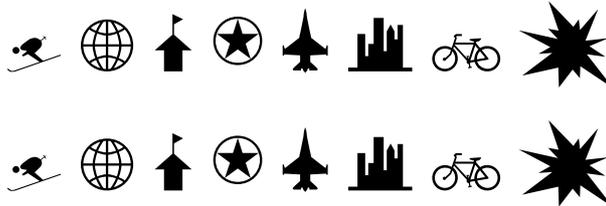


Figure 4: Top, original Word document (printed at 300 dpi) and below, Acrobat Exchange document (printed at 300 dpi)

Acrobat software properly displays symbolic fonts such as Carta™, because all the necessary font information (character shapes and metric information) are part of the PDF file and are guaranteed to be available for on-screen rendering and printing. (Acrobat does not rely on SuperATM™, which does not substitute for symbol fonts.)

Acrobat works with virtually any font; Type 1 and TrueType fonts can be embedded in PDF documents. Type 3 fonts are supported for printing and viewing – bitmaps are included in the PDF file.

Acrobat Exchange and Reader use the original TrueType font if the PDF file was created using the PDF Writer on that platform. The TrueType fonts in any original documents are converted by the driver to PostScript fonts (Type 1 or Type 3) if saved by a PostScript printer driver and converted to PDF by the Distiller. In this case, ATM does font substitution based on the font metric information in the PDF file. These fonts can be embedded by the Distiller.

In effect, you are embedding the Type 1 version of a TrueType font. (This is the case even if the named TrueType font is available to the operating system, because that font is designated as either a Type 1 or Type 3 font in the PDF file.)

### 3.4 File size

Space is precious. A successful electronic document needs to be as small as possible within the confines of the creator's quality requirements. Several factors affect file size.

Compression results in smaller PDF files. Acrobat employs a variety of methods to tailor the file size to user needs:

- Text, graphics (line art) and indexed color images are compressed using the LZW method.
- Color/grayscale images are compressed using the JPEG (Joint Photographic Expert Group) method. This is a lossy compression method; data is removed from the image to produce the compression.

- Image downsampling removes pixel information to decrease resolution and reduce file size.
- Font subsets maintain perfect fidelity by embedding only the characters used in a document.
- Monochrome images are compressed using one of four methods. The default CCITT Group 4 method provides the greatest compression in most cases. CCITT Group 3 compresses monochrome bitmaps one row at a time and is used by most fax machines. LZW produces the best compression for images that contain repeating patterns. Run Length produces the best results for images that contain large areas of solid white or black.

Acrobat 2.0 defaults to a binary file format (retaining the option for ASCII) to further reduce file size and provide more reliable file transfer in certain environments.



Figure 5: A typical Microsoft Word file and the resulting intermediate PostScript file.

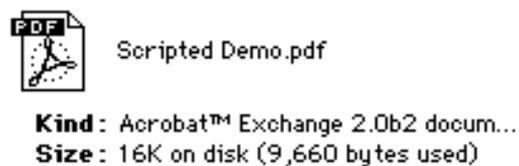


Figure 6: The resulting PDF file is less than half the size ( $9,660/21,595 = 45\%$ ) of the native Word file.

Where appropriate to the content, Acrobat 2.0's cross-document linking capability allows information to be broken up into separate files with links to related information.

## 4 Getting started with Adobe Acrobat

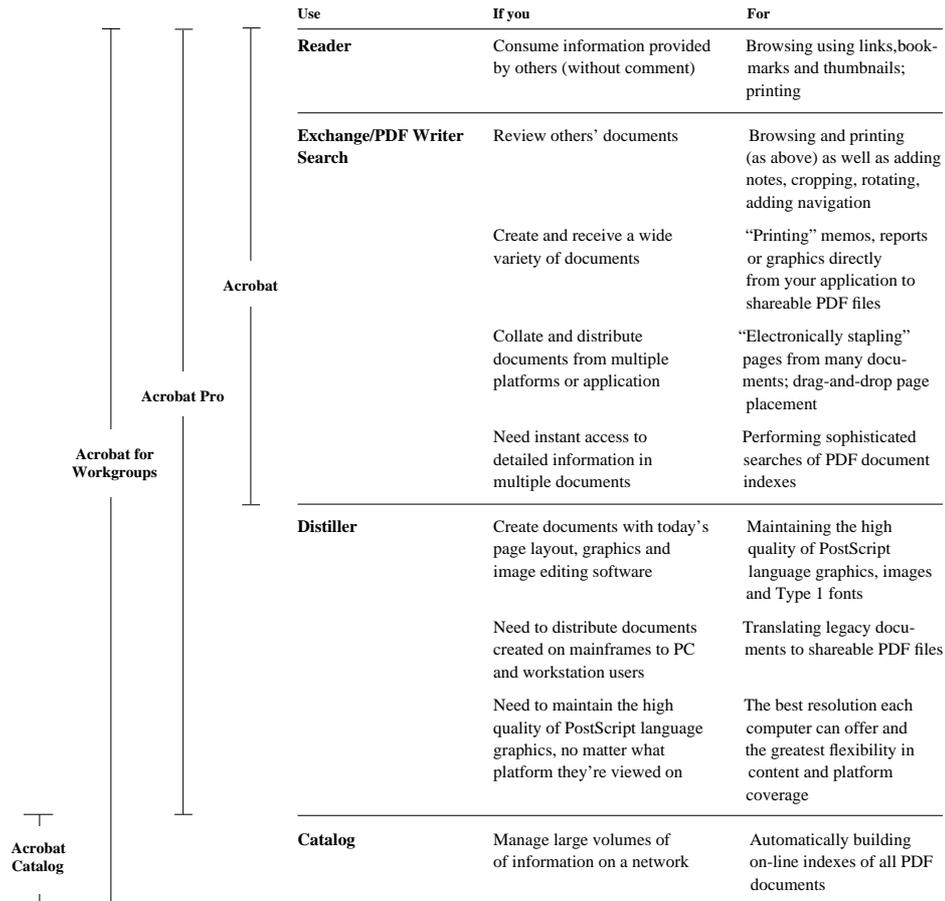


Figure 7: An overview of the Adobe Acrobat product family

### 4.1 Acrobat Exchange (included in all Acrobat products)

The Acrobat Exchange program gives you the power to exchange documents with other Adobe Acrobat users. With it, you can create, view, collate, navigate and print PDF documents. PDF Writer creates PDF files from any application by 'printing' to a file when you select Print from the application's File menu. Included in Acrobat 2.0 is Acrobat

Search, which provides full-text search capabilities for collections of PDF documents. Using the Search software, you can quickly search indexed PDF documents for single words or terms, phrases, or arbitrary character patterns specified with wildcard characters. You can also search for documents that contain combinations of words and phrases.

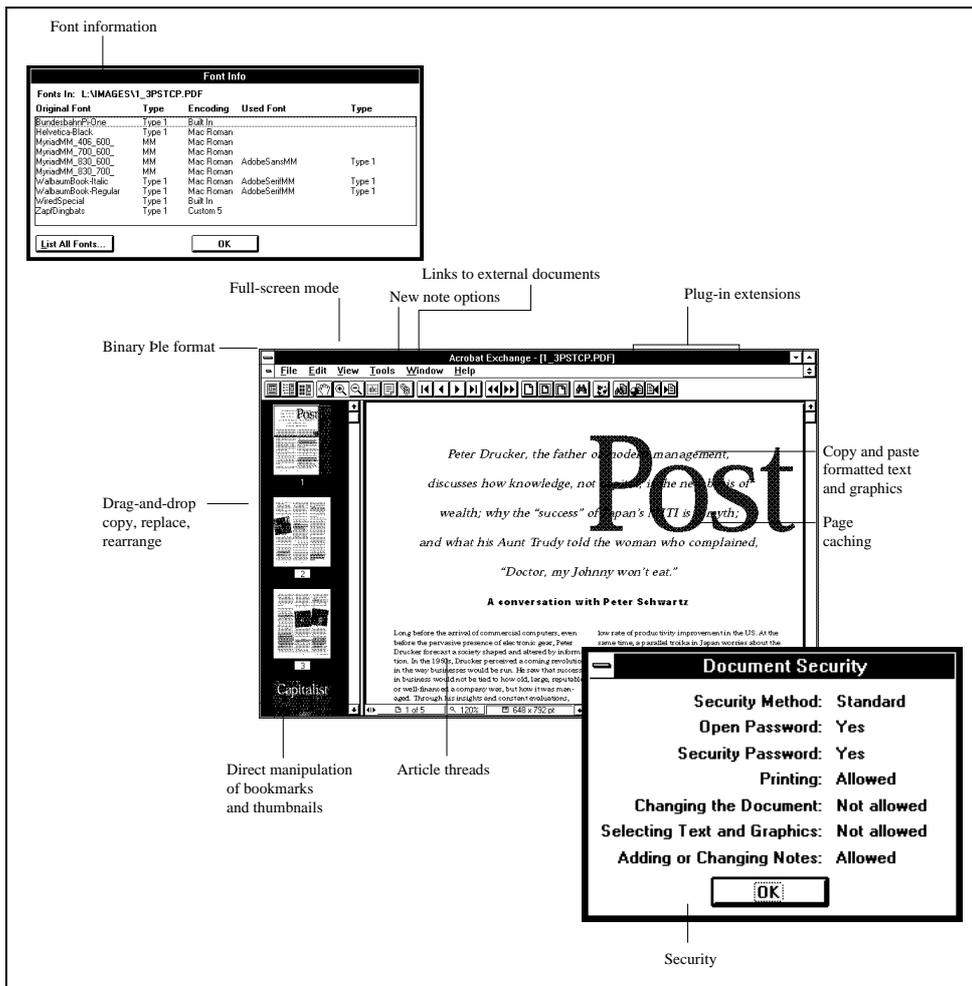


Figure 8: Acrobat 2.0 at a glance.

## 4.2 Acrobat Search (included with Acrobat 2.0)

Acrobat Search software gives you full-text search capabilities for collections of PDF documents that have been indexed with the Acrobat Catalog program. Acrobat Search

uses full-text indexes to find words and terms in the documents quickly, without having to open the documents.

Using Acrobat Search, you can quickly find indexed PDF documents for single words or terms, phrases or arbitrary character patterns specified with wildcard characters. You can also perform Boolean searches for documents that contain combinations of words and phrases. Unlike most other full-text search products, Acrobat Search software can find words and terms in illustrations, graphs and formatted tables. It can even find words that are rotated or attached to a curved line.

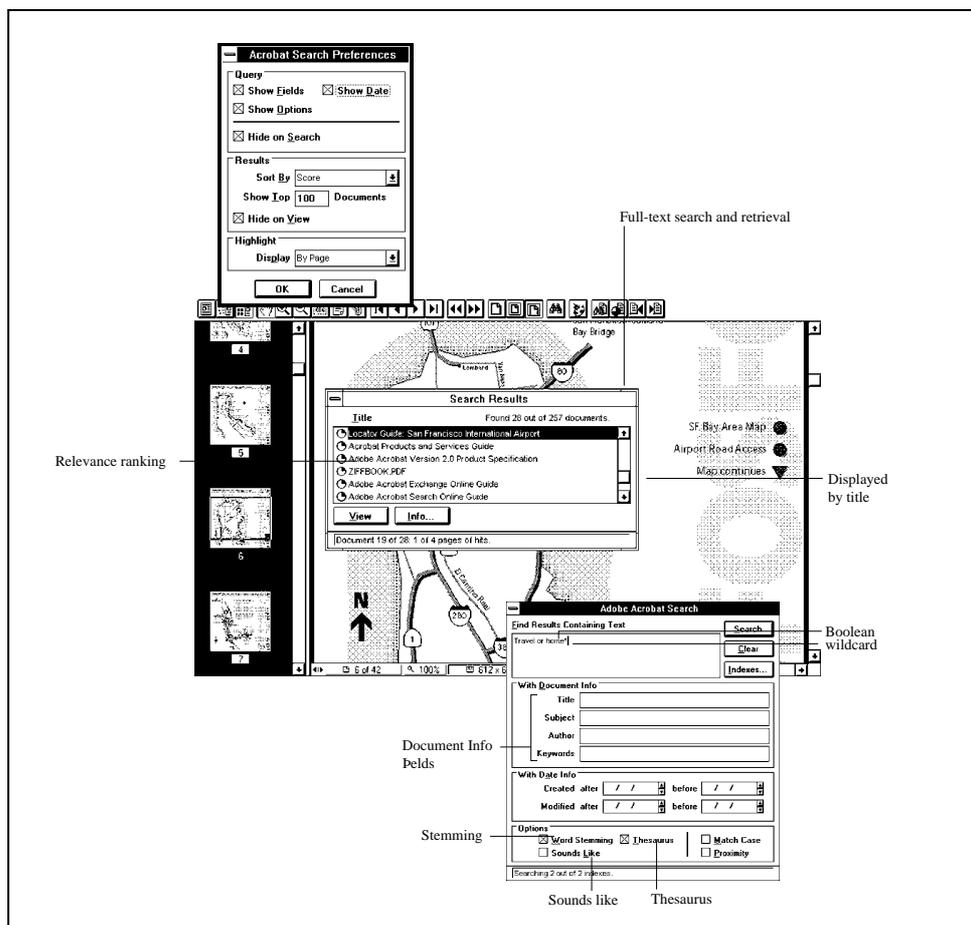


Figure 9: Acrobat Search provides powerful data access.

### 4.3 Acrobat Distiller (included with Acrobat Pro and Acrobat for Workgroups)

The Acrobat Distiller program creates PDF files from virtually any document that has first been saved as a PostScript language file. Distiller is recommended for high-quality reproduction of EPS artwork, 24-bit images and documents that take advantage of features (such as blends) that are only available on Adobe PostScript printers.

### 4.4 Acrobat Catalog (included with Acrobat for Workgroups)

The Acrobat Catalog program creates full-text indexes for collections of PDF documents. A full-text index (also referred to as an 'inverted word list') is an alphabetized list of all the words and terms that are used in a collection of documents. You can, for example, build an index for all the PDF documents on a CD-ROM. You can also point Acrobat Catalog at your largest network file server and automatically build an on-line index for all the PDF documents. As documents are added, modified or removed, the index is updated incrementally to ensure that you always search against the files currently on the network. When used with indexes built for large PDF document collections, Acrobat Search enables users to search hundreds or thousands of documents in seconds.

## 5 Glossary

**article threads** Documents in which text is in columns or otherwise separated and can be 'threaded' so that the reader can follow the flow. Acrobat Exchange automatically scrolls, centers the view and travels to additional pages in the thread.

**bookmarks** Used to mark parts of a document for quick access. Bookmarks are listed in the overview area of the Exchange window to locate hard-to-find or often-used information, or to create a custom outline of a document. Clicking a bookmark displays the location marked by that bookmark. Bookmarks can have the same actions as links (see that entry).

**CCITT** An abbreviation for the International Coordinating Committee for Telephony and Telegraphy standards body.

**CCITT Group 3** A lossless compression method used by most fax machines that compresses monochrome bitmaps one row at a time.

**CCITT Group 4** A lossless compression method used by fax machines and applications that compresses groups of monochrome bitmaps.

**downsampling** Deletes pixel information in an image to decrease resolution and reduce file size.

**Encapsulated PostScript (EPS)** An EPS file contains PostScript page description language information and maintains the full image quality across applications and output devices.

**font encoding** Characters in fonts are referenced in different ways depending on the computer platform. The way Windows 3.1 finds a lowercase 'a' in a font is different

from the way Macintosh or Sun™ workstations find the lowercase 'a' in the same font.

**font formats** Include Type 1 fonts, Type 3 fonts, multiple master fonts, TrueType fonts.

**font metrics** Specify character widths so applications can determine line lengths. There are two general types: integer metrics and fractional metrics. Font metrics also store kerning information. Kerning is an optional adjustment factor for the spacing between two characters. A positive kerning value moves characters apart; a negative value moves characters closer together.

**font substitution** PDF files carry information about the fonts used in a document without actually including the font. Acrobat substitutes a multiple master font to maintain the look and feel of the original document.

**full-text index** An alphabetical list of all the words and terms used in a collection of documents. Also referred to as an inverted word list.

**JPEG (Joint Photographic Expert Group)** Provides significant compression of color and grayscale images. It is a lossy compression method; data is removed from the image to produce the compression.

**ligature** Two or more letters tied into a single character. The sequences 'fi' and 'fl', for example, form ligatures in most serif typefaces.

**links** 'Hot' areas or text on a document page that the user clicks to travel to an associated destination. Links can go to another view or page in the document or another PDF document; open another file or program; or perform an arbitrary action defined by a plug-in.

**LZW (Lempel-Ziv-Welch)** A lossless compression method that is useful for data containing repeating patterns.

**multiple master fonts** Allow automatic copy fitting by starting with predetermined base designs and manipulating width, weight and optical scaling to create all other occurrences of the font.

**page caching** Storing already viewed page images in memory to improve display performance.

**Portable Document Format (PDF)** The key to cross-platform functionality of Adobe Acrobat products is a unique PostScript language-based file format. PDF is an open standard that Adobe Systems documents and publishes for use by software developers.

**proximity** The Proximity search option changes the way found documents are assigned a relevance ranking. With the Proximity option selected, the closer the words are together in a document, the higher the relevance ranking.

**Run Length Encoding** A compression method that produces the best results for images containing large areas of solid white or black.

**sounds like** An Acrobat Search option that finds incorrect spellings of a search word. Searching for misspelling, for example, also finds misspelling.

**stopwords** Terms that are excluded from a full-text index. Common examples are articles, conjunctions and prepositions.

**thesaurus** An Acrobat Search option that finds words that have the same meaning as the search word. Searching for *crypt*, for example, also finds *mausoleum*, *sepulcher* and *tomb*.

**thumbnails** Miniature pages in the overview area of the Acrobat Exchange window that allow users to jump quickly to a page; adjust the view of the current page; and move, copy and replace pages in a PDF document.

**TrueType fonts** Come in two forms on the Macintosh: bitmap and outline in one unit, or bitmap only in one unit. They are converted to Type 1 or Type 3 fonts by the PostScript printer driver when a PostScript language file is generated.

**Type 1 fonts** Actually PostScript language programs, they were the first outline fonts. They are called outline fonts because each character is described mathematically as a collection of lines and curves that form the character's outline.

**Type 3 fonts** Usually bitmap format instead of outline format. They are usually not hinted and tend to have lower quality than Type 1 fonts. Type 3 bitmaps are included in the PDF file. Type 3 fonts are PostScript fonts that do not work with ATM; they allow use of the full PostScript language, and ATM is not a full PostScript interpreter. Type 3 fonts are not hinted, resulting in poorer quality for screen display and when printing at small point sizes, and can exist in bitmap or outline form. They are useful for complex or ornamental fonts or logos, but they require more time for a printer to image.

**wildcard characters** Characters used in a search to find all the words that contain a word fragment, or all the words and terms that match an arbitrary character pattern. The wildcard characters are the asterisk, which matches zero, one or more characters; and the question mark, which matches any one character.

**word stemming** An Acrobat Search option that finds all the words with a common stem. With word stemming enabled, searching for *manager* also finds *manage*, *managed* and *managing*.

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Adobe Type Manager, ATM, Carta, Distiller, PostScript, SuperATM and Wild Type are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions. ITC Cheltenham is a registered trademark of International Typeface Corporation. Apple, ImageWriter and Macintosh are registered trademarks and AppleScript, QuickDraw and TrueType are trademarks of Apple Computer, Inc. FileMaker and HyperCard are registered trademarks of Claris Corporation. Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd. Sun is a trademark of Sun Microsystems, Inc. QuarkXPress is a registered trademark of Quark, Inc. CompuServe is a registered trademark of CompuServe Incorporated. Lotus Notes is a registered trademark of Lotus Development Corporation. All other brand or product names are trademarks or registered trademarks of their respective holders.

Adobe Systems Incorporated	Adobe Systems Benelux B.V.
1585 Charleston Road	Europlaza
P.O. Box 7900	Hoogoorddreef 54a
Mountain View	1101 BE Amsterdam Z.O.
California 94039-7900	The Netherlands
USA	

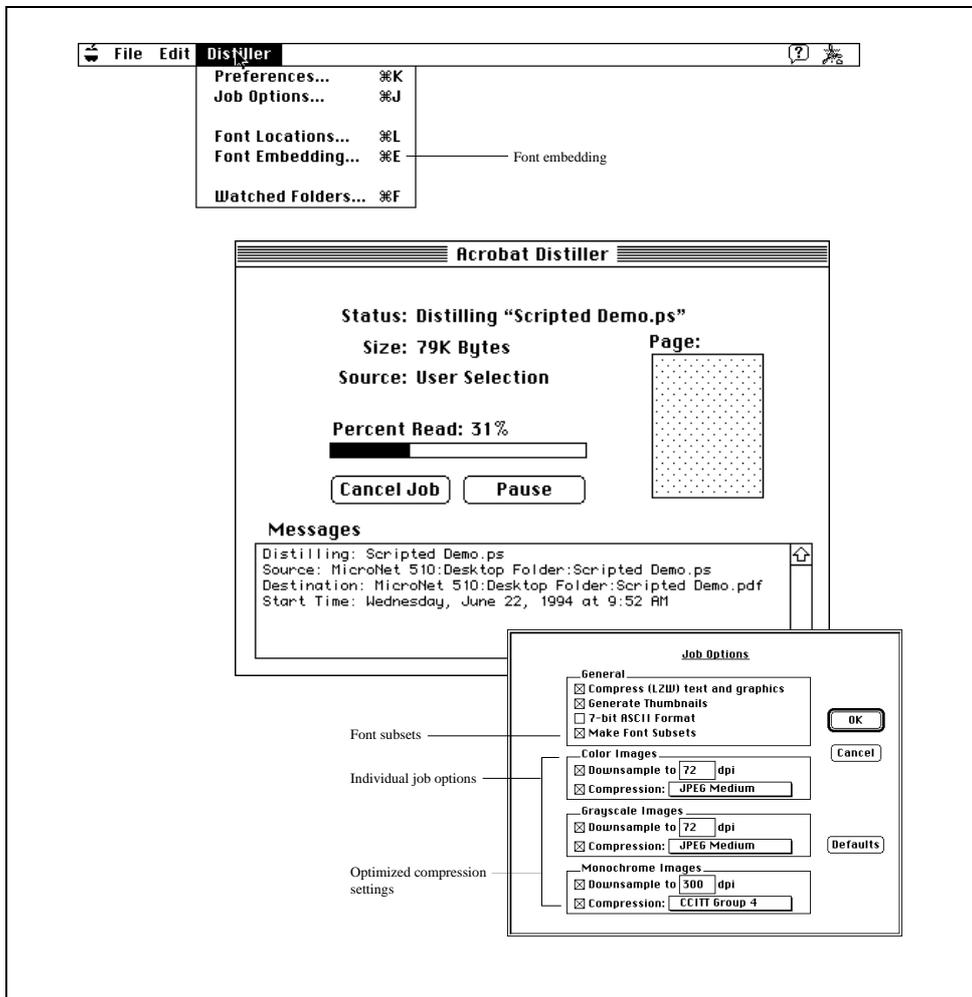


Figure 10: Acrobat Distiller creates PDF files from virtually any document – created on DOS, Windows, Macintosh, UNIX or mainframe machines.

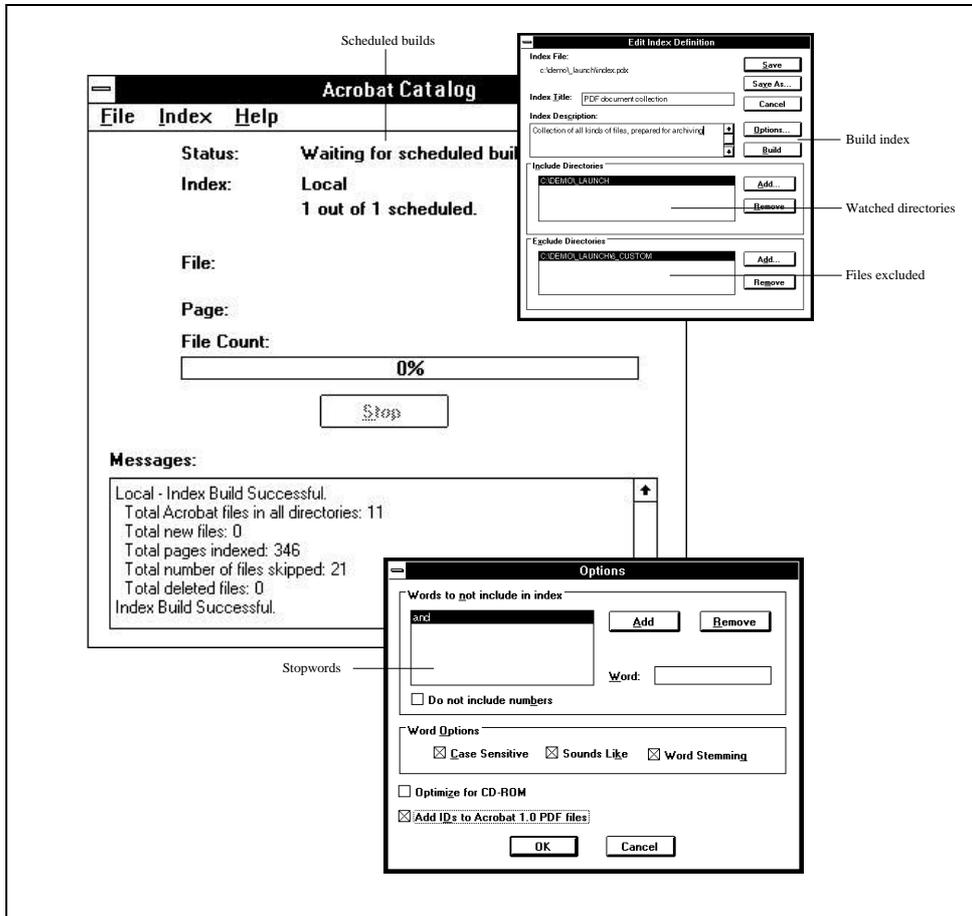


Figure 11: Acrobat Catalog manages large volumes of information on a network.

# Typesetting commutative diagrams

Gabriel Valiente Feruglio

University of the Balearic Islands  
Mathematics and Computer Science Dept.  
E-07071 Palma de Mallorca (Spain)  
dmigva0@ps.uib.es

## Abstract

There have been several efforts aimed at providing  $\text{\TeX}$  and its derivatives with a suitable mechanism for typesetting commutative diagrams, with the consequent availability of several macro packages of widespread use in the category theory community, and a long debate about the best syntax to adopt for commutative diagrams in  $\text{\LaTeX3}$  has taken place during 1993 in the `CATEGORIES` discussion list. From the user's point of view, however, there is not much guidance when it comes to choosing a macro package, and even after a decision is made, the conversion of diagrams from the particular conventions of a macro package to another macro package's conventions may prove to be rather hard.

Typesetting commutative diagrams is a surprisingly difficult problem, in comparison with  $\text{\TeX}$  macro packages for other purposes, as judged by the amount of code needed and years of development invested. The existing macro packages for typesetting commutative diagrams are reviewed in this paper and they are compared according to several criteria, among them the capability to produce complex diagrams, quality of the output diagrams, ease of use, quality of documentation, installation procedures, resource requirements, availability, and portability. The compatibility of the different macro packages is also analyzed.

## 1 Introduction

Commutative diagrams are a kind of graphs that are widely used in category theory, not only as a concise and convenient notation but also as a powerful tool for mathematical thought.

A diagram in a certain category is a collection of nodes and directed arcs, consistently labeled with objects and morphisms of the category, where 'consistently' means that if an arc in the diagram is labeled with a morphism  $f$  and  $f$  has domain  $A$  and codomain  $B$ , then the source and target nodes of this arc must be labeled with  $A$  and  $B$  respectively.

A diagram in a certain category is said to *commute* if, for every pair of nodes  $X$  and  $Y$ , all the paths in the diagram from  $X$  to  $Y$  are equal, in the sense that each path in the diagram determines through composition a morphism and these morphisms are equal in the given category. For instance, saying that the diagram<sup>1</sup>

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ g \downarrow & & \downarrow g' \\ C & \xrightarrow{f'} & D \end{array}$$

commutes is exactly the same as saying that

$$g' \circ f = f' \circ g.$$

As a notation, the graphic style of presentation inherent to commutative diagrams makes statements and descriptions involving categories more clear and manageable than textual presentations. For instance, consider the definition of an equalizer. A morphism  $e : X \rightarrow A$  is an *equalizer* of a pair of morphisms  $f : A \rightarrow B$  and  $g : A \rightarrow B$  if  $f \circ e = g \circ e$  and for every morphism  $e' : X' \rightarrow A$  satisfying  $f \circ e' = g \circ e'$  there exists a unique morphism  $k : X' \rightarrow X$  such that  $e \circ k = e'$ .

An equivalent definition is that  $e$  is an equalizer if the upper part of the diagram

$$\begin{array}{ccccc} X & \xrightarrow{e} & A & \begin{array}{c} \xrightarrow{f} \\ \xrightarrow{g} \end{array} & B \\ \uparrow k & \nearrow e' & & & \\ X' & & & & \end{array}$$

commutes and, whenever the lower part of the diagram also commutes, there is a unique  $k$  such that the whole diagram commutes.

As a tool for thought, proofs involving properties that are stated in terms of commutative diagrams can often be given in a 'visual' way, in what has been called *diagram chasing*. For instance, the proposition that if both inner squares of the following diagram commute, then also the outer rectangle commutes,

$$\begin{array}{ccccc} A & \xrightarrow{f} & B & \xrightarrow{g} & C \\ a \downarrow & & \downarrow b & & \downarrow c \\ A' & \xrightarrow{f'} & B' & \xrightarrow{g'} & C' \end{array}$$

can be proven as follows:

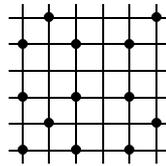
1. All the diagrams in this paper have been typeset using the Xy-pic macro package, unless otherwise stated. The reader should not infer any preference by the author for that particular macro package, but should understand that some macro package is needed for the examples in the paper. Sample diagrams typeset with the other macro packages are given in Appendix I.

$$\begin{aligned}
(g' \circ f') \circ a &= g' \circ (f' \circ a) && \text{(associativity)} \\
&= g' \circ (b \circ f) && \text{(commutativity} \\
&&& \text{of left square)} \\
&= (g' \circ b) \circ f && \text{(associativity)} \\
&= (c \circ g) \circ f && \text{(commutativity} \\
&&& \text{of right square)} \\
&= c \circ (g \circ f) && \text{(associativity)}.
\end{aligned}$$

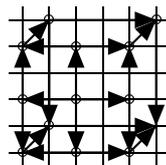
Commutative diagrams<sup>2</sup> range from simple, rectangular matrices of formulae and arrows to complex, non-planar diagrams with curved and diagonal arrows of different shapes.

## 2 Constructing commutative diagrams

Commutative diagrams are constructed in most cases as rectangular arrays, as Donald Knuth does in Exercise 18.46 of [4]. The objects or vertices are set much like a `\matrix` in `TEX` or an `array` environment in `LATEX`,



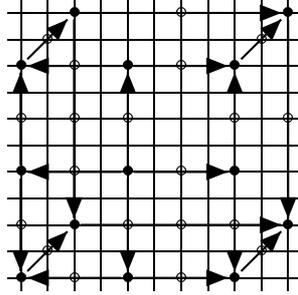
and the morphisms or arrows are set either right after the vertex where they start,



or in a cell on their own,

---

2. The epithet ‘commutative’ is traditional and it is originated in the fact that diagrams may be used to display equations such as the commutative and associative laws. Although not all such diagrams which people draw commute in the formal sense given, this paper adheres to tradition and all such diagrams are called commutative diagrams herein.



depending on the macro package being used, where the grids correspond to the sample diagram presented in Appendix I. (Sketching a commutative diagram on such a grid on paper may prove to be a mandatory step before typing the actual diagram, at least for all but the simplest diagrams.) This gives a first distinction,

- one object and all departing morphisms in each non-empty cell, or
- either one object or one or more morphisms in each non-empty cell.

Whether they belong together with their source object in a cell or they use a cell on their own, morphisms are specified by the address of their target cell. Such addresses can be implicit, absolute or relative to the source cell, and they can be either arbitrary or limited by the available diagonal slopes.

Moreover, some macro packages even support symbolic addresses, by which nodes are tagged with identifier names and arrows are specified by making reference to the names of their source and target nodes. This is a step forward in the sense of the L<sup>A</sup>T<sub>E</sub>X principle of emphasizing structural descriptions, and in fact it is of great help for designing complex diagrams because it divides the task into two separate subtasks, the one of producing a correct and elegant arrangement of nodes and the other one of laying out the correct arrows and positioning their labels.

### 3 Evaluation guidelines

The following aspects are considered in the next section for each of the macro packages in turn. The spirit of these guidelines is to give the potential user a feeling of what to expect from a macro package for typesetting commutative diagrams, and they are based on the experience of the author during the last few years, as user of some of the macro packages.

#### 3.1 Arrow styles

The arrows used in commutative diagrams often are of different shapes, in order to distinguish different kinds of morphisms such as monomorphisms, epimorphisms, isomorphisms, and inclusions, to name just a few, and sometimes they have a shaft other

than a solid line, for instance dashed or dotted, to indicate that it is the existence of the corresponding morphisms what is being characterized.

A collection of built-in arrow shapes and shafts is included in every macro package, and some macro packages even provide facilities for defining new arrow styles, for instance by defining a new control sequence name and choosing a particular combination of tail (the piece that appears at the source end), head (the piece that appears at the target end), and shaft, from a predefined palette of possible heads, tails, and shafts.

### 3.2 Automatic stretching

Most of the macro packages provide for the automatic stretching of arrows to meet their source and target nodes, where meeting a node means to get as close to the (rectangular) box enclosing the node as dictated by some predefined parameter.

While it may be appropriate for most horizontal and vertical arrows, in the case of diagonal arrows it may leave the arrow too far from the node, and extra diagram fine-tuning (see below) is needed in such cases in order to get the arrow closer to the node. The macro package by John Reynolds, however, incorporates basic facilities for associating an hexagon, octagon, or diamond to a node, instead of the usual rectangle, although it does not exploit them in the macros for commutative diagrams.

### 3.3 Diagram fine-tuning

Given a correct description of the structure, a macro package has the task of choosing the best possible arrows to produce the commutative diagram. Sometimes the best choice may not seem good enough, because only a limited number of slopes may be available for the arrows, because arrows may cross, and because arrow labels may superimpose. Manual fine-tuning belongs therefore to producing complex commutative diagrams.

Arrow stretching can be regarded as automatic fine-tuning. Manual fine-tuning facilities, on the other hand, include moving labels around, moving arrows around, modifying their size, changing the distance from the source node to the beginning of the arrow, as well as from the end of the arrow to the target node, and setting spacing parameters such as the gap between columns and between rows. Some macro packages provide the facility to adjust these gaps to different values between specific rows or columns, which is essential in order to get the proper perspective of a three-dimensional diagram. Otherwise, empty rows and columns have to be added to the diagram to get the desired perspective. Appendix III shows the degree of automatic stretching provided by each of the macro packages.

### 3.4 Installation

None of the macro packages requires a complex installation procedure, and in most cases the only requirement in order to get the package running is to drop a single macro or style file somewhere in the  $\text{T}_{\text{E}}\text{X}$  search path. Some macro packages, however, have

accompanying special fonts to get better diagonal lines and arrows, that is, they provide more diagonal slopes and a wider variety of arrow heads and tails to choose from.

In such a case, installation can get more complicated. METAFONT is not as easy to drive or as familiar to the user as T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X; many implementations do not make it available, and on others only the system administrator is able to install fonts. A ready-to-use collection of the additional fonts at standard magnifications is distributed, however, with some macro packages.

### 3.5 Documentation

Ranges from small text files to comprehensive user guides, and even to book chapters.

### 3.6 User support

The authors of the different macro packages have been receptive to comments and willing to provide user support. Almost all of the macro packages remain under development and are open to suggestions from users. Moreover, further development of the X<sub>Y</sub>-pic macro package by Kristoffer Rose and Ross Moore is being funded by three different sources.

### 3.7 Ease of use

The relative ease of use of a macro package is a subjective matter, depending to a large extent on previous experiences in using similar macro packages. Nevertheless, there are at least two characteristics of a macro package for typesetting commutative diagrams that are worth mentioning. The way in which the array of cells underlying a commutative diagram has to be conceived is of most importance. The requirement, found in some macro packages, of extra cells for morphisms makes the macro package much more difficult to use, because the user has to add many spurious rows and columns only to hold these morphisms and to get proper spacing, and the code for the diagrams gets bigger and more obscure (compare the last two grids in the previous section).

Orthogonal to the conception of the array of cells is the way in which coordinates for the source and target nodes of the arrows have to be specified. While such addresses are implicit in the name of the arrow in some macro packages, they are absolute coordinates, coordinates relative to the cell where they are declared, or even symbolic coordinates in other macro packages.

The other aspect is the degree of manual fine-tuning needed to achieve a readable commuting diagram. Even when the macro package provides enough facilities, fine-tuning a complex commutative diagram may take more time and effort than conceiving, designing, and coding the whole diagram. Some of the macro packages require visual or measured adjustment by the user of the size and position of every node, arrow, and label, whereas for others most diagrams may be input as easily as any other mathematical formula in T<sub>E</sub>X and they are typeset nicely without any manual adjustment at all.

### 3.8 Resource requirements

It is well known that  $\text{T}\text{E}\text{X}$  has been designed to support high-quality typesetting of mathematical text, and that it does not offer much built-in support when it comes to drawing and performing arbitrary computations. Because most of the macro packages are built on top of  $\text{T}\text{E}\text{X}$ , they are forced to resort to indirect ways of performing computations and to produce large diagrams by juxtaposition of small line and arrow segments. Therefore, a complex diagram may take up lots of computations, line segments, words of  $\text{T}\text{E}\text{X}$  memory, and time to typeset. Appendix IV compares resource requirements for the different macro packages, showing the main file size together with statistics of both total time and marginal time. The statistics are based on sample runs to typeset the sample diagrams presented in Appendix I with the different macro packages.

### 3.9 Availability

All the macro packages reviewed in this paper can be found in the CTAN archives, and either are in the public domain or are free software, subject to the terms of the GNU General Public License as published by the Free Software Foundation. They are listed in Appendix V.

### 3.10 Compatibility

Converting a commutative diagram among different macro packages is no straightforward task, not only because of the different approaches to constructing a diagram mentioned in the previous section, but also because of differences in naming conventions and in the available arrow styles and slopes. Converting the sample diagram in Appendix I has taken the author many hours of careful work, and in some cases building the diagram again from scratch for another macro package has proven to be the most efficient solution.

The macro packages are therefore highly incompatible. Nevertheless, the macro package by Paul Taylor provides some initial facilities for emulating other macro packages. Maybe a common, agreed-upon syntax for commutative diagrams (see the last section below) would provide a suitable framework for solving these incompatibilities. Moreover, although it may seem rather natural that the macro packages are not compatible with each other, because the idioms are under development and none of the authors is, in principle, under any obligation to the users of the other macro packages, the adoption of a common standard would have the advantage to the whole user community that the diagrams which have already been drawn with one macro package could be pasted into a document using another macro package.

### 3.11 $\text{T}\text{E}\text{X}$ format requirements

While it would be desirable to be able to typeset a commutative diagram under any derivative of  $\text{T}\text{E}\text{X}$ , some macro packages can only run on  $\text{L}\text{A}\text{T}\text{E}\text{X}$  because they borrow the `picture` environment and one or more of the special fonts `line10`, `linew10`, `circle10`,

and `circlew10`. Other macro packages require  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{T}\text{E}\text{X}$  or the `amstex` package in  $\text{L}\text{A}\text{T}\text{E}\text{X}$ . The other way round, some macro packages run on  $\text{T}\text{E}\text{X}$  but do not run when used in a  $\text{L}\text{A}\text{T}\text{E}\text{X}$  document.

### 3.12 Output quality

This is perhaps the most subjective aspect in these guidelines, and therefore it is left for the reader to evaluate. See the sample diagrams in Appendix I, and make a guess at which of the macro packages has been used in Valiente (1994).

## 4 Macro packages

The different macro packages are listed in turn in the following, under the name of the respective author, and they are analyzed according to the evaluation guidelines presented in the previous section. No attempt has been made to put them in chronological order of development, and the list is sorted by author name.

### 4.1 American Mathematical Society

$\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{T}\text{E}\text{X}$  includes some commands for typesetting commutative diagrams, which are also available in  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{L}\text{A}\text{T}\text{E}\text{X}$  as a separate option. Only horizontal and vertical arrows are supported, and therefore  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{T}\text{E}\text{X}$  can only handle ‘rectangular’ commutative diagrams. Moreover, only ‘plain’ arrows can be used within commutative diagrams, although  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{T}\text{E}\text{X}$  provides about 30 different arrow shapes, and arrows do not automatically stretch to their source and target vertices. Commutative diagrams are specified as an array of cells, with either one object or one or more morphisms in each non-empty cell, although unlike matrices, no column separator is needed (a special delimiter has to be used, however, in place of missing arrows). Arrow coordinates are implicit in the name of the arrow and only the four basic directions are available, where arrows can only extend to the adjacent row and/or column in the array. The only fine-tuning facilities provided are a stretching command to force arrows in the same column to be set to the same length (actually, to the width of the longest label in that column), which does not suffice in order to achieve appropriate arrow stretch when the vertices have different width (this manual stretching facility requires the whole `amstex` package to be loaded in  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{L}\text{A}\text{T}\text{E}\text{X}$ ), and a command to change the minimum arrow width in a diagram, for instance to get it to fit on a page. Documentation is as scarce as the facilities the package provides, only four pages in [11] and one page in [9].

### 4.2 Barr

Instead of using a matrix notation, commutative diagrams are specified in the macro package developed by Michael Barr by composing more elementary diagrams, using primitive shapes such as squares and triangles. Arrow coordinates are implicit within

these shape macros. Additional arrows can be specified by giving the absolute address, within an implicit `picture` environment, of their source node, together with the relative address of their target node as a slope and a length, but stretching is not automatic in these cases. It supports diagonal arrows only in the usual  $\text{\LaTeX}$  slopes, and only a few different arrow shapes are available. There are no facilities for diagram fine-tuning. It only runs on  $\text{\LaTeX}$ . Documentation consists of a 10-page document [1] which explains the principles and gives detailed examples.

### 4.3 Borceux

In the macro package developed by Francis Borceux, commutative diagrams are specified as an array of cells, with one object and all departing morphisms in each non-empty cell. There are facilities for introducing one object and one morphism, or two *crossing* morphisms, in each non-empty cell, but at most two items may belong to the same cell. The delimiter for columns is, unlike the `&` character used in *all* the other macro packages, the special character `§` that is not even available in many keyboard layouts. It supports diagonal arrows of different shapes and in many different, although not arbitrary, slopes, and it also supports parallel and adjoint (counter-parallel) arrows, some curved arrows, and automatic stretching. Arrow coordinates are implicit in the name of the arrow for the 32 principal directions. Different facilities for diagram fine-tuning are provided. It only runs on  $\text{\LaTeX}$ . Documentation consists of a detailed 12-page document [2]. Two restricted macro files are distributed for small  $\text{\TeX}$  implementations, one that only allows for plain arrows and another one that also provides parallel and adjoint (counter-parallel) plain arrows. A further macro file is distributed with the package that provides additional triple, quadruple, and quintuple arrows, parallel and disjoint.

### 4.4 Gurari

Unlike the case of most of the other macro packages, Eitan Gurari has developed a general drawing package on top of  $\text{\TeX}$ . It supports diagonal arrows of different shapes and arbitrary slopes, curved arrows and loops, automatic stretching, and symbolic addressing. Arrow coordinates can be symbolic, because of the possibility of naming any location within a drawing, but they are relative in the sample diagrams presented in the appendices because the macros used are the ones given in page 160 of [3]. It runs on both  $\text{\TeX}$  and  $\text{\LaTeX}$ . The macros are well documented in the book, with several basic chapters and one chapter devoted to general grid diagrams, but there is only one page describing commutative diagrams and there is only one sample diagram in the whole book.

### 4.5 Reynolds

John Reynolds has developed a macro package consisting of a collection of general macros for producing a wide variety of diagrams and another collection of macros, which depend on the general macros, for producing commutative diagrams. It supports

diagonal arrows only in the usual  $\text{\LaTeX}$  shapes and slopes, because the macros depend on the  $\text{\LaTeX}$  picture facilities to draw lines, arrows, and circles, although it also supports parallel and adjoint (counter-parallel) arrows, loops, and it provides automatic stretching. Commutative diagrams as specified by giving the absolute coordinates for each node and for the source and target node of each arrow, an approach close to symbolic addressing. Excellent facilities for diagram fine-tuning are provided. It only runs on  $\text{\LaTeX}$ . Documentation consists of a rather cryptic 12-page ASCII file [5] describing the macro package, together with a  $\text{\LaTeX}$  input file that produces a 7-page document of sample diagrams.

#### 4.6 Rose

A macro package has been developed by Kristoffer Rose on top of a more general drawing language, called the *XY-pic kernel*. It supports diagonal arrows of different shapes and in many different, although not arbitrary, slopes, and it also supports parallel and adjoint (counter-parallel) arrows, curved arrows, and loops. Arrows stretch automatically, and there are ample facilities for defining additional arrow styles. Commutative diagrams are specified as an array of cells, with one object and all departing morphisms in each non-empty cell. Arrow coordinates for the target node are implicit in the name of the arrow for the 16 principal directions, and they can be absolute or relative for all other directions. Different facilities for diagram fine-tuning are provided. It runs on both  $\text{\TeX}$  and  $\text{\LaTeX}$ . Documentation is excellent, both a comprehensive guide [6] and a more technical document [7] are provided with the package. The latter also describes the *XY-pic kernel*.

#### 4.7 Smith

The *Expanded Plain  $\text{\TeX}$*  macro package includes macros for typesetting commutative diagrams, written by Steven Smith, in a file named `arrow.tex`. It supports diagonal arrows only in the usual  $\text{\LaTeX}$  slopes, because the macros depend on the  $\text{\LaTeX}$  font `line10`, and only a 'plain' arrow shape is available, besides pairs of parallel and adjoint (counter-parallel) arrows. Commutative diagrams are specified as an array of cells, with either one object or one or more morphisms in each non-empty cell. There is not any automatic stretching of arrows. Arrow coordinates are implicit in the name of the arrow for the four basic directions, and they are relative addresses for all other directions. Designing a complex diagram using this macro package is as difficult as fine-tuning a simple diagram, even requiring manual computations of horizontal and vertical dimensions to get a desired arrow size and slope. It runs on both  $\text{\TeX}$  and  $\text{\LaTeX}$ . Documentation is enough to cover the facilities provided by the macros, seven pages in [8] and a two-page source document named `commdiags.tex`, reproducing eleven textbook commutative diagrams.

#### 4.8 Spivak

$\LaTeX$  includes an environment for producing commutative diagrams that supports diagonal arrows of different shapes and in many different, although not arbitrary, slopes. Arrows stretch automatically, and there are ample facilities for defining additional arrow styles. Commutative diagrams are specified as an array of cells, with one object and all departing morphisms in each non-empty cell. Arrow coordinates are relative addresses, and mnemonics can be easily defined for the most common arrow coordinates. Superb facilities for diagram fine-tuning are provided. It only runs on  $\TeX$ . Documentation is excellent, two chapters in [10] describing every detail from diagram design to coding and fine-tuning.

#### 4.9 Svensson

The most recent addition to the commutative diagrams family is the macro package `kuvio.tex`, developed by Anders Svensson. It supports diagonal arrows of different shapes and in many different, although not arbitrary, slopes (implemented by rotating horizontal arrows through PostScript `\special` commands). Arrows stretch automatically, and there are ample facilities for defining additional arrow styles. Commutative diagrams are specified as an array of cells, with either one object or one or more morphisms in each non-empty cell. Arrow coordinates are implicit in the name of the arrow, and they are complemented with explicit slope and length parameters. Different facilities for diagram fine-tuning are provided. It runs on both  $\TeX$  and  $\LaTeX$ . The macros are well documented in a 54-page guide and reference manual [12].

#### 4.10 Taylor

A macro package has been developed by Paul Taylor that supports diagonal arrows of different shapes and slopes, and even at arbitrary slopes (implemented by rotating horizontal arrows through PostScript `\special` commands). Arrows stretch automatically, and there are ample facilities for defining additional arrow styles. Commutative diagrams are specified as an array of cells, with either one object or one or more morphisms in each non-empty cell. Arrow coordinates are implicit in the name of the arrow, and they are complemented with explicit slope and length parameters. There are plenty of options for diagram fine-tuning, either global to the whole document or local to a single diagram. It runs on both  $\TeX$  and  $\LaTeX$ . Documentation is excellent, a quite comprehensive document [13] that is even provided typeset in booklet format.

#### 4.11 Van Zandt

As in the case of the macro packages by Eitan Gurari, `PSTricks` is a general drawing package built on top of  $\TeX$ . Instead of extending  $\TeX$  by defining graphics primitives, however, it is a collection of PostScript-based  $\TeX$  macros, and it can be seen in fact as a high-level  $\TeX$ -like interface to the PostScript language. It supports diagonal arrows of different shapes and arbitrary slopes, curved arrows and loops, automatic stretching, and

symbolic addressing for both node and arrow coordinates. It runs on both  $\text{T}_{\text{E}}\text{X}$  and  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ . The macros are well documented in [15], although there are only two pages describing commutative diagrams and there are only two sample diagrams in the whole document.

## 5 Discussion

### 5.1 Syntactic issues

Syntactic issues are so fundamental to user acceptance of a macro package for typesetting commutative diagrams, that a volunteer task within the  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}3$  project has been founded in October 1992 under the name *Research on Syntax for Commutative Diagrams*, with Paul Taylor as co-ordinator and Michael Barr and Kristoffer Rose as members.

After an initiative by Michael Barr, who started a discussion within the categorical community about the best syntax to adopt for commutative diagrams in  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}3$ , a rather heated debate has taken place in the `CATEGORIES` discussion list. There have been many contributions between June and August 1993, although the discussion list has been silent in these matters ever since.

### 5.2 Curved arrows

The need for curved arrows arises when ‘parallel’ morphisms have to be distinguished from each other, for instance when it is not known if the morphism  $h : A \rightarrow C$  is equal to the composition of the morphisms  $g : B \rightarrow C$  and  $f : A \rightarrow B$ ,

$$\begin{array}{ccccc} A & \xrightarrow{f} & B & \xrightarrow{g} & C \\ & & \underbrace{\hspace{10em}} & & \\ & & h & & \end{array}$$

because otherwise the composite morphism would not need to be made explicit.

The need for curved arrows also arises when there are loops in a diagram. For instance, consider the definition of an isomorphism.

A morphism  $f : A \rightarrow B$  in a given category is an *isomorphism* if there exist a morphism  $g : B \rightarrow A$  in that category such that  $g \circ f = id_A$  and  $f \circ g = id_B$ . That is, if the diagram

$$\begin{array}{ccc} & & id_A \\ & \curvearrowright & \\ & A & \xrightarrow{f} B \\ & \xleftarrow{g} & \curvearrowleft id_B \end{array}$$

commutes. One possible trick to eliminate the need for such curved arrows is to ‘straighten up’ the diagram by appropriately duplicating some nodes. For the previous example, a morphism  $f : A \rightarrow B$  is an isomorphism if the following two diagrams commute:

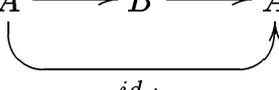
$$A \xrightarrow{f} B \xrightarrow{g} A$$

$$\xrightarrow{id_A}$$

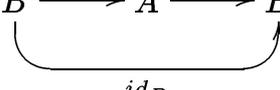
$$B \xrightarrow{g} A \xrightarrow{f} B$$

$$\xrightarrow{id_B}$$

These diagrams, however, look much better with a curved arrow,

$$A \xrightarrow{f} B \xrightarrow{g} A$$


$$\xrightarrow{id_A}$$

$$B \xrightarrow{g} A \xrightarrow{f} B$$


$$\xrightarrow{id_B}$$

and therefore the need for curved arrows cannot always be eliminated without sacrificing diagram clarity and, perhaps arguably, esthetics. While some authors of category theory textbooks seem to prefer to duplicate nodes, others make a thorough use of curved arrows.

### 5.3 Design issues

Diagrams are essentially a communication medium, and therefore good design means a design for readability. Although readability issues can be as subjective as esthetics issues, however, some basic principles may help in the design of readable diagrams. The first principle is to follow the natural order of writing, which at least within occidental writing conventions means left to right, top to bottom, and foreground to background. A second principle is to appropriately give depth to three-dimensional diagrams, in such a way that the foreground lies a little below the background. This principle finds no easy justification, because it may seem to contradict the top-to-bottom order of writing by imposing a bottom-to-top order from foreground to background, but it is true of all kinds of pictorial representations.

### 5.4 User interface

Most of the macro packages provide a simple user interface, consisting of a certain matrix notation. While it adheres to the L<sup>A</sup>T<sub>E</sub>X principle of emphasizing structural descriptions, such a specification may become much too obscure for a complex diagram. Some authors have argued against the use of alternative technologies (if you want WYSIWYG, use a pen and paper) but maybe the time has arrived to have a state-of-the-art drawing program with specific facilities for designing commutative diagrams. One possible scenario would be to sketch the arrangement of nodes and arcs on the computer screen using a mouse, and to let the drawing program translate the design into the language of (any of) the macro packages, taking care of all the time-consuming details of computing coordinates, choosing appropriate slopes for the arrows, placing arrow labels, fine-tuning, etc. Further facilities could include, for instance, trying different layouts based both on the structural description of the diagram as a graph and on knowledge of the kind of graphs that commutative diagrams are, and performing specific operations on descriptions such as, for instance, obtaining the *dual* of a commutative diagram.

## 5.5 Open issues

Although the conceptual framework used for evaluating the different macro packages resulted from the experience of the author using them and converting diagrams between them, it is precisely because of the evaluation having been carried out by only one person that the resulting data may be somewhat biased. A more general investigation would involve mathematicians and computer scientists writing their own diagrams, as well as (L<sup>A</sup>)T<sub>E</sub>X-competent secretaries typing their work, and would produce quantitative measures of learning times for the different macro packages and, once they are fluent in each macro package, measures of the time it takes them to transcribe a diagram drawn on paper.

Further additional investigations include evaluating the degree of help given by each macro package towards improving the quality of the output diagrams, for instance by means of informative messages; quantifying the degree of fine-tuning needed with each macro package in order to produce a complex diagram; evaluating the robustness of the different macro packages when the user makes common errors, such as omitting brackets or mistyping command names; and, last but not least, designing a standard library of common diagrams against which the different macro packages could be evaluated and compared.

## 6 Acknowledgement

In order to avoid name clashes among the control sequences defined in the different macro packages, all the diagrams have been typeset separately and included in the final document as encapsulated PostScript files. Thanks to Michel Goossens and Sebastian Rahtz for their advice. Ricardo Alberich Martí provided guidance during the design of the experiment to obtain time statistics.

## References

- [1] Michael Barr. The diagram macros. Electronic document distributed with the package.
- [2] Francis Borceux. User's guide for diagram 3. Electronic document distributed with the package.
- [3] Eitan M. Gurari. *T<sub>E</sub>X & L<sub>A</sub>T<sub>E</sub>X – Drawing and Literate Programming*. McGraw-Hill, New York, 1994.
- [4] Donald E. Knuth. *The T<sub>E</sub>Xbook*. Addison-Wesley, 15 edition, 1989.
- [5] John Reynolds. User's manual for diagram macros. Electronic document distributed with the package, December 1987.
- [6] Kristoffer H. Rose. X<sub>Y</sub>-pic user's guide. Electronic document distributed with the package, October 1994.

- [7] Kristoffer H. Rose and Ross Moore. Xy-pic reference manual. Electronic document distributed with the package, October 1994.
- [8] Steven Smith. Arrow-theoretic diagrams. Electronic document distributed with the package, May 1994. Chapter 5 in Karl Berry and Steven Smith, *Expanded Plain T<sub>E</sub>X*.
- [9] American Mathematical Society. *AMS-L<sub>A</sub>T<sub>E</sub>X* version 1.2 user's guide. Electronic document distributed with the package, January 1995.
- [10] Michael D. Spivak. *L<sub>A</sub>M<sub>S</sub>-T<sub>E</sub>X – The Synthesis*. The T<sub>E</sub>Xplorators Corporation, Houston, Texas, 1989.
- [11] Michael D. Spivak. *The Joy of T<sub>E</sub>X – A Gourmet Guide to Typesetting with the AMS-T<sub>E</sub>X Macro Package*. American Mathematical Society, 2 edition, 1990.
- [12] Anders G. S. Svensson. Typesetting diagrams with `kuvio.tex`. Electronic document distributed with the package, January 1995.
- [13] Paul Taylor. Commutative diagrams in T<sub>E</sub>X (version 4). Electronic document distributed with the package, July 1994.
- [14] Gabriel Valiente Feruglio. *Knowledge Base Verification using Algebraic Graph Transformations*. PhD thesis, University of the Balearic Islands, December 1994.
- [15] Timothy Van Zandt. PSTricks user's guide. Electronic document distributed with the package, March 1993.

### Appendix I: Sample diagrams

The following diagrams reproduce a fairly complex commutative diagram, taken from [14], using all the macro packages reviewed in this paper. The diagram consists of a pushout construction of partial closed morphisms of total unary algebras in the foreground, together with a corresponding pushout construction of total morphisms of total signature algebras in the background.

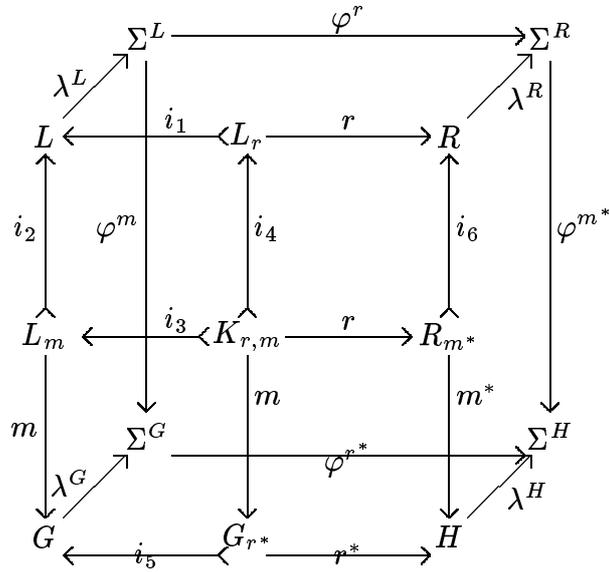
#### 6.1 American Mathematical Society

$$\begin{array}{ccccc}
 L & \xleftarrow{i_1} & L_r & \xrightarrow{r} & R \\
 i_2 \uparrow & & \uparrow i_4 & & \uparrow i_6 \\
 L_m & \xleftarrow{i_3} & K_{r,m} & \xrightarrow{r} & R_{m^*} \\
 m \downarrow & & \downarrow m & & \downarrow m^* \\
 G & \xleftarrow{i_5} & G_{r^*} & \xrightarrow{r^*} & H
 \end{array}$$

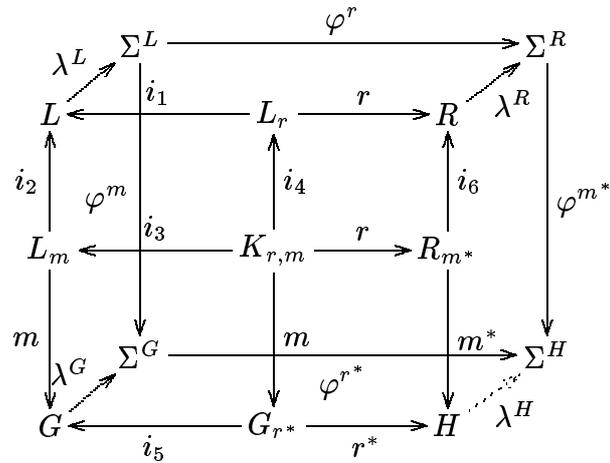
#### 6.2 Michael Barr

$$\begin{array}{ccccc}
 & & \Sigma^L & \xrightarrow{\varphi^r} & \Sigma^R \\
 & \nearrow \lambda^L & \downarrow i_1 & & \nearrow \lambda^R \\
 L & \xleftarrow{i_1} & L_r & \xrightarrow{r} & R \\
 i_2 \uparrow \varphi^m & & \uparrow i_4 & & \uparrow i_6 \\
 L_m & \xleftarrow{i_3} & K_{r,m} & \xrightarrow{r} & R_{m^*} \\
 m \downarrow & & \downarrow m & & \downarrow m^* \\
 & \nearrow \lambda^G & \Sigma^G & \xrightarrow{\varphi^{r^*}} & \Sigma^H \\
 G & \xleftarrow{i_5} & G_{r^*} & \xrightarrow{r^*} & H \\
 & & \downarrow & & \downarrow \varphi^{m^*}
 \end{array}$$

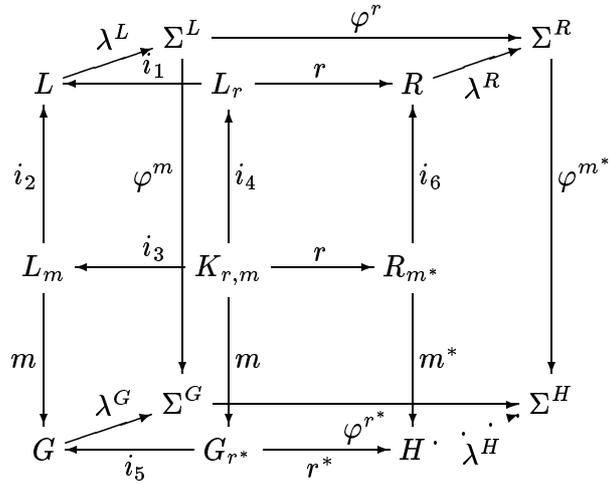
6.3 Francis Borceux



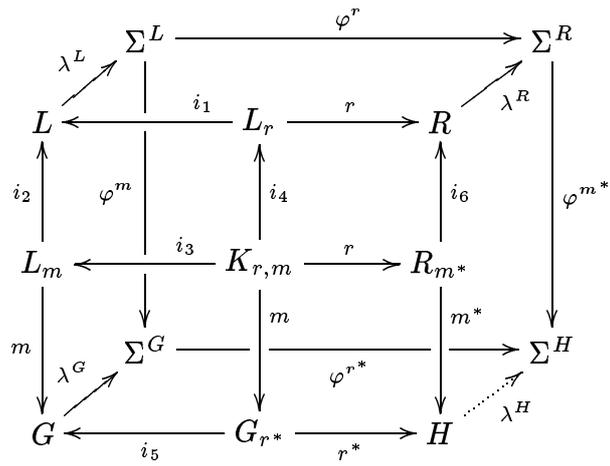
6.4 Eitan Gurari



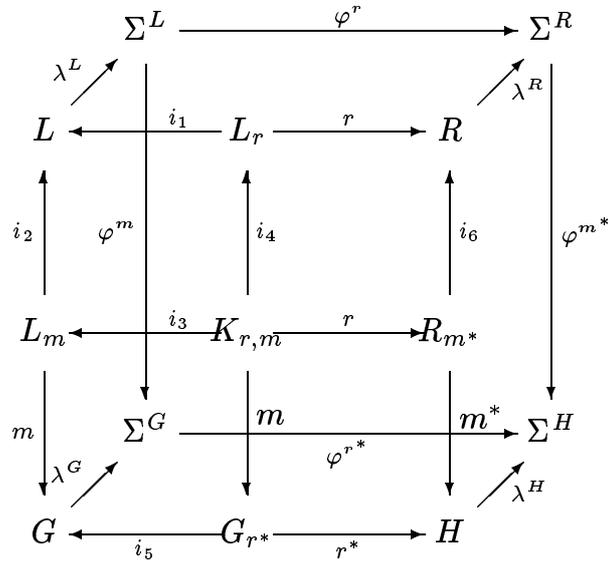
6.5 John Reynolds



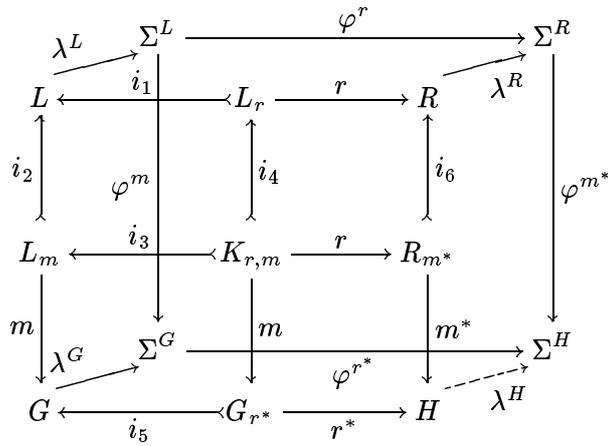
6.6 Kristoffer Rose



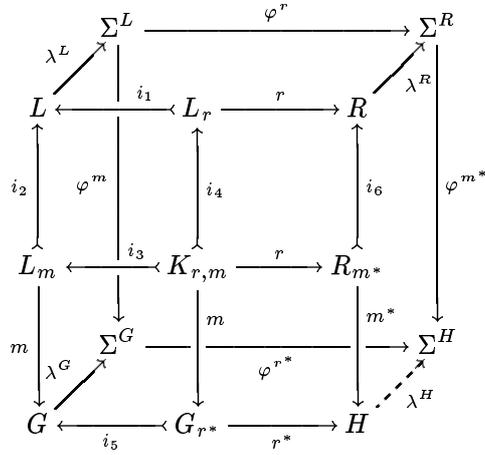
6.7 Steven Smith



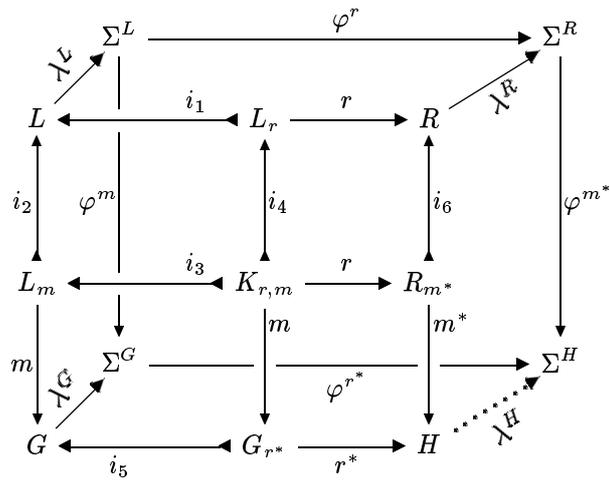
6.8 Michael Spivak



6.9 Anders Svensson



6.10 Paul Taylor



## 6.11 Paul Taylor emulating F. Borceux

$$\begin{array}{ccccc}
 & \Sigma^L & \xrightarrow{\varphi^r} & \Sigma^R & \\
 & \nearrow \lambda^L & & \nearrow \lambda^R & \\
 L & \xleftarrow{i_1} L_r & \xrightarrow{r} & R & \\
 \uparrow i_2 & \downarrow \varphi^m & \uparrow i_4 & \uparrow i_6 & \downarrow \varphi^{m^*} \\
 L_m & \xleftarrow{i_3} K_{r,m} & \xrightarrow{r} & R_{m^*} & \\
 \downarrow m & \downarrow m & \xrightarrow{\varphi^{r^*}} & \downarrow m^* & \\
 G & \xleftarrow{i_5} G_{r^*} & \xrightarrow{r^*} & H & \\
 & \nearrow \lambda^G & & \nearrow \lambda^H & \\
 & \Sigma^G & & \Sigma^H & 
 \end{array}$$

## 6.12 Timothy Van Zandt

$$\begin{array}{ccccc}
 & \Sigma^L & \xrightarrow{\quad} & \Sigma^R & \\
 & \nearrow \lambda^L & & \nearrow \lambda^R & \\
 L & \xleftarrow{i_1} L_r & \xrightarrow{r} & R & \\
 \uparrow \varphi^m & \downarrow \varphi^m & \uparrow i_4 & \uparrow i_6 & \downarrow \varphi^{m^*} \\
 L_m & \xleftarrow{i_3} K_{r,m} & \xrightarrow{r} & R_{m^*} & \\
 \downarrow \varphi^m & \downarrow m & \xrightarrow{\varphi^{r^*}} & \downarrow m^* & \\
 G & \xleftarrow{\quad} G_{r^*} & \xrightarrow{\quad} & H & \\
 & \nearrow \lambda^G & & \nearrow \lambda^H & \\
 & \Sigma^G & & \Sigma^H & 
 \end{array}$$

## Appendix II: Source code for the sample diagrams

```

\newcommand{\up}[1]{\raisebox{1em}{\#1}}
\newcommand{\down}[1]{\raisebox{-1em}{\#1}}
\newcommand{\Left}[1]{\makebox[5pt][r]{\#1}}
\newcommand{\Right}[1]{\makebox[5pt][l]{\#1}}

```

## 6.13 American Mathematical Society

```

\begin{CD}

```

```

L      @<i_1<< L_r      @>r>>   R      \\
@Ai_2AA      @AAi_4A      @AAi_6A \\
L_m      @<i_3<< K_{r,m} @>r>>   R_{m^*} \\
@VmVV      @VVmV      @VVm^*V \\
G      @<<i_5< G_{r^*} @>>r^*> H
\end{CD}$$$

```

### 6.14 Michael Barr

```

$$$$\bfig
\putsquare<-2'-2'-2'-2;500'500>(0,500)[L'L_r'L_m'K_{r,m};%
\qqquad i_1'i_2'i_4'
\putsquare<1'0'-2'1;500'500>(500,500)%
[\phantom{L_r}'R'\phantom{K_{r,m}}'R_{m^*};r' 'i_6'
\putsquare<0'1'1'-2;500'500>(0,0)%
[\phantom{L_m}'\phantom{K_{r,m}}'G'G_{r^*};%
\qqquad i_3'm'\up m'i_5
\putsquare<0'0'1'1;500'500>(500,0)%
[\phantom{K_{r,m}}'\phantom{R_{m^*}}'\phantom{G_{r^*}}'H;%
r'\up{m^*}'r^*]
\putsquare<1'1'1'1;1000'1000>(250,250)%
[\Sigma^L'\Sigma^R'\Sigma^G'\Sigma^H;%
\varphi^r'\varphi^m'\varphi^{m^*}'\varphi^{r^*}]
\putmorphism(125,1125)(1,1)%
[\phantom L'\phantom{\Sigma^L}'{\up{\Right{\lambda^L}}}]%
{0}{1}{1}
\putmorphism(1125,1125)(1,1)%
[\phantom R'\phantom{\Sigma^R}'{\down{\Left{\lambda^R}}}]%
{0}{1}{r}
\putmorphism(125,125)(1,1)
[\phantom G'\phantom{\Sigma^G}'{\up{\Right{\lambda^G}}}]%
{0}{1}{1}
\putmorphism(1125,125)(1,1)%
[\phantom H'\phantom{\Sigma^H}'{\down{\Left{\lambda^H}}}]%
{0}{1}{r}
\efig$$$

```

### 6.15 Francis Borceux

```

\setdefaultscale{40}
\begin{diagram}
? ? \Sigma^L ? ? ? ? \Ear[280] {\varphi^r} ? ? ? ? \Sigma^R ??
? \Near[50] {\lambda^L} ? ? ? ? ? ? ? \near[50] {\lambda^R} ??
L ? ? \Wmono[130] {\qqquad i_1} ? ? L_r ? ? \Ear[130] r ? ? R ?? ??
\Nmono[130] {i_2} ? ? \Sar[280] {\varphi^m} ? ? \nmon0[130] {i_4}%
? ? ? ? \nmon0[130] {i_6} ? ?
\saR[280] {\varphi^{m^*}} ?? ??

```

```

L_m ? ? \Wmono[100] {\qqad i_3} ? ? K_{r,m} ? ? \Ear[100] r ? ?%
R_{m^*} ?? ??
\Sar[130] m ? ? \Sigma^G ? ? \sar[130] {\up{m}} ? ? \eaR[280]%
{\varphi^{r^*}} ? ?
\sar[130] {\up{m^*}} ? ? \Sigma^H ??
? \Near[50] {\lambda^G} ? ? ? ? ? ? ? \near[50] {\lambda^H} ??
G ? ? \wmon0[130] {i_5} ? ? G_{r^*} ? ? \eaR[130] {r^*} ? ? H ??
\end{diagram}

```

## 6.16 Eitan Gurari

```

\Draw
\PenSize(0.25pt)
\ArrowSpec(V,5,3,2)
\ArrowHeads(1)
\GridSpace(10,10)
\GridDiagramSpec()\MyEdge)
\Define\L(4){,+#1..+#2\L, #3\, #4}
\Define\D(4){,+#1..+#2\D, #3\, #4}
\Define\MyEdge(5){
  \IF \EqText(#3,D) \THEN
    \EdgeSpec(D)
  \ELSE
    \EdgeSpec(L)
  \FI
  \IF \EqText(#1,#2) \THEN
    \RotateTo(#4)
    \CycleEdge(#1)
    \EdgeLabel(--$#5$--)
  \ELSE
    \Edge(#1,#2)
    \IF \EqText(,#4) \THEN
      \EdgeLabel(--$#5$--)
    \ELSE
      \EdgeLabel[#4] (--$#5$--)
    \FI
  \FI}
\GridDiagram(8,8)()({
& $\Sigma^L$ \L(6,0,+, \mbox{\varphi^m}) \L(0,6,%,
  \mbox{\varphi^r}) & & & & $\Sigma^R$
  \L(6,0,, \mbox{\varphi^{m^*}}) //
$L$ \L(-1,1, \mbox{\lambda^L}) & & $L_r$ \L(0,-3,+,%
  \mbox{\i_1}) \L(0,3, \mbox{\r}) & &
  $R$ \L(-1,1,+, \mbox{\lambda^R}) & //
& & & & & //
& & & & & //
$L_m$ \L(3,0,+, \mbox{\m}) \L(-3,0, \mbox{\i_2}) & & & %

```

```

    $K_{r,m}$ \L(-3,0,+, \mbox{$i_4$})
    \L(3,0,, \mbox{$m$}) \L(0,-3,+, \mbox{$i_3$}) \L(0,3,%,
    \mbox{$r$}) & & $R_{m^*}$
    \L(3,0,, \mbox{$m^*$}) \L(-3,0,+, \mbox{$i_6$}) & //
& & & & & //
& $\Sigma^G$ \L(0,6,+, \mbox{$\varphi^{r^*}$}) & & & & & %
    $\Sigma^H$ //
$G$ \L(-1,1,, \mbox{$\lambda^G$}) & & $G_{r^*}$ \L(0,-3,%,
    \mbox{$i_5$}) \L(0,3,+, \mbox{$r^*$})
    & & $H$ \D(-1,1,+, \mbox{$\lambda^H$}) & //})
\EndDraw

```

### 6.17 John Reynolds

```

\def\diagramunit{0.6pt}
$$\ctdiagram{
\ctv 0,0:{G}
\ctv 100,0:{G_{r^*}}
\ctv 200,0:{H}
\ctv 0,100:{L_m}
\ctv 100,100:{K_{r,m}}
\ctv 200,100:{R_{m^*}}
\ctv 0,200:{L}
\ctv 100,200:{L_r}
\ctv 200,200:{R}
\ctel 0,100,0,200:{i_2}
\cter 100,100,100,200:{i_4}
\cter 200,100,200,200:{i_6}
\ctel 0,100,0,0:{m}
\cter 100,100,100,0:{m}
\cter 200,100,200,0:{m^*}
\ctetg 100,200,0,200;60:{i_1}
\ctetg 100,100,0,100;60:{i_3}
\cteb 100,0,0,0:{i_5}
\ctet 100,200,200,200:{r}
\ctet 100,100,200,100:{r}
\cteb 100,0,200,0:{r^*}
\ctv 75,25:{\Sigma^G}
\ctv 275,25:{\Sigma^H}
\ctv 75,225:{\Sigma^L}
\ctv 275,225:{\Sigma^R}
\ctet 0,0,75,25:{\lambda^G}
\ctdot
\cteb 200,0,275,25:{\lambda^H}
\ctsolid
\ctet 0,200,75,225:{\lambda^L}
\cteb 200,200,275,225:{\lambda^R}

```

```

\ctelg 75,225,75,25;150:{\varphi^m}
\cterg 275,225,275,25;150:{\varphi^{m^*}}
\ctet 75,225,275,225:{\varphi^r}
\cteb 75,25,275,25:{\varphi^{r^*}}
}$$

```

### 6.18 Kristoffer Rose

```

\definemorphism{unique}\dotted\tip\notip
\spreaddiagramrows{-1pc}
\spreaddiagramcolumns{-1pc}
\diagram
& \Sigma^L \xto'[1,0]' [3,0]_{\varphi^m} [4,0] %
  \xto[rrrr]^{\varphi^r}
  & & & \Sigma^R \xto[dddd]^{\varphi^{m^*}} \ \
L \urto^{\lambda^L} & & \llto_{<<<<{i_1}} L_r \rrto^r & %
  & R \urto_{\lambda^R} \ \ \ \
L_m \uuto^{i_2} \ddto_m & & \llto_{<<<<{i_3}} %
  \uuto_{i_4} K_{r,m} \ddto^{<<<<m} \rrto^r
  & & \uuto_{i_6} R_{m^*} \ddto^{<<<<{m^*}} \ \
& \Sigma^G \xto'[0,1]' [0,3]_{\varphi^{r^*}} [0,4] & & & %
  \Sigma^H \ \
G \urto^{\lambda^G} & & \llto^{i_5} G_{r^*} \rrto_{r^*} & %
  & H \urunique_{\lambda^H}
\enddiagram

```

### 6.19 Steven Smith

```

\harrowlength=45pt
\sarrowlength=.30\harrowlength
$$\gridcommdiag{
& & \Sigma^L & & & & \{\harrowlength=100pt\mapright^{\varphi^r}\}
  & & & & \Sigma^R \ \cr
& \arrow(1,1)\lft{\lambda^L} & & & & & & \arrow(1,1)\rt{\lambda^R} \ \cr
L & & \mapleft^{\quad i_1} & & L_r & & \mapright^r & & R \ \cr \cr
\mapup^{i_2} & & \{\varrowlength=100pt\mapdown^{\varphi^m}\}
  & & \mapup_{i_4} & & & \mapup_{i_6} & &
  \{\varrowlength=100pt\mapdown_{\varphi^{m^*}}\}
  \cr \cr
L_m & & \mapleft^{\quad i_3} & & K_{r,m} & & \mapright^r & & R_{m^*} \ \cr \cr
\mapdown^m & & \Sigma^G & & \mapdown_{\up{m}}
  & & \{\harrowlength=100pt\mapright_{\varphi^{r^*}}\}
  & & \mapdown_{\up{m^*}} & & \Sigma^H \ \cr
& \arrow(1,1)\lft{\lambda^G} & & & & & & \arrow(1,1)\rt{\lambda^H} \ \cr
G & & \mapleft_{i_5} & & G_{r^*} & & \mapright_{r^*} & & H
}$$

```

## 6.20 Michael Spivak

```

$$\Cgaps{0.5}
\Rgaps{0.5}
\cgaps{1.3;0.7;1;1;1.3}
\rgaps{0.7;1;1;1.3;0.7}
\CD
  & \Sigma^L @() \L{\varphi^m} @ (0,-4) @() \L{\varphi^r} @ (4,0)
  & & &
  & \Sigma^R @() \l{\varphi^{m^*}} @ (0,-4)
\\
L @() \L{\lambda^L} @ (1,1)
  & &
  L_r @() \L{i_1} \Ot @ (-2,0) @() \L{r} @ (2,0)
  & & R @() \l{\lambda^R} @ (1,1)
  & &
\\
\\
L_m @() \L{i_2} \Ot @ (0,2) @() \L{m} @ (0,-2)
  & & K_{r,m} @() \L{i_3} \Ot @ (-2,0) @() \L{r} @ (2,0) @() %
  \l{i_4} \Ot @ (0,2) @() \l{m} @ (0,-2)
  & & R_{m^*} @() \l{i_6} \Ot @ (0,2) @() \l{m^*} @ (0,-2)
  & &
\\
& \Sigma^G @() \l{\varphi^{r^*}} @ (4,0)
& & & \Sigma^H
\\
G @() \L{\lambda^G} @ (1,1)
  & & G_{r^*} @() \l{i_5} \Ot @ (-2,0) @() \l{r^*} @ (2,0)
  & & H @() \l{\lambda^H} \a- @ (1,1)
& \\
\endCD$$

```

## 6.21 Anders Svensson

```

\scale=.5
\Diagram
& & \Sigma^L & & & \rTo^{\varphi^r} & & & \Sigma^R \\
& \ruTo^{\lambda^L} & & & & & & \ruTo_{\lambda^R} & & \\
L & & \dTo_{\varphi^m} \lMono^{i_1} : {.25} \br & & L_r & & %
  \rTo^r & & R & & \\
& & & & & & & & & \\
\lMono^{i_2} & & & & \lMono_{i_4} & & & \lMono_{i_6} & & %
  \dTo^{\varphi^{m^*}} \\
& & & & & & & & & \\
L_m & & & \lMono^{i_3} : {.25} \br & & K_{r,m} & & \rTo^r & & %
  R_{m^*} & & \\

```



```

& \Rnode{SL}{\Sigma^L} & & & \Rnode{SR}{\Sigma^R} \\ [0.15in]
\Rnode{L}{L} & & \Rnode{Lr}{L_r} & & \Rnode{R}{R} & \\ [0.15in]%
\\ [0.15in]
\Rnode{Lm}{L_m} & & \Rnode{Krm}{K_{r,m}} & & \Rnode{Rm}{R_{m^*}}%
& \\ [0.15in]
& \Rnode{SG}{\Sigma^G} & & & \Rnode{SH}{\Sigma^H} \\ [0.15in]
\Rnode{G}{G} & & \Rnode{Gr}{G_{r^*}} & & \Rnode{H}{H} & \\ [0.15in]
\end{array}
\psset{nodesep=5pt,arrows=->}
\everypsbox{\scriptstyle}
\ncLine{Lr}{R} \Aput{r}
\ncLine{Krm}{Rm} \Aput{r}
\ncLine{Gr}{H} \Bput{r^*}
\ncLine{Lr}{L} \bput{0}(0.3){i_1}
\ncLine{Krm}{Lm} \bput{0}(0.3){i_3}
\ncLine{Gr}{G} \Aput{i_5}
\ncLine{SL}{SR} \Aput{\varphi^r}
\ncLine{SG}{SH} \Bput{\varphi^r^*}
\ncLine{SR}{SH} \Aput{\varphi^m^*}
\ncLine{SL}{SG} \Bput{\varphi^m}
\ncLine{Lm}{G} \Bput{m}
\ncLine{Krm}{Gr} \aput{0}(0.3){m}
\ncLine{Rm}{H} \aput{0}(0.3){m^*}
\ncLine{Lm}{L} \Aput{i_2}
\ncLine{Krm}{Lr} \Bput{i_4}
\ncLine{Rm}{R} \Bput{i_6}
\ncLine{L}{SL} \Aput[1pt]{\lambda^L}
\ncLine{R}{SR} \Bput[1pt]{\lambda^R}
\ncLine{G}{SG} \Aput[1pt]{\lambda^G}
\ncLine[linestyle=dashed]{H}{SH} \Bput[1pt]{\lambda^H}$$

```

### Appendix III: Automatic stretching

The following diagrams illustrate the degree of automatic stretching of arrows provided by each of the macro packages. A simple square diagram is typeset with a long label for the top-leftmost node in order to determine if the bottom horizontal arrow stretches to meet its source node, and it is also typeset with a long label for the top horizontal arrow in order to determine if it stretches long enough to fit the label.

#### 6.25 American Mathematical Society

Arrows do not stretch to meet their source and target nodes, but they stretch to fit their labels, although only the arrow carrying the long label stretches. Manual fine-tuning is needed in order to get the same stretch in all the other arrows lying in the same column of the array.

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B \\
 g \downarrow & & \downarrow g' \\
 C & \xrightarrow{f'} & D
 \end{array}
 \quad
 \begin{array}{ccc}
 A \times A \times A \times A & \xrightarrow{f} & B \\
 g \downarrow & & \downarrow g' \\
 C & \xrightarrow{f'} & D
 \end{array}
 \quad
 \begin{array}{ccc}
 A & \xrightarrow{f \star f \star f \star f \star f} & B \\
 g \downarrow & & \downarrow g' \\
 C & \xrightarrow{f'} & D
 \end{array}$$

### 6.26 Michael Barr

Arrows within the shape macros stretch to meet their source and target arrows, but individual arrows obtained with `\putmorphism` do not. In both cases, arrows do not stretch to fit their labels and the required dimensions have to be given explicitly.

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B \\
 g \downarrow & & \downarrow g' \\
 C & \xrightarrow{f'} & D
 \end{array}
 \quad
 \begin{array}{ccc}
 A \times A \times A \times A & \xrightarrow{f} & B \\
 g \downarrow & & \downarrow g' \\
 C & \xrightarrow{f'} & D
 \end{array}
 \quad
 \begin{array}{ccc}
 f \star A & \xrightarrow{f \star f \star f \star f \star f} & B \star f \\
 g \downarrow & & \downarrow g' \\
 C & \xrightarrow{f'} & D
 \end{array}$$

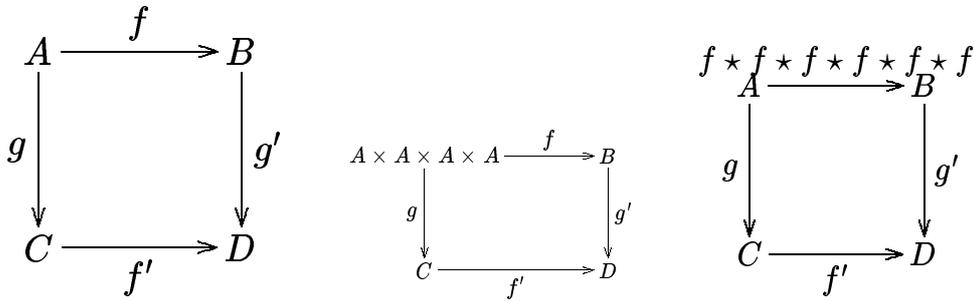
### 6.27 Francis Borceux

Arrows stretch to meet their source and target nodes, but they do not stretch to fit their labels.

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B \\
 g \downarrow & & \downarrow g' \\
 C & \xrightarrow{f'} & D
 \end{array}
 \quad
 \begin{array}{ccc}
 A \times A \times A \times A & \xrightarrow{f} & B \\
 g \downarrow & & \downarrow g' \\
 C & \xrightarrow{f'} & D
 \end{array}
 \quad
 \begin{array}{ccc}
 f \star A & \xrightarrow{f \star f \star f \star f \star f} & B \star f \\
 g \downarrow & & \downarrow g' \\
 C & \xrightarrow{f'} & D
 \end{array}$$

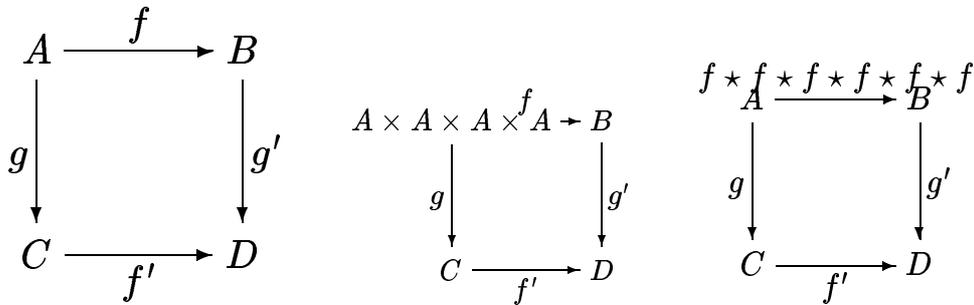
### 6.28 Eitan Gurari

Arrows stretch to meet their source and target nodes, but they do not stretch to fit their labels.



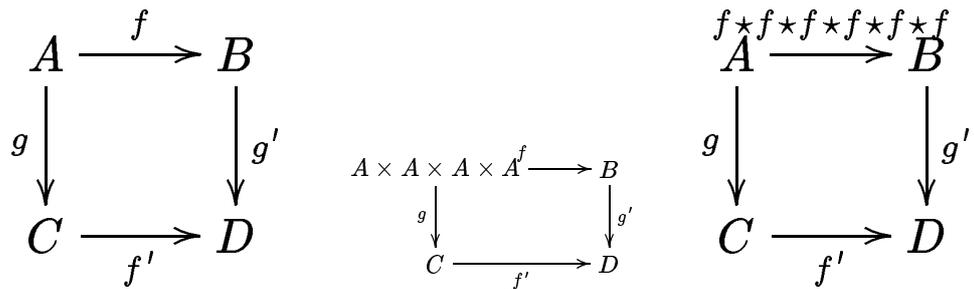
**6.29 John Reynolds**

Arrows stretch to meet their source and target nodes, although the labels do not get centered on the stretched arrows. They do not stretch to fit their labels.



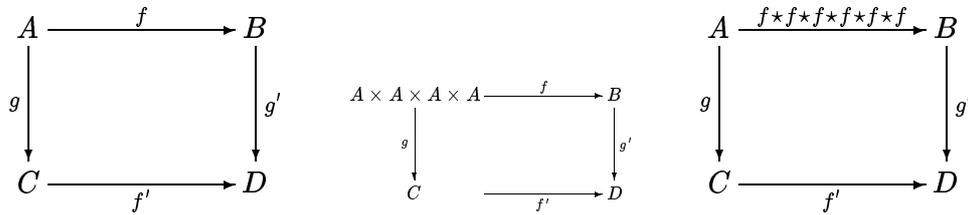
**6.30 Kristoffer Rose**

Arrows stretch to meet their source and target nodes, although the labels do not get centered on the stretched arrows. They do not stretch to fit their labels.



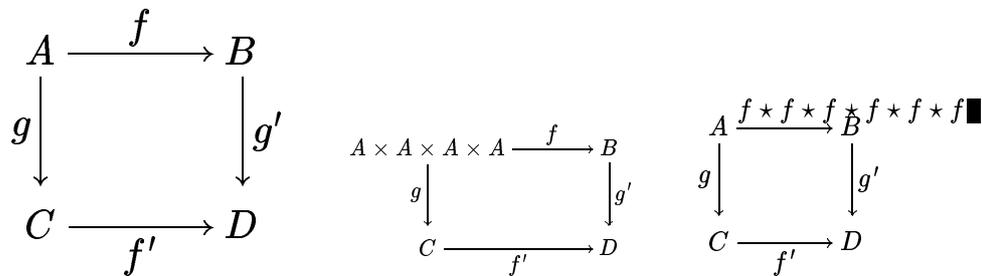
**6.31 Steven Smith**

Arrows do not stretch to meet their source and target nodes, but they stretch to fit their labels.



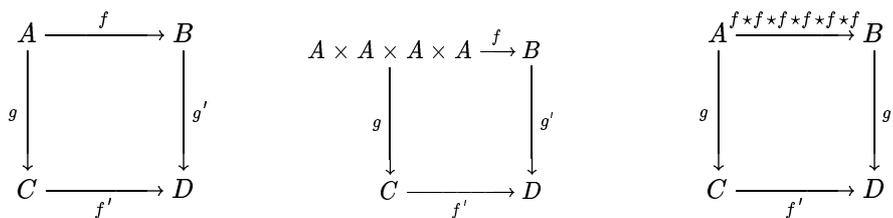
### 6.32 Michael Spivak

Arrows stretch to meet their source and target nodes, but they do not stretch to fit their labels, even producing overfull `\hboxes`.



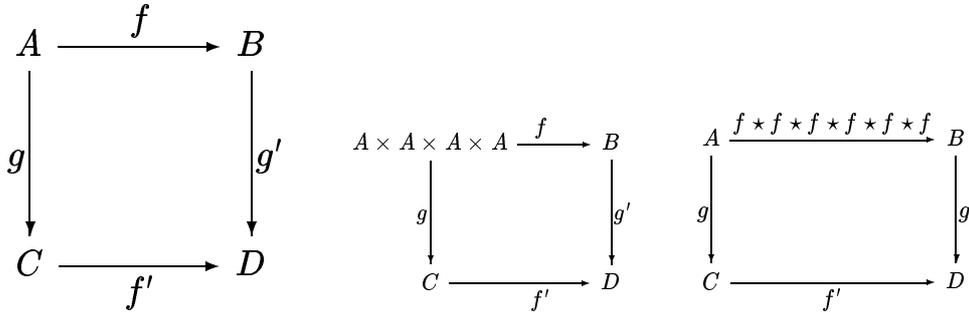
### 6.33 Anders Svensson

Arrows stretch to meet their source and target nodes, but they do not stretch to fit their labels.



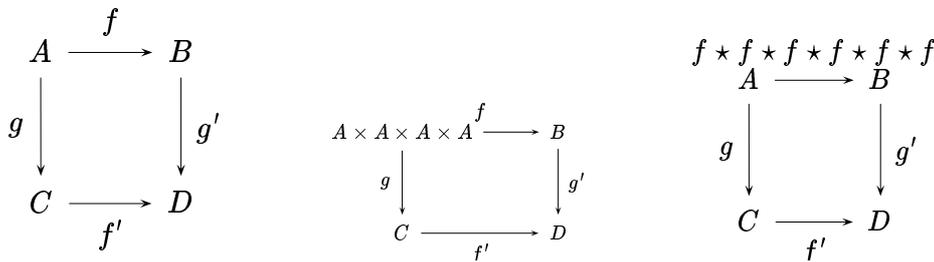
### 6.34 Paul Taylor

Arrows stretch to meet their source and target nodes, and they also stretch to fit their labels.



### 6.35 Timothy Van Zandt

Arrows stretch to meet their source and target nodes, although the labels do not get centered on the stretched arrows. They do not stretch to fit their labels, and the required dimensions have to be given explicitly.



## Appendix IV: Resource requirements

### 6.36 Package size

The following table lists the size (in kilobytes) of the main macro files that have to be loaded into  $\text{T}_{\text{E}}\text{X}$  or  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  in order to use the respective packages.

package	main files	size
$\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$	amscd.sty	10
Barr	diagram.tex	40
Borceux	Diagram	270
Gurari	dratex.sty and aldratex.sty	136
Reynolds	diagmac.sty	42
Rose	xypic.tex and xy.tex	68
Smith	arrow.tex	24
Spivak <sup>1</sup>	amstex1.tex and lamstex.tex	200
Svensson	kuvio.tex and arrsy.tex	86
Taylor	diagrams.tex	86
Van Zandt	pstricks.tex, pst-node.tex and pstricks.con	84

### 6.37 Time statistics

The following table lists statistics for the time (in seconds) needed to typeset the sample diagrams presented in Appendix I, using  $\mathcal{T}\mathcal{E}\mathcal{X}$  and  $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}2_{\varepsilon}$  on a Mackintosh SE/30, with the different macro packages. The mean time and the confidence interval at a significance level of 95% is given for the total time needed to typeset a diagram and for the marginal time, computed as the difference between the time needed to typeset two copies of the sample diagram using a macro package and the time needed to typeset one copy of the same diagram using the same macro package, where these two random variables are assumed to have a normal distribution and to be independent, and where the mean and the confidence interval have been estimated from a sample of 30 observations.

package	total time			marginal time		
	mean	95% confidence interval		mean	95% confidence interval	
$\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$	18.1367	18.0317	18.2416	1.6600	1.5544	1.7660
Barr	48.8033	48.7731	48.8335	29.9334	29.8800	29.9870
Borceux	127.5630	127.5060	127.6210	28.3170	28.1730	28.4600
Gurari	388.4630	388.4320	388.4950	638.8270	638.5000	639.1490
Reynolds	46.7000	46.6357	46.7643	26.8200	26.7520	26.8880
Rose	242.7400	242.3810	243.0990	210.0730	209.2900	210.8500
Smith	22.9600	22.9031	23.0169	5.2400	5.1817	5.2980
Spivak	37.3000	37.2445	37.3555	11.9833	11.9263	12.0400
Svensson	81.3867	81.2902	81.4831	44.5733	44.4668	44.6799
Taylor	66.8400	66.7553	66.9247	14.3133	14.1420	14.4850
Taylor emul. Borceux	67.3533	67.3243	67.3823	11.4767	11.4360	11.5170
Van Zandt	37.8233	37.7809	37.8657	14.2100	14.1520	14.2680

1. Although  $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$  offers much more than the macros for commutative diagrams, it has to be loaded as a whole in order to use the macros. Most such macros can be removed from  $\mathcal{T}\mathcal{E}\mathcal{X}$ 's memory by loading the file `cd.tex` (4 kilobytes), freeing up about 5800 words of memory, and can be later added again by loading the file `cd.tex` (36 kilobytes), but the whole  $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$  has to be loaded before.

## Appendix V: Availability

### 6.38 Availability

The following table lists the CTAN directories where the different macro packages are stored, together with the authoritative FTP addresses they are mirrored from.

package	CTAN directory	mirrored from
$\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$	macros/latex/packages/amslatex/	e-math.ams.org /ams/
Barr	macros/generic/diagrams/barr/	not mirrored
Borceux	macros/generic/diagrams/borceux/	theory.doc.ic.ac.uk /tex/contrib/borceux/diagram-3/
Gurari	macros/generic/dratex/	ftp.cis.ohio-state.edu /pub/tex/osu/gurari/
Reynolds	macros/latex209/contrib/misc/ diagmac.sty	not mirrored
Rose	macros/generic/diagrams/xypic/	ftp.diku.dk /diku/users/kris/tex/
Smith	macros/eplain/ arrow.tex	ftp.cs.umb.edu /pub/tex/eplain/
Spivak	macros/lamstex/	not mirrored
Svensson	macros/generic/diagrams/kuvio/	math.ubc.ca /pub/svensson/
Taylor	macros/generic/diagrams/taylor/	theory.doc.ic.ac.uk /tex/contrib/taylor/tex/
Van Zandt	graphics/pstricks/	princeton.edu /pub/tvz/pstricks/

# Conversion of the Euler Metafons into the PostScript Type1 font language

Erik-Jan Vens

erikjan@lunatix.icce.rug.nl

## 1 History of the Euler Metafons

Hermann Zapf designed the Euler family for the American Mathematical Society for mathematical use, not as a text face, in 1980–1981. This calligraphic family consists of Fraktur, Script, Upright Italic and Math extension. Several people have contributed to the Metafont programming, among them Don Knuth, John Hobby, David Siegel, Dan Mills and Carol Twombly. There is a description of Euler in TUGboat 10.1.

For this article I will concentrate on Euler Roman Medium. The chosen example is the capital A from this font. Figure 1 shows what the character looks like.

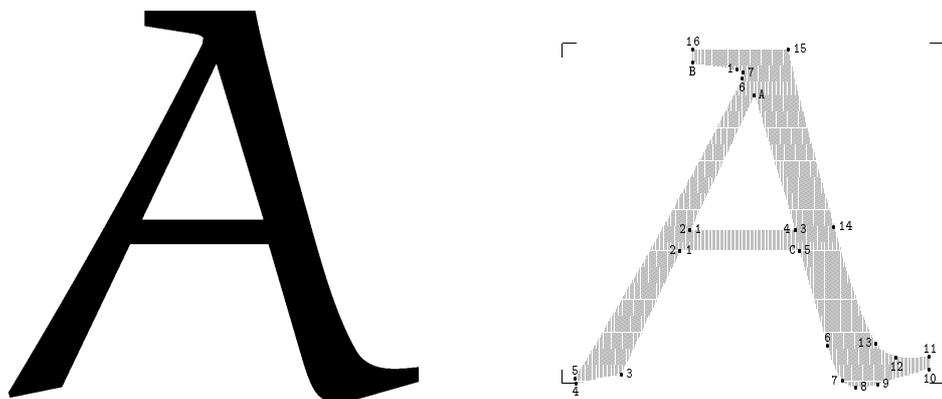


Figure 1: Left: The character capital A in PK format; Right: The same character in MF format

## 2 Different approaches to computerized curve description

There are several approaches possible to describe a curve. From the days of the Renaissance onwards people have tried to do so without good results. The first one to really succeed in doing so was Pierre Etienne Bézier. He applied the theories of Sergeĭ Bernshteĭn to his specific area, computer-aided design.

### 2.1 The richness of the Metafont language

METAFONT uses a method whereby curves can be drawn either by filling specified outlines or by simulating the movement of a pen. We can pick up a pen with a specified form of the head and start drawing, and we can use an eraser and remove ink from the paper. We can clip a form or rotate it. We can scale it or clip it out and paste it one or more times.

I will not go into further detail of METAFONT. It is sufficient to say that METAFONT is a very rich and powerful language.

### 2.2 The limitations of the Type1 language

The Postscript Type1 language is an enhanced subset of the Postscript language. It can be used to specify a path and fill it. This is also a part of the METAFONT language, so one can easily convert from Type1 to METAFONT, but the other way around provides us with a lot of difficulties. Again, I will not go into further detail of a language. Those persons who are interested in what comprises Type1 are encouraged to buy the so-called 'Black Book', the Adobe Type1 Font Specification.

## 3 Transformation of Metafont into Type1

One can discuss the necessity for the T<sub>E</sub>X community to use Postscript fonts. I feel that there are far too little good-quality fonts around in the METAFONT format, so one almost must use Postscript fonts. This doesn't mean the technical capabilities of METAFONT are under discussion.

The Postscript world is far larger than the T<sub>E</sub>X world. It would be nice to be able to donate what 'we' have to this Postscript world. This has to be done in Postscript form, hence the necessity for transformation.

There are several routes one can take to convert from METAFONT into Type1. I have outlined three of them. None of them is perfect in the sense that it can be fully automated. All of them need handtuning. This makes it a very tedious job, almost like starting from scratch. The first steps for all of them can be measured in minutes. The fine-tuning will take days, if not weeks or months per font.

I have used commercial tools in almost all of the stages. I don't know of any public domain or shareware programs that can do all of the tricks needed to bring a font up.

### 3.1 Using the GNU fontutils

Karl Berry and Kathy Hargreaves have written a suite of utilities to manipulate bitmapped fonts and trace them into outline fonts. All of their data is created by scanning fontbooks. But one can also use METAFONT to render a bitmap font in the form of a GF file and use this as starting point for tracing.

I created a very huge GF file by using

```
mf "\mode:=proof; mag:=3.171875; input eurm10"
```

I have not tried out whether smaller input files also create good results. This magnification was simply the largest before METAFONT issued a value too large error-message. Since I could create such large file, I didn't bother experimenting with the size. Then I used their program `limn` to create a `bzr` file:

```
limn -verbose -corner-surround=24 -filter-surround=24 \  
      -filter-alternative-surround=12 -subdivide-surround=24 \  
      -tangent-surround=24 -reparameterize-threshold=30 \  
      eurm10.8252gf
```

A `bzr` file is binary representation of curve information. It can be converted to several formats, amongst them METAFONT. But I converted it to Type1 with:

```
bzrto -pstype1 eurm10
```

This does not create a 'real' Type1 font, but a Ghostscript version of the same, which will only run under a Ghostscript version of the Postscript interpreter. I therefore had to convert this to a normal Type1 font with two little programs which I hacked together from two utilities I already had:

```
gsf2ps < eurm.gsf > eurm.chr  
chr2ps eurm.chr eurm.aapje
```

Now I could import the `Charstrings` section from the `eurm.aapje` file into a template file which I had created from a valid Type1 font.

I imported the thus created font into Fontmonger and edited all the characters. The amount of editing depends on the parameters as chosen for `limn`. It is very important to chose these as good as possible. The parameters I have shown in this example might not be the best around, but these can only be found by fiddling with them.

To show how much editing has to be done, look at Figure 2, which shows the character capital A before and after editing.

### 3.2 Using a commercial program

*Fontographer or Fontmonger*

In this case I used METAFONT to render the bitmaps which I converted to a picture format with Ghostscript. I made a small T<sub>E</sub>X file which consisted of a loop which put

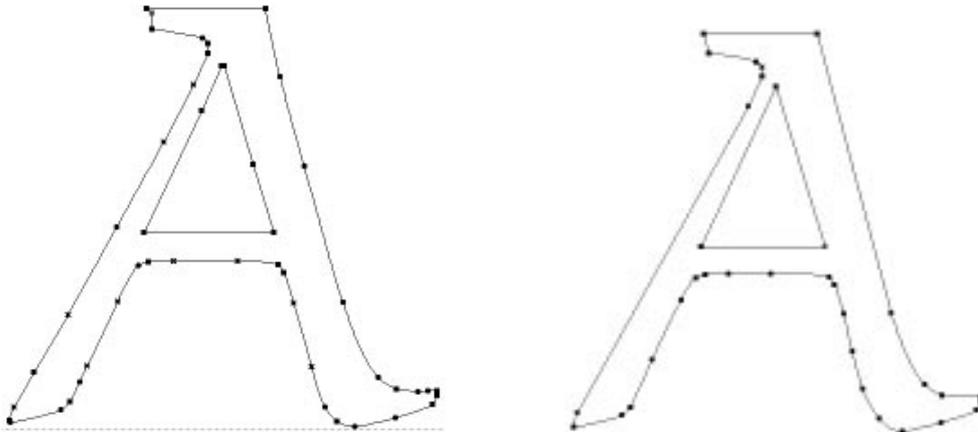


Figure 2: Left: The character before editing; Right: The character after some editing

a large character on each page, and then used `dvips` to create a Postscript file, and `Ghostscript` to render a picture for each page of the input file.

Then I started up `Fontographer` and for each character I imported a background which I then traced using the `Fontographer` trace function. This is a very fast process which gives fairly good results.

The real problem lies with the hinting. Contrary to the approach of `METAFONT` where each instance of a font at a specific pointsize can be rendered at it's best rasterizing for every conceivable outputdevice, `Type1` uses a system called hints to specify how the rasterizer should decide which pixels to turn on or off.

One can let `Fontographer` try to decide which hints are general hints and then see for oneself whether it has chosen right hints.

The result of the capital A is shown in Figure 3.

#### *CorelTrace*

Figure 4 shows what the output with `CorelTrace` looks like.

This is definitely not good. There are a lot of parameters one can adjust, but I haven't found any better settings. So, using `CorelTrace` is not a real option.

### 3.3 Using MetaPost

When one has a `METAFONT` source, it might be a good idea to directly convert it to Postscript using `MetaPost`. This program is `METAFONT` with some changes, and it can read almost all of the standard `METAFONT` files. There are a few exceptions, notably the cutoff function is not implemented. But the overall results are good and very fast.

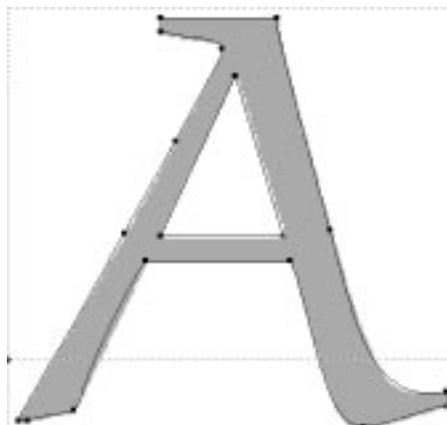


Figure 3: Capital A as traced by Fontographer

Just as with the other options, the main problem lies with the hinting. One has still to do the hinting after the conversion is done. I have used a commercial program to import the Postscript files for each character. The general result is shown in Figure 5.

#### 4 Conclusions and general remarks

To quote from Don Knuth's METAFONT book: 'We should realize before we begin that it would be a mistake to set our hopes too high. Mechanically generated letters that are untouched by human hands and unseen by human eyes can never be expected to compete with alphabets that are carefully crafted to look best on a particular device. There's no substitute for actually looking at the letters and changing their pixels until the result looks right. Therefore our goal should not be to make hand-tuning obsolete; it should rather be to make hand-tuning tolerable. Let us try to create meta-designs so that we would never want to change more than a few pixels per character, say half a dozen, regardless of the resolution. At low resolutions, six pixels will of course be a significant percentage of the whole, and at higher resolutions six well-considered pixel changes can still lead to worthwhile improvements. The point is that if our design comes close enough, a person with a good bitmap-editing program will be able to optimize an entire font in less than an hour. This is an obtainable goal, if rounding is done judiciously.'

The other realization one has to make is whether it is worthwhile to spend so much time into converting a font which is only relevant for mathematicians, who are usually already using  $\text{\TeX}$ , and hence do not need a Postscript font. On the other hand, the work of a genius such as Mr. Zapf should never be underestimated, and therefore the more widespread it's use, the better.

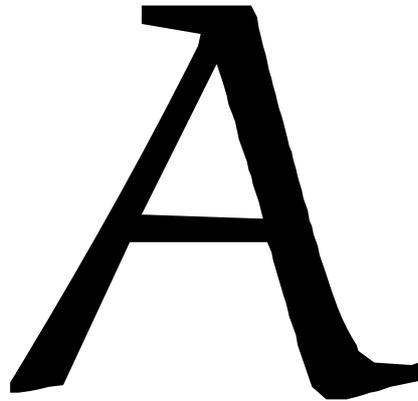


Figure 4: Capital A as traced by CorelTrace

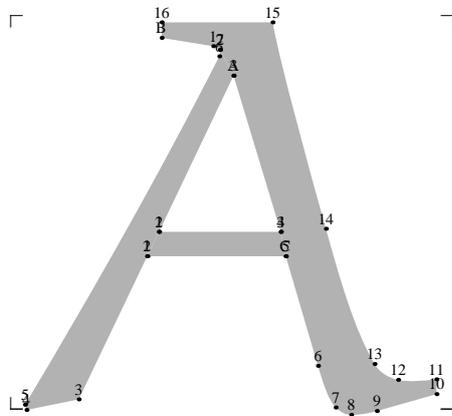


Figure 5: Capital A as created by MetaPost

# When METAFONT does it alone

Jiří Zlatuška

Faculty of Informatics  
Masaryk University  
Burešova 20  
602 00 Brno  
Bohemia  
zlatuska@informatics.muni.cz

## Abstract

Combining METAFONT and  $\TeX$  when typesetting text and graphics together has been shown on several occasions to bring very impressive results. A. Hoenig presented a method for communication between  $\TeX$  and METAFONT in order to solve two problems otherwise difficult to handle within  $\TeX$  or METAFONT alone: label placement for diagrams generated by METAFONT, and curvilinear typesetting. We show that the method for curvilinear typesetting (involving three passes in Hoenig's approach) can be considerably simplified by using the extended ligature mechanism of  $\TeX$  3, and that a single METAFONT pass is actually sufficient, with quite a simple interface on  $\TeX$ 's side. Institutional seal text placement can be realized as a simple METAFONT application using this method. While PostScript offers ready-to-use easy solutions to this class of problems, METAFONT solutions can still be preferable to PostScript because of the ability of adding META-ness, e.g., by introducing second-order magnitude corrections/distortions to the letters and/or logos in order to enhance legibility when used in smaller sizes.

## 1 Introduction

There are several methods available for including graphical information into  $\TeX$  documents. Some of them rely on the `\special` primitive of  $\TeX$  and consists in combining pictures created by tools independent of  $\TeX$  on the level of dvi drivers. Within the  $\TeX$  world, the METAFONT program can be used for defining graphic objects by using its capabilities as in the case of defining letterforms, resulting in a 'font' containing graphic images as 'letters' which can be typeset within a  $\TeX$ -composed document.

There are interesting possibilities arising from combination of METAFONT and  $\TeX$  especially when it comes to typesetting text material along curved baselines and/or

combined with other pieces of graphical information. Effects of this kind can also be prepared using PostScript transformations as prepared by, e.g., the `pstricks` collection by Timothy van Zandt. Nonetheless, reasons can be found for preferring a solution using just the combination of METAFONT and T<sub>E</sub>X, excluding effects caused by combination of the dvi driver and the underlying printing language. One of them can be the necessity of using either a printer or a previewer which does not understand PostScript. Another may be the need to use non-linear effects within the generated pictures, e.g., scaling the proportions of letters used within them similarly as they change when changing design sizes for METAFONT-generated fonts.

One of the problems of using METAFONT easily for creating pictures involving also text parts, is the lack of ‘typesetting’ capabilities (solved in John Hobby’s `metapost` generating PostScript output from an input formulated in a language extending METAFONT) which would allow efficient incorporation of typeset text into METAFONT-generated figures. Alan Hoenig [1, 2] defined a scheme for bidirectional communication between T<sub>E</sub>X and METAFONT allowing T<sub>E</sub>X to submit requirements for special effects under which METAFONT would generate particular instances of the letters (e.g., rotated and/or scaled) and T<sub>E</sub>X would place these letters onto the appropriate place within the typeset material. One particular application of this T<sub>E</sub>X and METAFONT ‘working together’ was curvilinear typesetting when typesetting centred text around the circumference of a circular area as used for institutional seals or logos. In this paper we show an approach for tackling this problem within a simpler scheme than the three-step method described by Hoenig. We use the capabilities of the ligature programs of METAFONT to create composite pictures which can be then invoked from within T<sub>E</sub>X documents with a certain level of ‘intelligence’ built into them. This can be simpler to use than the three-step method and the composition steps embedded into the font definition corresponding to the particular piece of graphics.

## 2 Typesetting along curved baselines with Metafont

When typesetting parts of a text using METAFONT in non-standard ways such as placing the text along a curve and/or combined with other graphic objects, it is often necessary to break the picture into separate parts stored as individual characters within a font which METAFONT generates as its output. There are several reasons for doing this. The resulting METAFONT picture comprising the picture as a whole may be too large for METAFONT’s memory limitations. We may also want to be able to use parts of the picture independently of the others, or to select just a few of them in particular cases. On the level of typesetting the pictures in T<sub>E</sub>X, it is necessary to be able to typeset the fragments of the picture (characters from the font representing it) at the proper places in the typeset material.

We can illustrate some of the requirements which should be handled by a METAFONT-based definition of an institutional logo for the author’s home institution – a logo

of the Faculty of Informatics of Masaryk University consisting of an Escher-like graphic based on a design by Petr Sojka, encircled by a pair of Latin inscriptions typeset around the circumference with different orientation each. The logo as such looks as follows:



The basic variations we may have in mind may be typesetting just the graphics drawing inside the seal, typesetting just the inscription alone, skipping out the shaded parts – hence obtaining variants of the picture looking as follows:



## 2.1 Hoenig's method

A. Hoenig proposed a method for combining METAFONT and  $\text{T}_{\text{E}}\text{X}$  in such a way that a sequence of three steps of communication takes place between METAFONT and  $\text{T}_{\text{E}}\text{X}$ . First,  $\text{T}_{\text{E}}\text{X}$  makes basic measurements of the text parts to be typeset. Second, METAFONT reads this information, generates the pictures and/or transformed letterforms and passes this back to  $\text{T}_{\text{E}}\text{X}$  as a font together with numeric information (e.g., positions onto which the characters should be typeset) encoded as kerns between pairs of special structure. Third,  $\text{T}_{\text{E}}\text{X}$  reads the metric information associated with the font, extracts any encoded data which are needed and then typesets the generated characters onto specified positions.

Although the communication between METAFONT and  $\text{T}_{\text{E}}\text{X}$  is solvable in this way, the resulting process is rather complicated. It is hard to imagine the technique becoming so easy to use that the resulting graphics could regularly be invoked in non-expert users' documents.

## 2.2 Leaving the placement to Metafont

The final composition of the picture is left to  $\text{T}_{\text{E}}\text{X}$  in Hoenig's 'METAFONT cooperates with  $\text{T}_{\text{E}}\text{X}$ ' method, and this is also the reason that communication between METAFONT and  $\text{T}_{\text{E}}\text{X}$  is introduced.

There is a simpler possibility of leaving the whole job of placing the parts of the final picture to METAFONT alone. METAFONT can generate characters which are placed correctly with respect to the resulting picture and use a common point of the resulting graphic composition as the reference point of each of the characters generated as parts of it. METAFONT knows this information in any case, so it can just use it for changing

the `currenttransform` transformation in order to move the character to the desired place. (Note that METAFONT will not exceed its memory limits if it just moves the picture within the coordinate system without actually setting on pixels far away of it.)

The resulting font METAFONT generates consists of characters which should be superposed one on top of another. The point where this should occur from T<sub>E</sub>X's point of view is the common reference point of the generated characters. A T<sub>E</sub>X loop independent of the structure of the picture can be used for this – just reserving space for the picture within the typeset document and overprinting all the characters from the font within the loop. In order for this to work, the widths of the individual characters in such a font are set to zero so that sequencing the characters on T<sub>E</sub>X's input actually means printing them on top of each other.

First four characters needed to typeset the upper part of the curved inscription above are (the dot indicating the reference point):

F                    A                    C                    U

Overprinting them on top of each other yields:

FACU

### 2.3 Character definitions

In order to generate these characters, we have to modify the METAFONT program files so that the letterforms are properly transformed and to add the code for computing their parameters.

The basic change in the METAFONT programs for characters can be done following the way A. Hoenig used, with just a few extra parameters added because the placement of the calculations should be based on them.

The code defining letters of the form

```
cmchar "The letter F";
beginchar
  (n,11.5u#-width_adj#,cap_height#,0);
  ...
endchar;
```

will be replaced by METAFONT macros of the form:

```
width.F:=11.5u-width_adj;
def F_(expr n, rotation_angle,
      position_shift) =
```

```

currenttransform:=identity
    rotated rotation_angle
    shifted position_shift;
def t_=transformed currenttransform enddef;
cmchar "The letter F";
beginchar
    (n,11.5u#-width_adj#,cap_height#,0);
    ...
endchar;

```

In this transformation we extracted the width information concerning the character (which will be needed for proper character placement) and defined a macro generating an instance of the letter as slot number  $n$  in the generated font consisting of the letter rotated by angle `rotation_angle` and moved to position given by vector `position_shift`.

Note that `currenttransform` in this definition may be further modified by other transformations needed. When typesetting texts in circular logos, it is for example good to stretch the letters a bit when the size of the logo becomes smaller. This can be achieved by introducing a global parameter `taller_letters` (e.g., to depend on the second order of logo size change), and modifying the `currenttransform` setting to

```

currenttransform:=identity
    yscaled taller_letters
    rotated rotation_angle
    shifted position_shift;

```

## 2.4 Computing character positions

For character position calculations it is enough to incrementally move the reference point of the text characters along the circle and to compute the positions and angular shifts of the letters to be typeset. These calculations can be carried out analytically, and use of the `solve` macro is not needed (in contrast with Hoenig's method).

For the upper arch of the circular text, the character position calculations are based on the widths of the characters only, and for the lower arch also on the height of the caps height (because the characters should be shifted out of the basic circle by this distance).

The essential piece of information are the widths of the characters (including any kerning which follows them – as Hoenig notes in [2], it is better not to rely on the default kerning used for linear text). We define an array for this,

```
numeric c[];
```

and fill in the width information including kerning for the circular text such as

```

c[1]:=width.F+kkk;
c[2]:=width.A+kk;

```

```

c[3]:=width.C;
c[4]:=width.U;
c[5]:=width.L+kk;
c[6]:=width.T+kk;
c[7]:=width.A;
c[8]:=width.S;
...
c[chars_placed_up]:=width.AE;

c[first_down]:=width.U;
...
c[last_down]:=width.A;

```

Now three arrays will be defined,

```

numeric centering[],
        rot_angle[];
pair    pos_shift[];

```

for recording the information concerning rotation angle and position shift of each of the individual instances of the letter, and an auxiliary array used for centering the texts along the vertical axis.

Now based on the character widths in the `c` array we are ready to calculate the co-ordinates of each of the characters `c[1]` up to `c[chars_placed_up]` placed on the upper arch. Note that two passes are done here. The first one starts typesetting at 180 degrees, calculates the overall angle length, and sets `centering[0]` to the actual angle where centered text should start from. The second pass then recalculates the positions and angles starting from this corrected initial setting.

```

centering[0]:= 180;
for j:=1,2:
pos_shift[1]:= radius*dir(centering[0]);
for i=1 upto chars_placed_up:
  half:=1/2 c[i];
  halfdist:= radius +-+ half;
  centering[i] := centering[i-1]
                - 2 * angle (halfdist, half);
  pos_shift[i+1]:=radius*dir centering[i];
  rot_angle[i] := angle (pos_shift[i+1]
                        - pos_shift[i]);
endfor;
centering[0]:= 180 - 1/2 centering
               [chars_placed_up];
endfor;

```

Parameters of the characters placed into the lower arch are calculated in the opposite direction using the same approach. We just need to align the upper parts of each of the characters and to move the reference point out of the base circle – hence the difference in calculating `pos_shift[i]`:

```

centering[last_down+1]:= 0;
for j:=1,2:
pos_shift[last_down+1]:=
    (radius + cap_height
     * taller_letters)
    * dir(centering[last_down+1]);
for i=last_down downto first_down:
    half:=1/2 c[i];
    halfdist:= radius +-+ half;
    centering[i] := centering[i+1]
        - 2 * angle (halfdist, half);
    pos_shift[i]:= radius*dir(centering[i])
        + (radius + cap_height
          * taller_letters)
        * (dir(centering[i+1]
              - angle (halfdist, half)))
        - radius * (dir(centering[i+1]
                        -angle (halfdist, half)));
    rot_angle[i] := angle (pos_shift[i]
                          - pos_shift[i+1]) + 180;
endfor;
centering[last_down+1]:=
    centering[last_down+1] - 1/2
    * (180 + centering[first_down]);
endfor;

```

## 2.5 Generating the characters

Now we are ready to generate the actual instances of the characters according to arrays `rot_angle[]` and `pos_shift[]`. We just need to pass the information to the appropriate procedures:

```

F_(1,rot_angle[1],pos_shift[1]);
A_(2,rot_angle[2],pos_shift[2]);
C_(3,rot_angle[3],pos_shift[3]);
U_(4,rot_angle[4],pos_shift[4]);
L_(5,rot_angle[5],pos_shift[5]);
T_(6,rot_angle[6],pos_shift[6]);

```

```
A_(7,rot_angle[7],pos_shift[7]);
S_(8,rot_angle[8],pos_shift[8]);
...
```

This font can now be used from within  $\TeX$  by saying, e.g.,

```
\char1\char2\char3\char4
\char5\char6\char7\char8
```

in order to generate the following fragment:



## 2.6 Mounting the pieces together using Metafont

It would still be clumsy to use METAFONT in order to generate the pieces of the picture, but still to have to compound them together manually within  $\TeX$  as the example above suggests. Fortunately we can do better, using the ligature mechanism of  $\TeX$  fonts. A similar trick is used within F. Sowa's `bm2font` or K. Horák's [3] method for decomposition of big METAFONT pictures.

Combinations of at least two letters from a font occurring adjacent to each other in the  $\TeX$  source suffice for invoking METAFONT's ligature program. Unlike the common ligatures used in ordinary Latin alphabet fonts, ligatures employed for this purpose make use of the fact that ligature handling is defined as simple rewriting system rewriting pairs of codes into results consisting of inserting a new character and either leaving the source characters in, or removing them. Moreover,  $\TeX$  inserts a special 'boundary' character before and after each word, including points where the font changes. Hence a simple way to define the full picture composed of the individual pieces is to define a ligature program combining the boundary character with a single letter triggering generation of the full picture. There can be several such triggers defining several parts of the picture.

Suppose for example that we want to be able to print three parts of the logo separately – the inscription, the Escher-like drawing inside of the logo, and the color areas inside of the drawing. Let us select three identifiers for this purpose – 'S' standing for 'seal', 'L' standing for 'logo', and 'C' standing for 'color'. In order the ligature mechanism to work, we add them as empty characters with zero dimensions:

```
beginchar("S",0,0,0); endchar;
beginchar("L",0,0,0); endchar;
beginchar("C",0,0,0); endchar;
```

Before designing the ligature program, let's consider one more feature of the resulting picture. So far all the characters generated had zero width so that composing them did not change the position of the reference point within  $\TeX$ . This works for every character *inside* of the composition of the picture except for the first and the last-half of the 'bounding box' of the resulting picture should be inserted there. Using slot 254

for the half of the bounding box we can define one additional character with non-trivial dimensions:

```
beginchar(254,radius#+cap_height#,
          radius#+cap_height#,
          radius#+cap_height#);
endchar;
```

Now the ligature program capable of starting everything off would have the form of:

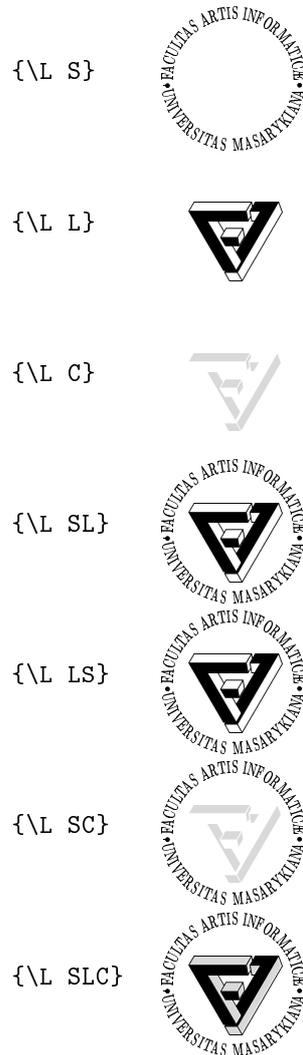
```
boundarychar:=255;
ligtable
||: "S"  =: | 254,
    "L"  =: | 254,
    "C"  =: | 254,
254: "S" |=: |> "S",
254: "L" |=: |> "L",
254: "C" |=: |> "L",
"S": "S" =: | 1,
  1: "S" |=: | 2,
  2: "S" |=: | 3,
  3: "S" |=: | 4,
  4: "S" |=: | 5,
  5: "S" |=: | 6,
  6: "S" |=: | 7,
  7: "S" |=: | 8,
  8: "S" |=: | 10,
10: "S" |=: | 11,
11: "S" |=: | 12,
12: "S" |=: | 13,
13: "S" |=: | 14,
14: "S" |=: | 16,
  ...
chars_placed_up: "S" |=: | first_down,
  ...
last_down-1: "S" |=: | last_down,
last_down: "L" |=: |> "L",
           "C" |=: |> "C",
           255 |=: |> 254,
"L": "L"  =: | 254,
254: "L" |=: | "A", %logo char
"A": "S" |=: |> "S",
           "C" |=: |> "C",
           255 |=: |> 254,
```

```

"C": "C"  =: | 254,
254: "C" |=: "B", %color char
"B": "S" |=: |> "S",
      "L" |=: |> "L",
      255 |=: > 254,

```

The font is intelligent enough to be used in such a way that after saying `\font\L=our-logo` at 2cm we can use the following input in order to define pictures of the form:



## 2.7 Driver problems

The scheme outlined above works fine except for a minor problem with certain dvi drivers which may slightly distort the resulting appearance of the complete picture. As a general rule, resetting `max_drift` to zero may be a good idea with most drivers, or else the first component may be slightly mis-aligned (alternatively one can add an empty character with zero dimensions to the beginning of every ligature chain in order to compensate for the drift with a harmless character first).

With dvips there's one more problem: it rejects empty characters with non-trivial dimensions. Before this gets fixed, the remedy may be including one pixel into the 254 character so that it's no longer empty. The pixel should be placed in a position that is set in any case. In our case there is no pixel shared by all the possible variants, hence the 254 character had to be split into some six other ones which are used depending on the context within the activated ligature chain.

Output drivers within the em $\TeX$  family exhibit even more peculiar behavior: The characters to-be-overprinted are off-paleced by positive horizontal skips so that the resulting picture gets completely distorted. Note this is not a problem with the em $\TeX$  implementation which does generate a correct dvi file in this case, but purely a problem with driver handling the somewhat unusual font rather unfaithfully.

## 3 Conclusion

We have described a method for composing images containing typesetting circular texts and pictures with sufficiently rich functionality using just the possibilities offered by definitions in METAFONT alone. The ideas used mostly derive from A. Hoenig's ideas from [2], yet are enough to locate all the necessary mechanism into a single METAFONT pass instead of invoking iterative processes involving communication between METAFONT and  $\TeX$ .

### 3.1 Acknowledgement

This work has been supported by GA ČR grant 201/93/1269.

## References

- [1] A. Hoenig. When  $\TeX$  and METAFONT talk: typesetting on curvilinear paths and other special effects. *TUGboat*, 12:554–557, 1991.
- [2] A. Hoenig. *When  $\TeX$  and METAFONT work together*. Proceedings Euro $\TeX$  92, Prague, 1992.
- [3] K. Horák. *Fighting with big METAFONT pictures when printing them reversely or landscape*. Proceedings Euro $\TeX$  94, Gdańsk, 1994.