

# 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  $\text{\LaTeX}$ , and can be printed exactly in the same way as before.

## 2 Overview

$\text{\TeX}$  and  $\text{\LaTeX}$  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  $\text{\TeX}$  community, and one would be effectively limited to CM fonts, which are the only fonts (almost) guaranteed to be present on all  $\text{\TeX}$  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  $\text{\TeX}$  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  $\text{\TeX}$  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.

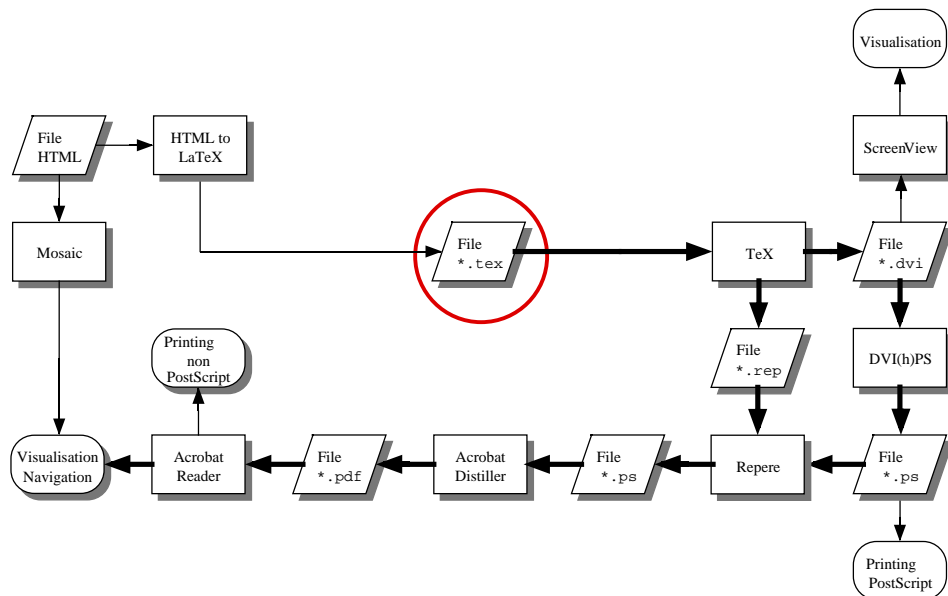
What then can we do? An obvious candidate for storing documents – at least, today – is the PDF format (*Portable Document Format*) developed by Adobe Systems. It is an extension of the PostScript language, closely resembling the syntax of files produced by Adobe's Illustrator program, with two important 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).  $\text{\TeX}$  will produce a `.dvi` file (it also uses macro files, and font metrics). If the  $\text{\LaTeX}$  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`.

Figure 1: Flow diagram for processing hypertext L<sup>A</sup>T<sub>E</sub>X files

the *hyperref* package has been loaded, the cross reference commands, bibliographic citations and indexing will produce hypertext links, included in the .dvi file with the help of `\special` commands.<sup>4</sup>

Another possible starting point is an HTML file (in the top left of the diagram). An HTML file can be easily converted to L<sup>A</sup>T<sub>E</sub>X, and the hypertext links in the former can be kept in the latter.

But let's get back to our .dvi file. We can inspect it directly, using a previewer, or a non-PostScript printer. But we can also convert it to PostScript with the *dvips* program. An extended version of *dvips* by Mark Doyle, called *dvihps* (DVI to HyperPostScript), keeps the hypertext links contained in the .dvi file. Once the PostScript file has been created, we can print on a PostScript printer (or a non-PostScript printer with the help of the GhostScript program), or convert it to PDF format using Adobe's Acrobat Distiller. This program recognizes the hypertext links and includes them in the PDF document.

Finally, to visualize a PDF document we can use Adobe Acrobat Reader, which is freely available for Mackintosh, Windows, DOS and Sun Unix. This program allows us

4. These commands have no effect on the typesetting of a page and their argument is written verbatim to the DVI file, so that they constitute excellent means to communicate information to post-processors.

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.

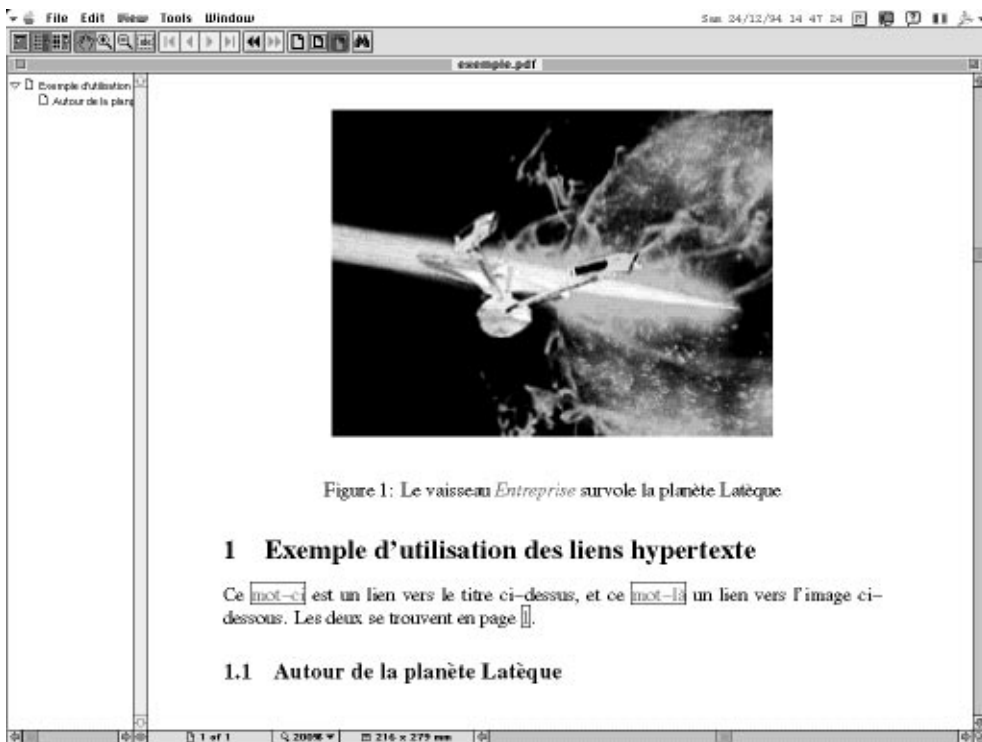


Figure 2: PDF file being displayed with Acrobat Reader

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*e 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 HyperT<sub>E</sub>X 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 HyperT<sub>E</sub>X 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.