HTML& TEX: Making them sweat*

Peter Flynn

Abstract

HTML is often criticised for its presentation-oriented conception. But it does contain sufficient structural information for many everyday purposes and this has led to its development into a more stable form. Future platforms for the World Wide Web may support other applications of SGML, and the present climate of popularity of the Web is a suitable opportunity for consolidation of the more stable features. TEX is pre-eminently stable and provides an ideal companion for the process of translating HTML into print.

1 Markup

HTML, a HyperText Markup Language [1], is the language used to structure text files for use in the World Wide Web, an Internet-based hypertext and multimedia distributed information system. HTML is an application of SGML, the Standard Generalized Markup Language, ISO 8879 [3]. Contrary to popular belief, neither SGML nor HTML is new: SGML gained International Standard status in 1986 and HTML has been in use since 1989.

SGML is a specification for writing descriptions of text structure. In itself SGML does not *do* anything, any more than, say, Kernighan and Ritchie's specification of the C language [4] *does* anything: users and implementors have to do something *with* it. It has been slow to achieve popularity, partly because writing effective Document Type Descriptions (DTDs) is a non-trivial task, and partly because software to make full use of its facilities has traditionally been expensive. It was therefore seen as a 'big business only' solution to text-handling problems until the popularisation of HTML owing to increased use of the World Wide Web. Since 1992 the software position has also improved considerably—an extensive list of tools is maintained by Steve Pepper at UIO [6].

2 The World Wide Web

WWW (W3 or just 'the Web') is a client-server application on the Internet. Users' clients ('browsers') request files from servers which are run by information providers, and display them using the HTML markup embedded in the text to render the formatting. Some of the markup can provide filenames for the retrieval of graphics as illustrations, or act as anchor-points for links to other documents, which can be further text, or graphics, sound or motion video. This latter capability gives the Web a hypertext and multimedia dimension, and allows crosslinking of files almost anywhere on the Internet.

Because the HTML files are plain text with embedded plain text markup, in traditional SGML manner, they are immediately portable between arbitrary makes and models of computer or operating system, making the Web one of the first genuinely portable, multiplatform applications of its kind.

2.1 HTML Markup

An example of simple markup and an appropriate rendering is illustrated in Figure 1. The conventions of SGML's Reference Concrete Syntax [3] are used, so markup 'tags' are enclosed in angle brackets (less-than and greater-than signs), in pairs surrounding the text to which they refer, with the end-tag being preceded by a slash or solidus immediately after its opening angle bracket.

The rendering is left almost entirely to the user's client program, as there are almost no facilities within HTML for the expression of appearance apart from a minimal indication of font change (italics, boldface and typewriter-type). Indeed, most recent browsers allow the *user* arbitrary control over which fonts, sizes and colours should be used to instantiate the tagged elements of text.

2.2 Implementation

HTML was devised by non-SGML-experts who saw it as an ideal mechanism for implementing plain-text portability while preserving sufficient structural information for online rendering: one of the classical reasons for adopting SGML. It is now becoming standardised by an IETF (Internet Engineering Task Force) Working Group who have produced a draft specification in the form of a formal DTD [1]. Because of the need to allow this specification to model existing 'legacy' documents (most of which would be regarded as fragments rather than document instances), as well as provide for more robust usage, the current DTD has two modes: a non-rigorous 'deprecated' mode for describing the legacy and a 'recommended' mode for creating and maintaining files in conventional form.

HTML is sufficient for minimal documents, providing the structural and visual features shown in Figure 2. A future version (3.0) is being developed by the IETF Working Group, which will allow the description of mathematics, tables and some additional visual- and content-oriented features.

* This paper is based on one published in *Baskerville*, vol. 5, no. 2, March 1995

```
<html>
  <head>
    <title>Fleet Street Eats</title>
  </head>
  <body>
  <h1>Where to eat in Fleet Street</h1>
  <p>There are many restaurants in the City, from
     fast-food joints to <i>haute cuisine</i>.</p>
  ...
```

Document title: `Fleet Street Eats`

Where to eat in Fleet Street

There are many restaurants in the City, from fast-food joints to *haute cuisine*.
...

Figure 1: Example of HTML markup and possible rendering

Structural	
`html`	document type
`head`	document header
`title`	document title
`base`	root address for incomplete hypertext references
`meta`	specification of mapped headers
`link`	relationship of document to outside world
`isindex`	specifies a processable document which can take an argument
`body`	contains all the text
`h1...h6`	six levels of section heading
`p`	paragraph
`pre`	preformatted text
`blockquote`	block quotations
`address`	addresses
`ol`	ordered lists
`li`	list item
`ul`	unordered lists
`li`	list item
`menu`	menu lists
`li`	list item
`dir`	directory lists
`li`	list item
`dl`	definition lists
`dt`	definition list term
`dd`	definition list description

Descriptive	
`a`	hypertext link anchor-point
`cite`	citations
`code`	computer code
`em`	emphasis
`kbd`	keyboard input
`samp`	sample of input
`strong`	strong emphasis
`var`	program variable
Form-fill	
`form`	contains a form
`textarea`	free-text entry
`input`	input field (text, checkbox, radio button, *etc.*)
`select`	drop-down menu
`option`	menu item
Visual	
`b`	bold type
`br`	forced line-break
`hr`	horizontal rule
`i`	italics
`tt`	typewriter type
`img`	illustrations
Obsolete	
`listing`	use `pre`
`xmp`	use `pre`
`plaintext`	use `pre`
`nextid`	editing control
`dfn`	definition of term

Figure 2: Markup available in HTML 2.0 (indentation implies the item must occur within the domain of its [non-indented] parent)

Despite the forthcoming improvements, HTML is likely to be joined in the Web by other DTDs. One well-known SGML software house already has a prototype browser which can handle instances of arbitrary DTDs, given sufficient formatting information. This would make it possible to use the Web for transmission and display of documents using other SGML applications such as CALS (US Military), DocBook (O'Reilly/Davenport), the TEI (Text Encoding Initiative) and corporation-specific DTDs (such as those of Elsevier).

The next version of the DTD, HTML3, contains specifications for mathematics, tables and some additional elements for content-descriptive material, as well as a few extra visual keys such as an `ALIGN` attribute for positional specification. Most of this work is being implemented on a test basis in the Arena browser (Unix/X only at the moment) at CERN.

Although Web browsers can reference files by any of several methods (HTTP, the Web's 'native' protocol; FTP; Telnet; Gopher; WAIS; and others) by using the URL (Uniform Resource Locator: a form of file address on the Internet), the most powerful tool lies at the server end: the ability of servers to execute scripts, provided their output is HTML. A trivial example is shown in Figure 3, which returns the date and time.

Such a script can contain arbitrary processing, including the invocation of command-line programs and the passing of arguments. Data can be gathered from the user either with the `<isindex>` tag in the header, which causes a single-line data-entry field to appear, or with the more complex `<form>` element with scrollable text boxes, checkboxes, radio buttons and menus. In this manner, complete front-ends can be manufactured to drive data-retrieval engines of any kind, provided that they operate from the command line, and that the script returns their output in HTML. The user (and the browser) remain unaware that the result has been generated dynamically.

2.3 Presentation

HTML is criticised for being 'presentation-oriented', but as can be seen from Figure 2, the overwhelming majority of the markup is structural or content-descriptive. However, this does not prevent the naive or sophisticated author from using or abusing the markup in attempts to coerce browsers into displaying a specific visual instantiation, primarily because none of the browsers (with the partial exceptions of Arena and `w3-mode` for GNU Emacs) performs any form of validation parsing, and will thus display any random assemblage of tags masquerading as HTML. This behaviour has misled even some eminent authorities to dismiss HTML as 'not being SGML'.

Therefore a conflict exists between the SGML purist on the one side, who decries any attempt at encoding visual appearance; and the uninformed author on the other, who has been unintentionally misled into thinking that HTML and the Web constitute some kind of glorified networked DTP system.

The purists are few in number but eloquently vocal: however, in general, they acknowledge that visual keys can be included if they are carefully coded. A perceived requirement to allow an author to recommend the centering of an element is thus achieved in HTML3 by the `align="center"` attribute, rather than the unnecessary `<center>` element proposed by the authors of Netscape.

The demands of the author are at their most marked in the approach of publishers and marketing users, who have been accustomed for the last 550 years to exert absolute control over the final appearance of their text. But the Web is not paper, and the freedoms and constraints of the Press do not apply: it is as much a new medium as radio or television. For such an author to insist that she must be able to control the final display to the same extent as on paper is as pointless as insisting that a viewer with a black-and-white television must be able to see the colours in a commercial.

The paradigm has been established that the browser controls the appearance, using the markup as guidelines. There is indeed no reason at all why attributes could not be added so that an author could write

```
<h1 color=green font=LucidaBrightBoldItalic
                        size=24 shading=50>
```

but the user of `Lynx` or `WWW` (two popular text-only browsers for terminal screens) would still only see the heading in fixed-width typewriter characters. The habit of insisting that everyone 'must' see a particular typographic instantiation is an unfortunate result of a misinterpretation of the objective of the Web: to deliver information in a compact, portable and arbitrarily reprocessable form.

But publishers accustomed to paper, insistent on 'keeping control', have of course an entirely valid point, one with which the present author has great sympathy. Why should a carefully-prepared document be made a hames of by a typographically illiterate user who has set `<h2>` to display as 44pt Punk Bold in diagonal purple and green stripes?

```
#! /bin/sh

echo Content-type: text/html
echo

cat <<EOH
<html><head><title>Date and time</title></head><body><p>It is now
EOH
date
cat <<EOT
</p></body></html>
EOT
```

Figure 3: Example of a Unix shell script to return the date and time as an HTML file

The solution probably lies in the implementation of style sheets, perhaps along the lines of those discussed by the authors of Arena [5]. They would in any case only be recommendations: not every user has a CD-ROM of Adobe or Monotype fonts. In any event, if 100% control is essential, as in the display of typographic examples, all graphical browsers can be configured to spawn a window to display a PostScript file, although the download time may be a strong disincentive.

It is entirely possible that the control of content will ultimately prove a more attractive option than the control of appearance.

3 Publishing with HTML

Setting aside the unresolved questions of display, there are more pressing business problems about publishing on the Web.

The authentication of users can be addressed at several levels, from simple non-authoritative checks using `identd`, to the more complex username-and-password systems employed on some Web pages. From the user's end, the authentication of the data being accessed is equally important. The openness of the Internet in its raw form allows 'spoofing' in both directions, so the emergence of protocols to provide checks is to be welcomed.

The security of network-accessible texts from break-ins remains a concern to anyone providing high-value merchandise, and Web text is in this sense no different from any other computer data. Normal precautions must therefore be taken to prevent theft through other channels (such as remote login), as distinct from theft perpetrated by falsification of Web access.

There is a need for robust solutions to charging and billing for usage, and the secure transmission of financial data, including credit card numbers, digital signatures, and perhaps even EFT transactions. The Secure HTTP (SHTTP) mechanism being marketed by MCom and others is becoming popular as a way of achieving some of this, but the Internet must shed some of its image of lax controls and sloppy housekeeping if it is to achieve sufficient 'respectability' to attract the business of those who are not networking specialists.

The handling of copyright and the intellectual property of electronic texts remains, as ever, an unsolved problem. While copyright law can be used to provide a remedy for breach, the difficulty lies in preventing the breach occurring in the first place. The reason is that (as with other electronic material), copying and reproduction is fast, cheap and easy, once the material is in the hands of the customer. While a supplier may use SHTTP to protect the details of the transaction, once a print file has been sent to someone, the supplier retains no control whatsoever over its use, reuse and abuse. Copies could be sent to dozens of others, or printed many times, in the space of minutes.

3.1 Printing from HTML

The demand for printed copies of Web material is surprisingly high. Although in some cases it is reminiscent of those people who insist on printing their email, it is undeniable that there is a serious requirement for good quality print from Web documents.

Existing solutions to printing SGML text are usually application-specific, embedded in SGML editors or DTP systems, but there are also some more generic packages:

- `Format` by Thomas Gordon (LaTeX)
- `HTMLtoPS` by Jan Kårrman (PostScript)
- `SGML2TeX` and `WebSet` by Peter Flynn (TeX/LaTeX)
- `SimSim` by Jonathan Fine (TeX)

The use of TEX systems for most of these seems to indicate that the similarity of markup concepts has not gone unnoticed by practitioners. The author's own contributions are experimental, but `WebSet` is planned as an interactive Web service, to be introduced in the summer of 1995. Emailing a URL to the point of service will cause it to be retrieved, typeset, and the output returned to the user by email in PostScript form. As a form of email browser, the control of appearance may lie in the hands of the user, but suggestions for how to implement this are currently being sought [2].

3.2 Problems

Implementing a professional level of typesetting from HTML raises some interesting questions:

- most HTML files are invalid
- most HTML authors don't understand SGML
- most HTML authors couldn't care less
- most World Wide Web users couldn't care less

The handling of missing, damaged or abused tags in a gracious manner is not a feature of most SGML parsers. At the best, a typesetter-browser can only be expected to report to the user that a file is invalid, and while it may be displayed by browsers which do not make any claim to typographic quality, an attempt to make a respectable print job of an invalid file is unlikely to succeed.

4 Development

The future of the World Wide Web and HTML is uncertain. While development continues, and while new users are anxious to start surfing the net, the existing designs and implementations will suffice. In the longer term, a coalescing of services is likely to occur, but for this to happen, a number of changes need to take place:

- The Web will start to make use of other DTDs, as outlined above. Any file containing a document type declaration (i.e., `<!doctype...>`) at the beginning could cause a browser to retrieve the DTD specified, along with a style sheet, and work much as any SGML-conformant DTP system would.
- Browsers will become pickier, able to offer better services at the expense of rejecting invalid or badly broken files. Arena already perfoms a form of consistency check on the HTML code of files, and displays 'Bad HTML' in the top corner when an offender is spotted.
- Users will become pickier, demanding better response from the browser, better response from the server, and better facilities from both. As users become more educated about the use of SGML, developers will no longer be able to hide the deficiencies of products under the cover of technical detail.
- This presupposes more user education, which is inevitable in a developing technology. One hundred years ago, motor cars appeared on the roads, but few passengers in them understood the use of the levers and rods which controlled them. With some minor exceptions, it is now expected that a driver knows that turning the wheel clockwise turns the car to the right, and *vice versa*. It will not take us that long to perceive the innards of HTML, but it can only be done by training and education.
- At some stage, investment is always needed. Many companies have invested substantial sums into the development of Internet resources, and those that have done so with forethought and planning deserve to reap a rich reward. It is a long-term investment, more akin to a partnership, but support is always needed by those who undertake the developments, especially as much of it is done in personal time and at personal expense.

There is still some way to go before we achieve the ease of use of the telephone or the radio, but the path is becoming easier with each new development.

References

[1] Berners-Lee T. & Connolly D. *HyperText Markup Language Specification — 2.0*, Internet Draft, IETF Working Group on HTML, December 1994.

[2] Flynn P. *Typographers' Inn*, TEX and TUG News, **4**, 1, March 1995.

[3] Goldfarb C. *The SGML Handbook*, OUP, 1990, ISBN 0-19-853737-9.

[4] Kernighan B.W. & Ritchie D.M. *The C Programming Language*, Prentice-Hall, 1978.

[5] Lie H. *et al. HTML Style sheets*, `http://www.w3.org/hypertext/WWW/Style/`

[6] Pepper S. *The Whirlwind Guide: SGML tools and vendors*, `ftp://ftp.ifi.uio.no/pub/SGML/SGML-Tools/SGML-Tools.txt`

◇ Peter Flynn
Computer Centre
University College
Cork, Ireland
Email: `pflynn@curia.ucc.ie`

The Inside Story of Life at Wiley with SGML, LaTeX and Acrobat*

Geeti Granger

1 Introduction

John Wiley & Sons is a scientific, technical and medical publisher. It is an independent, American family-owned company that was established in 1807, with subsidiaries in Europe, Canada, Australia and Singapore. The European subsidiary opened in London in 1960 and moved to Chichester in 1967 (if folklore is to be believed this was so that the then Managing Director could more easily pursue his love of sailing!).

We publish books, including looseleaf and encyclopaedias, and journals, and most recently electronic versions of some of our printed products. In the future the electronic component of our publishing programme is bound to include products that are only available electronically.

2 Setting the Scene

To the topic in hand—Portable Documents: Acrobat, SGML and TeX. Our association with TeX dates back to 1984 when we made the significant decision to install an in-house system for text editing and composition. It was the only software available that wasn't proprietary, which stood a chance of coping with the complex mathematical material we had to set.

As a company we have monitored the progress of SGML since 1985, but have only recently used it in earnest. Our first project is a 5000 page encyclopaedia about Inorganic Chemistry. We rarely get the opportunity to dip our toes in the water—it's straight in at the deep-end! Having said this, we do have a set of generic codes that has been used for a number of years, and everyone is well aware of the principles involved and the value of this approach to coding data.

Adobe Acrobat was launched in June 1993. Our experience of this software dates back a little further than this, because of our links with Professor David Brailsford and the Electronic Publishing Research Group at the University of Nottingham, and their work on the CAJUN (CD-ROM Acrobat Journals Using Networks) project, which we jointly sponsored with Chapman & Hall.

* This paper is based on one published in *Baskerville* vol. 5, no. 2, March 1995.

3 Complementary not Competitive

The first thing to make clear is that SGML, TeX and Acrobat do not compete with each other in any way. SGML is a method of tagging data in a system-independent way. TeX is one possible way of preparing this data for presentation on paper, while Acrobat is software capable of delivering data electronically for viewing on screen, or for committing to paper.

From our point of view the fundamental requirement for:

- capturing data
- processing data (text and graphics)
- delivering data (paper/disk/CD/Internet)

is to remain system independent for as long as possible.

SGML, TeX and Acrobat achieve this in their part of the whole process. PostScript provides the link that completes the chain.

4 SGML in Practice

To describe our experience with SGML I will use the *Encyclopedia of Inorganic Chemistry* as a case study. This encyclopaedia is an 8 volume set made up of 5000 large-format, double-column pages (more than 3 million words). The data consists of approximately 250 articles interspersed with 750 definitions and 750 cross-reference entries. The text was marked-up and captured using SGML, validated and preprocessed for typesetting. The floating elements (all 2300 figures, 8000 equations, 2000 structures, 1100 schemes and 900 tables) were prepared electronically and delivered as encapsulated PostScript files. Some 150 halftones, about a third of which are colour, complete the data set!

Despite the complex nature of this project, or maybe because of it, we were convinced that using SGML was the right approach. We had to be very sure because this decision presented us with many additional difficulties. Different considerations had to be made at all stages of the production process. (Manufacturing remained untouched.)

Once we had established the probable requirement for an electronic version, there was the need to justify the use of SGML because of:

- the extra cost involved in data capture
- the different working practices that had to be established
- the project management overhead
- the need to find new suppliers, and the risks that this involved for such a large, high profile project.

4.1 Production Considerations

This project had an external Managing Editor to commission and receive contributions before it became a live project. Once contributions started to arrive it very quickly became apparent that a project management team was needed if this project was to succeed. The initial steps had to be ones of project analysis, determining data flow, deciding who was responsible for what, and ensuring that a progress reporting system was established. It certainly semed like a military operation at times.

Having made the decision to go with SGML and to ensure that all components were captured electronically we had to find a set of new suppliers. None of our regular suppliers could meet our specifications. Locating potential suppliers was the first hurdle, and then assessing their suitability was the next. Having done this we then had to draw them all together to establish who did what, and who was responsible for what. It had to be a team effort from start to finish and regular progress meetings involving representatives of all parties was the key to an ultimately successful project.

4.2 Problems Encountered

One of the first considerations was how on earth do we name the files? To ensure portability we set ourselves the restriction of the eight plus three DOS convention. It took some time but we achieved it in the end so you can now identify from the file name the type of text entry, the type of graphics and whether it is single or double column or landscape, and its sequential placement within its type. When you consider the number of files involved, this was no mean feat.

Designing the DTD without all the material available is not the best way to start, but needs must. It meant that some amendments had to be made as the project progressed but none of them proved to be too significant.

Choosing Adobe typefaces, to avoid problems later on, meant that some compromises had to be made. Many people feel that the Adobe version of Times is not as elegant as some other versions of the typeface.

Also the quality of the typesetting, hyphenation and justification, interword spacing and overall page make-up is not as high as that normally achieved by a dedicated chemistry typesetter.

In addition to the above, we found a bug in Adobe Illustrator! Because the EPS files were being incorporated electronically the accuracy of the bounding-box coordinates was crucial. To cut a long story short they weren't accurate. We spent quite some time establishing the cause of the problem and then had to have a program written to resolve it.

This is not an exhaustive list but I think it will give you a feel for the practical issues involved. Having shared all this with you I should add that all of us involved in the original recommendations remain convinced that it was the right approach. In fact we are now processing two more projects in the same way!

5 LaTeX in Practice

We have done far too many projects in TeX (many in Plain, but a growing number in LaTeX) to select one as a case study. What I can do is very readily identify the production issues involved in using this software in a commercial environment.

5.1 Steps in the Process

Establishing ourselves as a forward-thinking, progressive company by developing in-house expertise has brought with it certain pressures. In the early days, not only did we have to learn how to use TeX, we also had to make it achieve typesetting standards expected of more sophisticated systems. Our colleagues could not see why they should accept lower standards from us—after all they were paying us (we operate a recharge system so that it doesn't distort the project costing when compared with externally processed projects).

Next came the requests for us to supply style files. Authors knew we used the same software as they did, and wanted to prepare their submission so it looked like the finished product. Some wanted to produce camera-ready copy. In principle this would seem a sensible idea; in fact our commissioning editors, especially those who handle a number of CRC projects, thought it was a brilliant idea. It would save them an immense amount of time and hassle.

Now, preparing style files for in-house use is one thing; preparing them for use by others is something else again. We have to work within strict time and cost constraints, and there are many occasions (dare I admit it?) when we have to resort to, shall we say, less than the most sophisticated way of achieving the required visual result!

When I have attended courses on TeX and have asked about writing style files the answer has often been along the lines of 'leave it to the professionals'. (I should say it's usually people who make their living in this way who give this response.) This may be fine if *a*) you can find and afford the professional; *b*) you don't need to support the file when it is in general use. In our experience the first is difficult

to do and the second is an impossibility. The need to support style files cannot be ignored; once they have been provided, no matter on what pre-agreed conditions, queries will arise. It can be very time-consuming, as often queries are not restricted to the style file, but relate to the sytem being used. It can also take a while to establish the context of the query, resolve it and respond. To meet the expectation that we will support, customise at short notice, resolve technical issues, and communicate via e-mail (preferably responding within the hour) can be difficult, given the level of human resource available.

Once you've got over this initial stage, the practical issues involved in accepting LaTeX submissions can be many. Delivery is the first. Now that we have the ability to receive data electronically our authors cannot understand why we hesitate, and why we still insist on hard copy. Experience tells us that, without hard copy, it is difficult to be sure we have received the final version, and discovering this after a project has been processed is very costly, both in time and money. Any submission that circumvents a stage in the current administration process may drop through a hole and end up taking more time, rather than less, to reach publication. Consideration is being given to this issue, and there is no doubt that in the future electronic delivery will be an acceptable method of submission, but in the meantime everyone has to be patient.

Copy-editing remains a conventional process in the main, although experiments are taking place with copy-editing on disk. This issue is not restricted to LaTeX projects, but the rate of progress is dictated by the ability of our freelance copy-editors to provide this service.

Once you move on to the processing stage the first thing you have to do is find a supplier who is capable of actually processing in this software. This is easier said than done, because it is not considered to be cost-effective by most of our regular suppliers. However, as a result of our persistent requests, some can now provide this service, so we don't have to process all such submissions in-house.

From our own experience we know that producing page proofs is not always straightforward. Over the years we have struggled with amending style files to achieve the correct layout and controlling page make-up. Now that authors are submitting graphics on disk, as well as the text, we are faced with another set of problems. Portability of graphic formats is even more difficult to achieve. I think the number of answers to the question 'When is a PostScript file (or EPS file) not a portable PostScript file?' must be infinite. Even when the content of the file itself is OK, you can still be faced with problems in achieving the required size and position on the page.

Despite all these disadvantages our lives would not be the same without LaTeX, and when compared with processing in other software it can be a real joy! Our archive of projects coded in a form of TeX will be far easier to reuse than those processed in other software.

6 Acrobat at Arm's Length

Although we haven't used Acrobat on a live project in-house yet, we have been closely involved with the development of the EPodd CD. The CAJUN project has been running for well over a year and during this time the complete archive of volumes 1–6 has been converted to PDF, annotated to add `pdfmark`s and generally massaged into a suitable format for delivery on CD.

As always, the work involved in such a project is more than anticipated at the outset, but it has been an invaluable learning exercise. Being involved in the beta-testing of the software helps you appreciate just how much development work is required for a new piece of software, and although it currently has its limitations the future looks good. Version 2, which (at the time of writing) is due for release any day now, is much improved, and it is rewarding to see that many of the comments put forward by members of the team have been incorporated.

We are experimenting with small projects in-house to give us a deeper understanding of the practical advantages and limitations of Acrobat. It is easy to get caught up in the euphoria and hype that accompanies the release of a new product, and to overlook the day-to-day difficulties its rapid adoption might bring. Having said this, there is no doubt that it will have a place in our publishing procedures, and may be used in the production cycle for journal articles. Provided that the general administration can cope with the deviation from the norm, supplying author proofs in this way has its attractions. The fact that readers are now freely available and the PDF file can be read on any of the three main platforms is a real boon.

The use of Acrobat for delivering existing print products in an electronic form is one worth considering, especially now that it is possible to integrate it with project-specific software and the security issue has been addressed.

From an inter-company point of view the perceived use of Acrobat for distributing internal documents could again have its attractions. For this to be a real possibility it must be recognised that the

use of such procedures is not an innate skill, and so the appropriate level of training and support must be available if it is to be successful.

7 Conclusion

The comments I have made and the case study I have described may leave you with a somewhat negative feeling. I wonder if I have emphasised the problems and not balanced these by identifying the plus points. To put this into context I should say that details of the advantages of any particular approach are usually more readily available, so I have tried to capture a more down-to-earth view.

In reality I am very enthusiastic about the use of SGML, TEX and Acrobat, but am also well aware of what their use in a productive environment can mean. I believe, as do several of my colleagues, that portability of documents is crucial to our ability to deliver data efficiently in a variety of forms, whether this be page-based, highly structured databases or tagged ASCII files. To this end we must be flexible in our approach, and must not be afraid of making investments now that may not bear fruit until some time in the future. This can be a very unnerving decision to make, and for one I am glad it isn't ultimately mine. While I can extol the virtues of a purist's technical approach, obtain the relevant costs and assess the schedule implications, I do not have the entrepreneurial skills required to know when a project is commercially viable (or worth taking a risk on). It is at this point I take my hat off to our commissioning editors, who have the responsibility for turning these experiments into profit for us to reinvest in the next Big Thing!

◇ Geeti Granger
John Wiley & Sons Ltd
Baffins Lane
West Sussex
Chichester PO19 1YB, UK
Email: granger@wiley.co.uk

The Los Alamos E-print Archives:* HyperTEX in Action

Mark D. Doyle

Abstract

The Los Alamos E-print Archives houses more than 25,000 research papers in about 25 fields of physics and mathematics, with the vast majority written in TEX. This paper describes HyperTEX and how it is transforming the archives from a loose conglomeration of independent papers into a single, large hyperlinked database available via the World Wide Web.

1 Los Alamos E-print Archives

The Los Alamos E-print Archives were created in 1991 by Paul Ginsparg. In the beginning there was a single archive dedicated to High Energy Physics Theory, but now it has grown into a collection of over 25 archives, each dedicated to a fairly narrow field in physics, mathematics, economics, or computation and linguistics. The archives contain over 25,000 papers, with over 90% submitted as TEX source (for some archives, including the largest, the figure is over 99%). The rest of the papers are submitted as PostScript, and almost all of that is generated by TEX/`dvips`. We expect that Adobe's Portable Document Format (PDF) will start to appear over the next year.

We have recently implemented an auto-TEXing script that processes over 90% of the TEX source into PostScript (failures are due to careless submitters who don't bother checking that their source was transmitted correctly via email or who didn't supply all of the necessary style/macro files). Soon TEX-ability will become a criterion for accepting a paper onto the archives (it is already an effective "referee", correlating well with the scientific quality of the work).

The archives are accessible via electronic mail (`arch-ive@xxx.lanl.gov`, where `arch-ive` is one of the archive names, e.g. `hep-th`), anonymous ftp (`ftp://xxx.lanl.gov/`), and the World Wide Web (`http://xxx.lanl.gov/`). Submission always involves email because most WWW browsers do not yet allow files to be sent even though the HTTP protocol includes this capability. WWW usage has grown exponentially and our server gets about

* Supported by the U.S. National Science Foundation under Agreement No. 9413208 (1st March 1995 to 28th February 1998).

20,000 hits per day now (see `http://xxx.lanl.gov/cgi-bin/show_weekly_graph`). Another measure of the vitality of the archives was noticed when we put the auto-TEXing script on-line in June, 1995: fully one third of the papers were accessed during that month. Furthermore, in some fields (High Energy Physics for example), the archives have effectively replaced the traditional print journals as the primary means of accessing new research.

The use of the World Wide Web has greatly enhanced the accessibility of the archives and we have actively developed HyperTEX to further enhance the on-screen reading of the papers. HyperTEX creates hypertext documents and, with the proper viewers, allows links to other documents via a World Wide Web Uniform Resource Locator (URL).

2 HyperTEX

On-screen reading of information is greatly enhanced by hypertext functionality. For instance, a paper with mathematics has the equations numbered sequentially and the reader is often referred to another equation via its number. Hypertext functionality allows the reader to use the mouse to click on the numbered equation reference and either jump back to the referenced equation or display it separately in a new window. Similarly, clicking on a citation to a reference listed in the bibliography should bring up the bibliographic entry, and if the entry refers to another paper on the archives, say, then clicking on it should bring up the abstract of that paper in your World Wide Web browser.

So how do we produce something with hypertext functionality from over 25,000 TEX papers?

This question came to the forefront for Paul Ginsparg in the late fall of 1993 when he saw a demo of Adobe's newly introduced Portable Document Format and their Acrobat PDF viewer. The sample was 150 pages of TEX-produced lecture notes by Ginsparg that were distilled into PDF. During the demo it was demonstrated that hyperlinks could be added so that the table of contents would be linked to the proper sections. And here is the reaction:

> "... horrifying to contemplate armies of people adding hyperlink overlays "by hand" after the fact, especially when much of the contextual structure is already present in the TEX source, only to be lost in the conversion to dvi and then e.g. to PostScript."

Were the typesetters displaced by TEX destined to become hyperlinkers?

Any solution to the problem of converting TEX into hypertext should satisfy at least these three criteria:

- Take advantage of contextual information already implicit in TEX documents
- Provide interoperability with the World Wide Web
- Maintain the high quality of TEX's output

By contextual information, I mean the information implicitly present in the association of a label with an object and the subsequent use of this label as a way of referring to that object. Examples of this are the way that TEX handles equation numbering and citations.

One possible solution which has attracted interest is to convert TEX/LATEX into HTML, the hypertext markup language used by the WWW. Then one would read a paper directly in a WWW browser. However, for material with a lot of mathematical content, this conversion fails to meet the third criterion above.

HyperTEX provides a better solution. The central idea is to export the contextual information into the DVI file via TEX's `\special` command. To do this we modify the basic macros for equation numbering, citations, footnotes, tables of contents, indices, etc., to output appropriate `\special`'s. This was first done by Tanmoy Bhattacharya for the standard LATEX styles and some of the physics styles like RevTeX. Paul Ginsparg also modified his plain TEX harvmac macros into lanlmac providing complete HyperTEX functionality.

Having the contextual information in the DVI file doesn't do much good if there isn't a way to take advantage of it. DVI previewers need to be modified, as well as DVI drivers. Arthur Smith modified `xdvi` into `xhdvi` giving the first Hyperdvi previewer. With the help of Tanmoy Bhattacharya, I modified Tom Rokicki's `dvips` into `dvihps`. Initially the goal was to produce PostScript that would be distillable into PDF by the Adobe Distiller. This was accomplished using the Distiller built-in command `/pdfmark`. However, it was quickly realized that this new "HyperPostScript" could be an end unto itself. The need for a format like HyperPostScript was necessitated by the fact that Adobe has been slow to provide things like WWW access from their readers. Tanmoy then hacked `ghostview` into a HyperPostScript viewer that communicates with WWW viewers.

The upshot of this is that TEX can be transformed into hypertext in three different, parallel formats: Hyperdvi, HyperPostScript, and PDF. The

first two are finding uses because everything is public domain and we are free to enhance the tools as necessary. On the other hand, PDF is currently produced only via Adobe's commercial Distiller and one is limited by whatever functionality Adobe chooses to provide. Still, it would seem that ultimately PDF will be the dominant endpoint for HyperTEX source since PDF viewers are now widely available, and Adobe continues to enhance the PDF standard so that things like WWW access are becoming well-integrated.

HyperTEX has quite a few positive features. First, it preserves all of the contextual information present in the TEX source. There is no need for complex conversions of pre-existing TEX files into HTML, and no need to wait for a future version of HTML with good support for mathematics. TEX's high quality output is retained, and the printed version is unchanged (HyperPostScript is designed so that it can be rendered by any PostScript interpreter). By modifying macro packages in a way that preserves the keywords, HyperTEX allows the creation of hypertext with little or no effort by the author. In fact, it is completely backward-compatible, and can be applied retroactively to TEX documents already in existence. In most cases, a single line addition converts TEX into HyperTEX.

For the E-print Archives, this means that we can turn old papers into hypertext. Even better, since our Hyperdvi and HyperPostScript viewers can communicate with WWW browsers, we can automatically translate references to other papers on the archives into URLs for the referenced paper through a simple substitution in the TEX source as we process it. Providing the archive reference (for example, hep-th/9201076) for a cited paper has become increasingly popular and as this practice grows, an increasing fraction of the archive becomes woven together into a single large hyperlinked database.

Since HyperTEX does not depend on authors modifying their source, it has few drawbacks: one problem is that Hyperdvi and HyperPostScript viewers are not available on all platforms. This is partially offset by the fact that the platforms without active viewer development are the same platforms where PDF is quickly becoming a dominant format for document exchange. HyperTEX is not a universal solution for producing hypertext. Hypertext is often not linear and with TEX being so "papyrocentric", it is not easy to see how to apply it to general hypertext. Still, HyperTEX fills its niche very well and has turned out to be quite useful.

```
\special{html:<a name="section.1">}{1.}{
      \special{html:</a>}
\special{html:<a href="#section.1">}{1.}{
      \special{html:</a>}
\href{http://xxx.lanl.gov}{This http URL}
```

Figure 1: HyperTEX

3 HyperTEX in a Nutshell

The following is meant to be a brief overview of how HyperTEX works and how the contextual information is passed along. Most of the following is taken from the HyperTEX FAQ maintained by Arthur Smith (`ftp://snorri.chem.washington.edu/hypertex/`).

HyperTEX adds five new `\special` commands:

```
\special{html:<a name = "namestring">}
\special{html:<a href = "hrefstring">}
\special{html:</a>}
\special{html:<img src = "hrefstring">}
\special{html:<base href = "hrefstring">}
```

to which a reference can be made. This is used, for instance, to turn an equation's number into an anchor. The 'href' `\special` then allows a refence to be made to an anchor established by the 'name' `\special`. But it is also more general than that because you can put use an URL as the 'hrefstring' and this will be interpreted as something to pass off to a WWW browser. The third `\special` is for ending the others, and is used to delineate the text associated to a 'name' or 'href' that should appear on the page. The fourth `\special` is for including images, but none of the viewers or drivers currently deal with it. The final `\special` is for making references to other documents easier. It allows you to change the 'base URL' to which all following 'href' `\special`'s should be considered relative (the default is that a relative 'href' refers to an item in the current document).

A convention for naming links within documents has also been given so that it is easier to refer to items in other documents:

Page 5 is at	doc.dvi#page.5
Section 2 is at	doc.dvi#section.2
Equation 3 is at	doc.dvi#equation.3
Reference 11 is at	doc.dvi#reference.11

The items in the righthand column are those which would appear in place of the 'hrefstring' in an 'href' `\special`.

Now let's take a quick look at how the contextual information is represented and passed along in the various formats. First we consider HyperTEX

```
HPSdict begin
/TargetAnchors
605 dict dup begin
...
(section.1) [5 [72 706 83 718] 792] def
...
end targetdump-hook def
...
(#section.1) [[72 627 81 639]
             [1 1 1 [3 3]] [0 0 1]] pdfm
```

Figure 2: HyperPostScript

itself. The example (Figure 1) shows how a section heading might be made into an anchor that is linked to the section number (1 in this case). Only the '1.' is printed on the page. Both the name `special` and an href `\special` that might refer back to it are shown in a raw form without macros. Normally a command like `\section` would just put in the proper information transparently. Also shown is how an external reference can be handled by a macro `\href` that hides the `\special`s. The URL is given in the first set of braces, the text that appears on the page appears in the second set. The information in the `\special` commands is just stamped into the DVI file at the proper place, as can be seen here:

```
html:<a
    href="#section.1">\2531.\357html:</a>
```

Passing the DVI file through `dvihps` produces HyperPostScript, as shown in Figure 2. While this might look complicated, it is quite straightforward. The first few lines create a dictionary that stores all of the anchors created by the name `\special`s. In this case, the dictionary has 605 anchors (it is from Ginsparg's 150 pages of lecture notes written with harvmac in 1988 and turned into HyperTEX by changing from the harvac macros to lanlmac). Somewhere in the dictionary there appears the line associated with the name `\special` for section 1. The information that follows it is an array giving the page number on which the anchor appears, the coordinates of the rectangle in which the text associated with it appears, and a number that can be passed to the Distiller so that the PDF viewer zooms to a region of the page containing the anchor (in this case, `792` means zoom to the top portion of the page). Later in the HyperPostScript there is a reference created by the href `\special`. Note the hash mark that distinguishes this as a link. The array that follows contains the information needed to highlight the link as something clickable: the rectangle containing the text associated with the link, an array given the type of border to draw (in this a dashed box), and the color to use for the box (blue). Then there is the `pdfm` operator.

```
1040 0 obj <<
/Type /Annot /Subtype /Link
/Rect [72 627 81 639]
/Dest [23 0 R /FitH 792]
/T (#section.1)
/C [0 0 1 ]
/Border [1 1 1 [3 3 ] ] >> endobj
```

Figure 3: PDF

All of the magic of HyperPostScript is contained in the definition of the `pdfm` operator which is contained in the header file `hps.pro` that is embedded in the prologue of the HyperPostScript file. In particular, the operator is smart enough to just get rid of all of this if the file is being interpreted by an ordinary PostScript interpreter. Otherwise, it tries to figure out the version of the PDF Distiller being used and then it transforms the information for the link into the format needed for that version of the Distiller and incorporates it into a proper `pdfmark` (part of the information is here and part is in the `/TargetAnchors` dictionary entry for section 1). HyperPostScript viewers can also define the `pdfm` operator for their own use.

Looking at the PDF version of the same information should make this clearer. A PDF file consists of "objects" that are written in a slimmed down PostScript. In particular, there is an object for each hypertext link and there is an object (number 1040) for the link in our earlier example. The interpretation of Figure 3 is straightforward. We have an object that is an annotation, specifically a link. The box for it appears in the rectangle shown, the title or name of the link is `section.1`, the box should be blue (`/C` is color), and the border should be dashed. The only thing that isn't immediately obvious is the destination. `23 0 R` means that the destination is object 23 (which in this case would be page 5). The `/FitH` means that the viewer should zoom so that the page's horizontal width is expanded to the size of the viewer's window, and the `792` means scroll so that the coordinate 792 is at the top of the viewer's window. In this case, 792 means the top of the page (72 PostScript units per inch $\times$ 11 inch page height).

The key point is that almost all of the information from the TEX file is there. The main deficiency of the PDF compared to the other formats is that the destination is only a page number and where on

Macro Packages:

hyperbasics.tex Basic set of macros for implementation of the HyperTEX `\special`'s (*Tanmoy Bhattacharya*)

lanlmac.tex Plain TEX macro package (*Paul Ginsparg*)

hyperlatex.tex Variety of `.hty` files for different LATEX styles are available. Note that this does not work with LATEX 2ε because it uses undocumented LATEX internals. (*Tanmoy Bhattacharya*)

hyperref.dtx Completely new, but compatible, implementation for LATEX 2ε (*Sebastian Rahtz* and *Yannis Haralambous*)

hyper.dtx Similar to hyperlatex, but for LATEX 2ε (*Michael Mehlich*)

Hyperdvi Previewers:

xhdvi Extension of `xdvi` for X-Windows (*Arthur Smith*)

HyperTeXview Extension of Tom Rokicki's TeXview for NeXTSTEP (*Mark Doyle*, based on early version by *Dmitri Linde*)

DirectTeX A full Macintosh implementation by *Wilfried Ricken* which supports HyperTEX

Hyperdvi to HyperPostScript:

dvihps Extension of Tom Rokicki's `dvips` to produce HyperPostScript Distillable into hyperlinked PDF (*Mark Doyle*, with assistance by *Tanmoy Bhattacharya*)

ghostview Hacked version of GhostView to support HyperPostScript (*Tanmoy Bhattacharya*)

Figure 4: Current HyperTEX Tools

that page to zoom. The name of the target and its precise location on the page have been lost. Adobe has recently extended the PDF standard to include named destinations, so it is likely that `dvihps` and the `hps.pro` will be updated to present the information in a different but equivalent manner. Various features of the PDF could also be configurable. Examples would be the color of the box or how to zoom to the anchor, and newer versions of `dvihps` and the HyperTEX macros will allow this.

4 The Future of HyperTEX

Before giving some future directions, it would be useful to summarize the HyperTEX tools that are already out there (see Figure 4 for a list; all are public domain and can be found on the net; pointers will be given at the end of this article). There is still plenty of work to be done. The macros, while quite usable, can always use improvement. There are some fundamental problems that need to be handled in a better way (notably line breaks, page breaks, and footnotes breaking across pages). The footnote problem is the trickiest and work is being done by the DVI standards TEX Working Group to provide a standard way of handling this situation (the use of color in TEX has similar problems). The viewers can also use improvement, and support is still needed for the 'image' `\special` and the 'base' URL `\special`.

Now that Adobe finally is coming out with support for URLs, `hps.pro` needs to be enhanced to output the information in a way that the Adobe Distiller can use it. The conversion from Hyperdvi to PDF could also stand some improvement. Right now `dvihps` doesn't give any options for color, images, or other PDF features like bookmarks, etc. None of this is particularly difficult. There are still a few sticky points having to do with TEX and Adobe's Distiller. In particular, the Distiller (as of 2.0 anyway) optimizes away 'blanks' (character code 32) which are really glyphs in TEX fonts (e.g. the Greek letter ψ). There are workarounds for this problem though. Perhaps the most ambitious solution to these problems would be to write a real DVI to PDF converter that completely bypasses the Adobe Distiller. This is rather difficult, but it would free us from having to use the Distiller which is the only commercial product in the whole chain. In the meantime, enhancements to HyperPostScript viewers could obviate the need to go all the way to PDF.

5 Conclusion

For such a simple idea, HyperTEX works amazingly well. It makes the on-screen reading of TEX documents easier and allows TEX to interact with the World Wide Web. All of this while preserving the superior formatting and typesetting of TEX. PDF generation gives good results and can be completely automated. Finally, HyperTEX has turned the Los Alamos E-print Archives into a hyperlinked database of over 25,000 papers.

Acknowledgments

I would like to thank Tanmoy Bhattacharya and Paul Ginsparg for helpful discussions regarding all aspects of HyperTEX.

HyperTEX Resources

- On the Web drop in on `http://xxx.lanl.gov/hypertex/`
- FTP locations:
  `ftp://xxx.lanl.gov/pub/hypertex/`

`ftp://gita.lanl.gov/people/tanmoy`
`ftp://gita.lanl.gov/people/doyle`
`ftp://snorri.chem.washington.edu/pub/hypertex/`
`ftp://ftp.shsu.edu/` and other CTAN sites

- Listserver and mailing lists maintained by Arthur Smith, `majordomo@snorri.chem.washington.edu` (requests go in body of message)

Announcements:	subscribe hypertex-announce
Developers:	subscribe hypertex-dev
Email archive:	by email request to listserver

◇ Mark D. Doyle
Los Alamos National Laboratory
University of California
Los Alamos, New Mexico
Email: `doyle@mmm.lanl.gov`

The Hyperlatex Story

Otfried Schwarzkopf

Abstract

Hyperlatex is a little package that allows you to use a LaTeX-like language to prepare documents in HTML, and, at the same time, to produce a neatly printed document from your input. It is possible to use arbitrary LaTeX commands for the typesetting of the printed output by including them in the input file.

About two years ago my drawing editor Ipe[1] was getting sufficiently complex to merit a real manual instead of a simple `readme` file. Of course, Ipe should be able to show its manual on-line, but on the other hand I also wanted to be able to print a well-formatted manual on paper. My first attempt at this used the latexinfo system to write the manual, so I was able to print it nicely and to have the on-line version as an `info` file. However, `info` files are simply text files, and it is impossible to include figures in the on-line manual. Quite a shortcoming when you are trying to write a manual for a figure editor! The second problem was that the first Ipe users were Emacs-illiterate, and they found it very hard to cope with the `info` reader.[2]

At that time HTML and HTML-readers like Mosaic became widely used. These readers solved both problems — an HTML document can include figures, and HTML readers are basically designed to be foolproof (how else could one explain the success of the World Wide Web?). So, as the next step, I used the `LaTeX2HTML` converter. I was now able to write the manual in plain LaTeX (unadorned with the special commands that latexinfo required), and `LaTeX2HTML` would turn it into a set of HTML files.

But I soon found that I had a hard time making `LaTeX2HTML` generate the kind of HTML that I wanted. This was no flaw with `LaTeX2HTML`, but with the general approach of converting from LaTeX. In my eyes, conversion is not a solution to HTML authoring. A well written HTML document must differ from a printed copy in a number of rather subtle ways. I doubt that these differences can be recognized mechanically, and I believe that converted LaTeX can never be as readable as a document written in HTML.

This is most prominent in the formulation of cross references in a document. A LaTeX converter can turn the reference into a hyperlink, but it will have to keep the text the same. If we wrote "More details can be found in the classical analysis by Harakiri [8]", then the converter may turn "[8]" into a hyperlink to the bibliography in the HTML document. In handwritten HTML, however, we would probably leave out the "[8]" altogether, and make the *name* "Harakiri" a hyperlink.

The same holds for references to sections and pages. The Ipe manual says "This parameter can be set in the configuration panel (Section 11.1)". A converted document would have the "11.1" as a hyperlink. Much nicer HTML is to write "This parameter can be set in the configuration panel", with "configuration panel" a hyperlink to the section that describes it. If the printed copy reads "We will study this more closely on page 42," then a converter must turn the "42" into a symbol that is a hyperlink to the text that appears on page 42. What we would really like to write is "We will study this more closely later," with "later" a hyperlink — after all, it makes no sense to even allude to page numbers in an HTML document.

The Ipe manual also says "Such a file is at the same time a legal Encapsulated Postscript file and a legal LaTeX file — see Section 13." In the HTML copy

[1] Ipe is an attempt to fully integrate LaTeX text with PostScript drawing information. Ipe stores files in a format that is at the same time a legal PostScript and a legal LaTeX file, and the drawing editor runs LaTeX in the background to determine the size of text objects.

[2] That's where the `alt.religion.emacs.haters` pun in the Hyperlatex manual comes from.

the "Such a file" is a hyperlink to Section 13, and there's no need for the "—see Section 13" anymore.

There are also differences between LaTeX copy and HTML copy that have to do with the fact that HTML is still a somewhat enhanced text format. Many LaTeX concepts are hard to represent in HTML.

For instance, how do you present a mathematical expression like x_i or $a^2 + b^2 = c^2$ in HTML? `LaTeX2HTML` converts these to little bitmaps. That is quite sophisticated, but is it the best representation? I don't think so. With current technology, bitmaps eat too much transmission time, and they only look good when the resolution of the browser is nearly the same as the resolution at which the bitmap has been created, which is not a realistic assumption.

Isn't there an easier way? If x_i is the ith element of an array, then I would write it as `x[i]` in HTML. If it's a variable in a program, I'd probably write `xi`. In another context, I might want to write *x_i*. To write Pythagoras' theorem, I might simply use `a^2 + b^2 = c^2`, or maybe `a*a + b*b = c*c`. To express "For any $\varepsilon > 0$ there is a $\delta > 0$ such that for $|x-x_0| < \delta$ we have $|f(x)-f(x_0)| < \varepsilon$" in HTML, I would write "For any *eps* > *0* there is a *delta* > *0* such that for |*x-x0*| < *delta* we have |*f(x)-f(x0)*| < *eps*."

Of course a converter could be told to translate ε to *eps*. But the best representation in HTML very often depends on the context, and is beyond the reach of any (non-human) converter.

So I ended up not using `LaTeX2HTML`; but `LaTeX2HTML` is a good general converter and I had and have no ambition to improve on that.[3]

Instead, I turned back to the lisp macros from the latexinfo package and changed them to generate HTML output instead of `info` files. Of course this was intended to be a hack, and never meant for wide use... How could I know that I would end up by having to write a short manual for Hyperlatex itself, and would even be invited to write a *TUGboat* article about it?

Although I still keep getting Email messages saying "Hey, your Hyperlatex converter is rubbish. It fell over immediately when I tried to convert this LaTeX file!", Hyperlatex was not intended to be a general LaTeX-to-HTML converter — for the reasons explained above.

[3] And before anybody accuses me of being unfair — yes, all the differences described above can be achieved using the `texonly` and `htmlonly` environments of `LaTeX2HTML`, and its macros for making cross references. But I soon found this too cumbersome.

The idea of Hyperlatex is to make it possible to write a document that will look like a flawless LaTeX document when printed and like a handwritten HTML document when viewed with an HTML browser. In this it completely follows the philosophy of latexinfo (and texinfo). Like latexinfo, it defines its own input format — the *Hyperlatex markup language* — and provides two converters to turn a document written in Hyperlatex markup into a DVI file or a set of HTML documents. If you have written a document `sample.tex` in Hyperlatex markup,[4] you simply run LaTeX on your file to generate a DVI file, which you can print as usual.

On the other hand, you can type

```
hyperlatex sample.tex
```

to generate a set of HTML files, probably called `sample.html`, `sample_1.html`, `sample_2.html` and so on. (The command `hyperlatex` is a simple shell script that calls GNU Emacs in batch mode and runs the Emacs lisp macros that implement the conversion to HTML. It is also possible to call these macros directly from inside Emacs.)

Obviously, this approach has the disadvantage that you have to learn a "new" language to generate HTML files. However, the mental effort for this is quite limited. The Hyperlatex markup language is simply a well-defined subset of LaTeX that has been extended with commands to create hyperlinks, to control the conversion to HTML, and to add concepts of HTML such as horizontal rules and embedded images. Furthermore, you can use Hyperlatex perfectly well without knowing anything about HTML markup.

The fact that Hyperlatex defines only a restricted subset of LaTeX does not mean that you have to restrict yourself in what you can do in the printed copy. Hyperlatex provides many commands that allow you to include arbitrary LaTeX commands (including commands from any package that you would like to use) which will be processed to create your printed output, but which will be ignored in the HTML document. However, you do have to specify that *explicitly*. Whenever Hyperlatex encounters a LaTeX command outside its restricted subset, it will complain bitterly.

The rationale behind this is that when you are writing your document, you should keep both the printed document and the HTML output in mind. Whenever you want to use a LaTeX command with no defined HTML equivalent, you are

[4] Yes, I do use the extension `.tex` for my Hyperlatex files, sharing it with TeX and LaTeX. The Hyperlatex format is much more similar to LaTeX than LaTeX is to TeX, so this seems justified.

thus forced to specify this equivalent. For instance, if you have marked a logical separation between paragraphs with a LaTeX `\bigskip` command (not in Hyperlatex's restricted set of commands, since there is no HTML equivalent), then Hyperlatex will complain, since very probably you would also want to mark this separation in the HTML output. So you would have to write

```
\texonly{\bigskip}
\htmlrule
```

to imply that the separation will be a `\bigskip` in the printed version and a horizontal rule in the HTML-version. Even better, you could define a command `\separate` in the preamble and give it a different meaning in DVI and HTML output. If you believe that `\bigskip` should always be ignored in the HTML version, then you can state so in the preamble as follows.

```
\W\newcommand{\bigskip}{}
```

The `\W` command, introduced later, ensures this redefinition apples only to the HTML version. This philosophy implies that in general an existing LaTeX file will not make it through Hyperlatex. In many cases, however, it will be sufficient to go through the file once, adding the necessary markup that specifies how Hyperlatex should treat the unknown commands.

The `LaTeX2HTML` converter will convert any environment for which it does not have a built-in translation to HTML to a bitmap. This option exists in Hyperlatex as well, but again you have to explicitly ask for it by enclosing the unknown environment in a `GIF` environment.

Unlike `LaTeX2HTML`, Hyperlatex does not create a temporary LaTeX file with the `GIF` environments. In fact, the `GIF`-making is mostly implemented in TeX! The `hyperlatex.sty` package defines the `gif` environment as follows if the flag for `GIF`-making is set.

```
\def\gif{\setbox\@gifbox=\vbox\bgroup}
\def\endgif{\egroup\shipout
  \copy\@gifbox\unvbox\@gifbox}
```

This means that the contents of the `gif` environment is put in a box which is shipped out on a separate page of the DVI file without disturbing LaTeX too much. Later, a shell script extracts the extra pages from the DVI file using `dvips` and turns them into bitmaps using ghostscript. The shell script itself is created by the LaTeX run.

Hyperlatex implements most LaTeX commands that have a clear HTML analog, such as sectioning, font styles and sizes, displays and quotations, lists, accents (well, the ones defined in HTML), verbatim text, and there's even a weak implementation of `tabular`. I tried to be faithful to the spirit of LaTeX when adding new commands. latexinfo had some commands with non-LaTeX syntax, but those all had to go. I also changed the parser such that it much more closely mimics TeX's parsing. One of the basic rules is that the meaning of no LaTeX command has been changed.

Hyperlatex provides a number of different ways of treating parts of your document differently in LaTeX and HTML mode. The two simple commands `\texonly` and `\htmlonly` ignore their argument if in the wrong mode. The command `\texorhtml` takes two arguments of which only one is evaluated. The two environments `iftex` and `ifhtml` are convenient to ignore larger chunks of input in one mode. Finally, you can prefix a single line with `\T` or `\W`, so that you could write

```
We are now in
\T \LaTeX-mode.
\W Html-mode.
```

Hyperlinks are created with the commands `\link` and `\label`: `\link{`*anchor*`}{`*label*`}` typesets the text *anchor* and makes it an active hyperlink to the label *label* in the HTML document. To also create a reference in the printed document, you will need to use `\ref` or `\pageref`. This is facilitated by the optional argument of `\link`.

```
\link{anchor}[printed reference]{label}
```

The LaTeX output of this command will contain the anchor and the printed reference, while the HTML output will only show the anchor as a hyperlink to the position marked with the label. So you can write

```
This parameter can be set in the
\link{configuration panel}%
[~(Section~\ref{con-panel})]{con-panel}.
```

The starred version `\link*` suppresses the anchor in the printed version, so that we can write

```
We will see
\link*{later}[in Section~\ref{sl}]{sl}
how this is done.
```

It is common to cross-reference by using `\ref{`*label*`}` or `\pageref{`*label*`}` inside the optional argument, where *label* is the label set by the link command. In that case the reference can be abbreviated as `\Ref` or `\Pageref` (with capitals). These definitions are already active when the optional arguments are expanded, so we can write the example above as

```
This parameter can be set in the
\link{configuration panel}%
[~(Section~\Ref)]{con-panel}.
```

This even works when we need the `\Ref` command outside, but after the `\link` command, as for instance in

```
\link{Such a file}{\Ipe-file} is at
the same time ... a legal \LaTeX{}
file\texonly{---see Section~\Ref}.
```

References to external resources are made in exactly the same way, using the `\xlink` command, and references to the bibliography work using the same principle.

To facilitate typing short pieces of mathematics, Hyperlatex has a `\math` command whose argument is read in math mode in the printed version. In the HTML version, it is simply left untreated, so you can write `\math{x_i}` to get x_i in the HTML document. You could also use the optional argument, and writing `\math[\code{x[i]}]{x_i}` will give you `x[i]` in the HTML version. The Pythagorean example can by written as either `\math{a^2 + b^2 = c^2}`, or as `\math[a*a + b*b = c*c]{a^2 + b^2 = c^2}`.

The most important shortcomings in Hyperlatex 1.3 are: there are no footnotes; `\newcommand` and `\newenvironment` can only be used without arguments; and there is no support for the new `<fig>`, `<table>`, and `<math>` tags of HTML3 (which are already supported by some browsers). Also, there are a few idiosyncrasies that still stem from Hyperlatex's origin in latexinfo. The most important of these is the treatment of special characters. While LaTeX has ten of these, latexinfo has only three, namely `\`, `{`, and `}`. latexinfo is mainly used for software documentation, where one often has to use these characters without their special meaning, and since there is no math mode in `info` files, most of them are useless anyway. In the first version of Hyperlatex, I had only added the unbreakable space `~`, so there were four special characters. Since my main use was to write the Ipe manual, I found it convenient that I didn't even have to escape the `%` sign, for instance. However, it soon turned out that most other Hyperlatex users found this more confusing than convenient, and in Hyperlatex 1.1 the percent sign became a comment. And now that HTML3 declares a `<math>` tag, I find that it is time to return all 10 special characters to their special status, and this is going to be realized in Hyperlatex 1.4. That means that the `$` sign can now be used to enclose math mode material, that `&` can be used as a separator in formulas, and `#` for parameters for new commands. I hope that this will make it easier for new Hyperlatex users.

A problem is the growing number of HTML dialects. By users' request, I had added the font size-changing command when the Web browser Netscape became available. Some other Netscapeisms can be used using optional arguments (which simply generate raw HTML attributes). I also changed the `center` environment to use Netscape's `<center>` tag (it used to be the same as `quotation`). It's getting more messy all the time, and Hyperlatex 1.4, which I'm testing right now, will have a `\htmllevel` command to set the type of HTML that will be generated (HTML2, Netscape, or HTML3).

When HTML3 is selected, then Hyperlatex 1.4 will generate `<math>` tags for math mode material. It will also translate LaTeX's `figure` and `table` environment to `<fig>` tags, and will have a fuller implementation of `tabular` using `<table>`.

Hyperlatex 1.4 will also have footnotes, and commands and environments with arguments, and I hope that this will make Hyperlatex users even happier.

More information about Hyperlatex is available on the Web at `http://hobak.postech.ac.kr/~otfried/html/hyperlatex.html` or `http://www.cs.ruu.nl/~otfried/html/hyperlatex.html`

You may find Hyperlatex 1.4 already there by the time this article appears.

◇ Otfried Schwarzkopf
Dept. of Computer Science
Postech
San 31, Hyoja-Dong
Pohang 790-784, South Korea
Email: `otfried@vision.postech.ac.kr`