General Delivery

Opening Words

Michel Goossens,
President, TEX Users Group

Electronic Documents

Nowadays the media are buzzing with terms like the "global village", "information highways", the "web", "virtual money", "hypertext", and in our daily work, we are confronted more and more with the rapid developments of new techniques in the field of electronic publishing. As this issue of *TUGboat* will show, LATEX users are not left behind, and can continue to use their tool of preference for taking advantage of all these latest possibilities.

Today, the Internet connects more than two million computers and over twenty million people worldwide. For the first time humanity, has the capability of being informed of events at the moment when they arrive, but thanks to the Internet, you no longer have to undergo these events passively, like watching television, but you can go and look up information anywhere on the tens of thousands of information servers and form your own opinion. Moreover, the "Net" is an ideal medium facilitating research, offering a database of entertainment possibilities in-town, or during your next trip to a faraway place. In one word, you are now connected and, more importantly, you can set up your own information server for your colleagues, friends, or other people interested in your activities.

LATEX as a Hypertool

It is this latter point that is of interest when we want to find ways to optimize our investment in the re-use of existing documents marked up in LATEX for this new electronic hyperweb. As explained in the editorial by the guest editor of this issue, Malcolm Clark, several possibilities exist to transform LATEX input files, in a more-or-less automatic way, into HTML, the SGML-based language of the WWW, the popular and user-friendly interface to the Internet, or into PDF, an optimized form of PostScript, which allows hyperlinks.

I thank the various authors, and the developers of these software packages for their interesting work that will allow each of us to fully exploit the advantages of the generic approach of LATEX, with those of the hypertext language of the Web and the typographic quality of PDF.

I, of course, also want to acknowledge the work of Malcolm Clark, who is well-known in the TEX world, not only because he is a former President of both UK-TUG, and TUG itself, but for his continuous struggle to make TEX better known as a tool of excellence for the typographic composition of documents. He has once more done an perfect job, and I am sure our readers will find this issue all the more pleasant and informative to read.

Living up to our Promises

As I promised at TUG'95 in St. Petersburg last summer, and restated in the two previous issues of *TUGboat*, we are now well underway to catch up the delay in publishing *TUGboat*. In the near future, you will have the pleasure of receiving and enjoying the TUG'95 Proceedings issue, guest-edited by Robin Fairbairns, whom I wholeheartedly congratulate upon being the newly elected President of UK-TUG.

Moreover, the *TUGboat* Production Team[1] is quite confident that *TUGboat* **16**, 4, the last issue of 1995, will be sent to the printer before Christmas, so that you will hopefully receive it in the first half of January.

Looking forward

I feel it is thus appropriate to have a brief look at the future, and, apart from the two "standard" and the Dubna TUG'96 Proceedings of *TUGboat* next year, we are also planning to continue to have a guest-edited theme issue. For 1996 we hope to be able to offer our readers an overview of how well TEX can be used to compose "beautiful books" in (almost) all languages of the world.

◇ Michel Goossens,
President, TEX Users Group
CERN, CN Division
CH-1211 Geneva 23
Switzerland
Email: `goossens@cern.ch`

Introduction

Malcolm Clark

Some of the stimulus for collecting the papers in this volume together comes from a happy coincidence: both GUTenberg and the UK TEX Users Group had meetings on this theme on the same day — January 19th, 1995. In the event, we have drawn only three

[1] Consisting of Barbara Beeton, Mimi Burbank, Robin Fairbairns, Sebastian Rahtz, Christina Thiele, and myself, with Malcolm Clark helping with this issue of *TUGboat*.

papers presented at these meetings and published in *Cahiers Gutenberg* [4] and *Baskerville* [1].

Portability of an electronic document often implies SGML, the Standard Generalized Markup Language. As Goossens and Saarela point out, SGML has been around for many years, and in fact traces its ancestry back to GML [3] (Generalized Markup Language), developed by IBM. SGML concentrates on document content, and has very little to say about appearance. We can contrast this with TEX which, in its rawer forms, can be obsessed with the details of appearance. LATEX provides a convenient compromise between content (or structure), and the form 'on the page'. As Doyle rightly notes, TEX (and LATEX) are rather unhealthily targeted towards paper. If we really are thinking about electronic documents and electronic delivery, we should also be thinking about electronic presentation. This is not to claim that paper is dead as a medium, but there are others which are appropriate and may be more convenient.

Another thread comes through hypertexts: it is a straightforward concept to develop the notions of tables of contents, citations, cross-references and so on, to techniques for navigating through a document, and then to ways of relating similar pieces of content. The notion of hypertext is hardly new: by 1945, Vannevar Bush laid down many of the principles which are still current. To many, the World Wide Web realised Bush's concepts. In its present form, the Web depends upon SGML. The particular instantiation it uses, HTML, owes something to concepts present in 'richtext', which seems remarkably similar to `texinfo`, a TEX-variant. But nothing which appears in this volume is fundamentally Web-specific: the documents could be also available from CD-ROM.

The papers of Doyle and Schwarzkopf demonstrate that LATEX can be brought into the hypertext fold, completely independent of HTML, and yet operating happily over the Internet. The Los Alamos E-print archives are especially interesting, since they were developed for the same sort of community for whom the World Wide Web was created.

If you must ensure that your document 'looks right' when displayed on a screen, then one of the most appropriate vehicles is Adobe's Acrobat, which crops up notably in the papers by Haralambous & Rahtz and Granger. Acrobat (a sort of *hyper*-PostScript) parallels the Web, but fits into the model remarkably well. It enables any browser to see an electronic document which has the same 'look and feel' whatever device is used to view it. By making some dynamic font-making capabilities available, Acrobat can emulate the size and shape of most fonts, ensuring that the perceived document retains the geometry it started with. True, it remains an approximation, but it is a very close one. Added to this, Acrobat has hypertext navigation tools built in. A LATEX document can be created which can be converted to Acrobat format, complete with highly portable, non-proprietary, fonts, and the hyperlinks.

Exploring another strategy, Goossens & Saarela examine how LATEX documents may be converted to HTML and made available on the Web with the hyperlinks they require to exploit the inherent potential.

And to square the circle, Flynn notes how TEX and LATEX may be used to translate HTML into print.

It says much for the original concept of TEX and LATEX that they appear to fit so well into this brave new world of distributed, linked, electronic documents.

References

[1] *Baskerville.* (Sebastian Rahtz, editor), vol. 5, no. 2, March 1995.

[2] Vannevar Bush. As we may think. *Atlantic Monthly*, vol. 176, pp. 101–108, 1945.

[3] C. F. Goldfarb, E. J. Mosher and T. I. Peterson. An online system for integrated text processing. *Proceedings of the American Society for Information Science*, vol. 7, pp. 147–150, 1970.

[4] *Cahiers Gutenberg.* Diffusion des documents électroniques: de LATEX à WWW, HTML et Acrobat, no. 19, January 1995.

◇ Malcolm Clark
Computing Services
University of Warwick
Coventry CV4 7AL
U.K.
Email: `m.clark@warwick.ac.uk`

A Practical Introduction to SGML

Michel Goossens and Janne Saarela

Abstract

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

1 Why SGML?

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

Once data have been stored in electronic form one can derive multiple products from a single source document. For instance, an address list can be turned into a directory on paper, but it can also be put on CD-ROM, as a database allowing interactive or e-mail access on the Internet or to print a series of labels. Using a set of law texts or a series of articles on history marked up in SGML, one can first publish a textbook containing complete law texts, or a historic encyclopedia, and then provide regular updates or extract a series of articles on a given subject; one can also offer a service which may be consulted or interrogated on Internet, via gopher or WWW, or develop a hypertext system on CD-ROM.

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

To recapitulate, the main aims of generic markup (in SGML) are the following:

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

1.1 The origins of SGML

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

A big step forward was the publication by ISO (the International Standards Organization, with its headquarters in Geneva, Switzerland) in October 1986 of SGML as Standard ISO 8879 (ISO, 1986). Because SGML had been officially endorsed by ISO, the Standard was quickly adopted by various national or international organizations and by the large software developers. One can thus be fairly confident that SGML is here to stay and that its role in electronic publishing will continue to grow.

1.2 Who uses SGML?

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

First among the application areas we will consider in which SGML is at present actively used is the work of the American Association of Publishers (AAP). The AAP (see AAP (1989) to AAP (1989c)) selected three types of documents in the field of publishing: a book, a series publication, and an article. For each of these a *document type definition* (DTD, see below, especially Section 4) has been developed. Together, the AAP and the EPS (European Physical Society) have proposed a standard method for marking up scientific documents (especially tables and mathematical documents). This work forms the basis of ISO/IEC 12083.

Another application actively developed during the last few years is the CALS (*Computer-aided*

Acquisition and Logistic Support) initiative of the US Department of Defense (DoD). This initiative aims at the replacement of paper documents by electronic media for the documentation of all arms systems. The DoD decided that all documentation must be marked up in SGML, thus also making (the frequent) revisions a lot easier.

A few other examples of the use of SGML are:[1]

- the Publications Office of the European Communities (FORMEX);
- the Association of German editors (Börsenverein des Deutschen Buchhandels);
- the British Library with "SGML: Guidelines for editors and publishers" and "SGML: Guidelines for authors";
- in France, the *Syndicat national de l'édition* and the *Cercle de la librairie*, two associations of French publishers, have defined an application for the French editing world (Vignaud, 1990);
- the ISO Publishing Department;
- the British Patents Office (HMSO);
- Oxford University Press;
- the Text Encoding Initiative (classic texts and comments);
- the technical documentation of many major computer manufacturers or scientific publishers, for instance the `Doc-Book` or other dedicated DTDs used by IBM, HP, OSF, O'Reilly, etc.
- many text processing and database applications have SGML input/output modules (filters), for example, Frame, Interleaf, Microsoft, Oracle, Wordperfect;
- McGraw-Hill (Encyclopedia of Science and Technology);
- the electronics industry (Pinacle), the aerospace industry and the airlines (Boeing, Airbus, Rolls Royce, Lufthansa, etc.), the pharmaceutical industry;
- press agencies;
- text editors and tools with direct SGML interfaces, such as ArborText, EBT (Electronic Book Technologies), Exoterica, Grif, Softquad;
- and, of course, HTML and WWW!

[1] See also the "SGML Web Page" at the URL `http://www.sil.org/sgml/sgml.html` for more information on who uses SGML and why.

2 SGML Basic Principles

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

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

2.1 Different types of markup

The "text processing" systems that have found their way into almost every PC or workstation nowadays are mostly of the WYSIWYG type, i.e.,, one specifically chooses the "presentation" or "formatting" characteristics of the various textual elements. They can be compared to an earlier generation of formatting languages, where specific codes were mixed with the (printable) text of the document to control the typesetting on the micro level. For example, line and page breaks, explicit horizontal or vertical alignments or skips were frequently used to compose the various pages. In general these control characters were extremely application-specific, and it was difficult to treat sources marked up in one of these systems with one of the others. On the other hand, this type of markup does a very good job of defining the specific physical representation of a document, and for certain kinds of documents it might be more convenient for obtaining a given layout, in allowing precise control of line and page breaks. This approach makes viewing and printing documents particularly easy, but re-using the source for other purposes can be difficult, even impossible.

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

Figure 1 shows a few examples of marking up the same text. One clearly sees the difference between *specific* markup, where precise instructions are given to the text formatter for controlling the layout (for example, the commands `\vskip` or `.sp`), *Specific markup*

TEX

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

Script

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

Generic or logical markup

LATEX

```
\chapter{Title of Chapter}
\par
```

HTML (SGML)

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

Figure 1: Different kinds of markup

and *generic* markup, where only the logical function (chapter or beginning of paragraph) is specified.

2.2 Generalized logical markup

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

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

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

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

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

Figure 2: Two instances of the same document class "article"

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

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

2.3 A few words about the DTD

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

To fulfill all these aims a DTD defines:

- the *name* of the elements that can be used;
- the *contents* of each element (Section 4.2.1);
- *how often* and in what order each element can occur (Section 4.2.3);
- if the begin or end tag can be *omitted* (Section 4.2.2);
- possible *attributes* and their default values (Section 4.3);
- the name of the *entities* that can be used (Section 4.4).

3 Transmitting the Information Relative to a Document

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

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

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

The basic character set is ASCII, as defined by International Standard ISO/IEC 646. One can change the character set by changing this declaration at the beginning of the parsing of the document, when the SGML declaration associated to the DTD is read in (see Appendix C on page 131).

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

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

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

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

4 The Structure of a DTD

To better understand how SGML works we propose to examine a real example of a modern SGML application, namely HTML level 2, which corresponds to the functionality offered by popular HTML viewing programs, such as Mosaic, Netscape or Lynx. The complete DTD of HTML2 is shown in Appendix B starting on page 124. To make it easier to identify the various parts of the DTD the lines have been numbered.

Before starting to parse a DTD the SGML declaration is read in by the parser. For HTML this declaration is shown in Appendix C on page 131. It defines the character set, special characters and option settings used in the DTD and allowed in the document instance. For instance, in the area of markup minimization, the parameter `OMITTAG` (Line 66) has the value `YES`, which allows tag minimization, i.e.,, under certain circumstances (specified in the DTD) tags can be omitted, as explained in Section 4.2.2. If, on the other hand, the value is specified as `NO` then tag minimization is disallowed altogether.

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

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

4.1 Comments

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

An SGML comment has the form:

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

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

4.2 The elements

4.2.1 An element declaration

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

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

For instance Lines 614 and 616 are equivalent to the declaration:[3]

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

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

4.2.2 Omit tags

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

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

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

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

[4] The meaning of the symbols `|` and `+` is explained in Section 4.2.3, see especially Table 1; the definition of the parameter entity `%flow` can be found on Line 313, see also Section 4.2.3.

symbol	*description*
`,`	all must appear and in the order indicated (ordered "and")
`&`	all must appear but any order is allowed (unordered "and")
`|`	one and only one can appear (exclusive "or")
`+`	element must appear once or more
`?`	optional element (0 or one)
`*`	element can appear zero times or more

Table 1: Order and choice operators

4.2.3 The contents model

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

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

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

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

An element with multiple members that can appear in any order is defined on Lines 548–553. These lines essentially stipulate that an HTML head can contain, in any order, a title (`TITLE`), zero or one `<ISINDEX>`, `<BASE>`, and `<NEXTID>` tags, and zero or more `<META>` and `<LINK>`:

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

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

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

`PCDATA` *parsed character data.*
The characters are supposed to have been treated by the parser and can thus no longer contain entity references or tags.

For instance, on Line 557 an HTML title is defined as:

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

`RCDATA`	*replaceable character data.* The parser can expect to find only characters or entity references, i.e.,, (begin and end) tags are forbidden.
`CDATA`	*character data.* No further processing is needed by the SGML parser (nevertheless, the data might be processed by another program, for instance PostScript). A telephone number in a letterhead could be declared thus: `<!ELEMENT TEL CDATA>`
`ANY`	The element can contain data of type `PCDATA` or *any* other element defined in the DTD.
`EMPTY`	The element has an *empty content.* It can, however, be qualified by possible attributes (see Section 4.3). An example of this is the `<IMG>` tag and its attributes as defined on Lines 233–240.

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

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

This states that the `<FORM>` element can contain everything specified by the `%body.content` parameter entity (Lines 430, 267, 146, and 309–311). Moreover, all these elements can contain, *at any level* the tags `<INPUT>`, `<SELECT>`, or `<TEXTAREA>`. On the other hand, forms are not recursive, since the `<FORM>` tag cannot itself contain (`-(FORM)`).

4.3 Attributes

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

An attribute declaration consists of:

- the name of the element(s) that it refers to;
- the name of the attribute;

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

Table 2: Keywords for attribute types

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

An attribute declaration thus takes the following form:

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

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

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

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

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

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

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

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

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

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

Table 3: Keywords for attribute default values

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

4.4 Entities

Entities can be used for the following purposes:

- The definitions of abbreviated notations to ease repetitive text strings (general entities); for example,
  ```
  <!ENTITY TUG "\TeX{} Users Group">
  ```
- The definition of notations to input special characters, accents or symbols (general character entities). An example of character entities can be found on Lines 102–105;
  ```
  <!ENTITY amp CDATA "&#38;"
               -- "&" (ampersand) -->
  ```
  ISO has defined several standard character entity sets, for instance, for national characters (see Appendix E on page 134), graphical symbols, mathematics, etc.
- The inclusion of external files (external entities).
- The definition of variables in a DTD (parameter entities).

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

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

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

```
&entity_name;
```

For example, if one wants to use the entity "`TUG`" defined above, one should type in the document source the string of characters `&TUG;` and the parser replaces this by the string "TEX Users Group".

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

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

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

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

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

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

4.5 Other DTDs

In order to get a better idea of what DTDs for more complex documents look like, we shall briefly discuss HTML3, `Doc-Book` and ISO/IEC 12083.

4.5.1 HTML3

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

[5] See URL `http://www.hpl.hp.co.uk/people/dsr/html/CoverPage.html`.

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

Mathematics support for equations and formulae in HTML3 mainly uses TEX's box paradigm. The implementation uses a simple markup scheme, that is still powerful enough to cope with the range of mathematics created in common word processing packages. Filters from TEX and other word processing systems will allow one to easily convert existing sources into HTML3.

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

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

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

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

To illustrate the use of this DTD one can look at the table and mathematics parts of the HTML3 DTD (see Appendix F on page 135) and at the markup examples and generated output (Figures 3 and 4).

4.5.2 DocBook

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

[6] See URL `ftp://ftp.ora.com/pub/davenport/docbook/fullguide.sgm`.

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

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

4.5.3 The AAP effort and ISO/IEC 12083

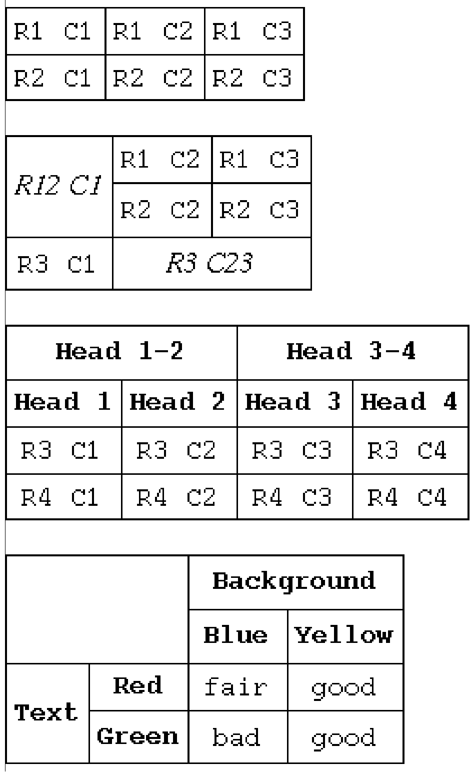
The American Association of Publishers (AAP) has been working since the publication of the SGML Standard in 1986 on promoting SGML as an electronic standard for manuscript preparation. This document, developed over several years as the "AAP Standard," was later promoted to by the Electronic Publishing Special Interest Group (EPSIG) and the AAP as "the Electronic Manuscript Standard," and is now a NISO (National Information Standards Organization) publication. The AAP/EPSIG application is SGML-conforming, and provides a suggested tag set for authors and publishers. It defines the format syntax of the application of SGML publication of books and journals. The Standard achieves two goals. First, it establishes an agreed way to identify and tag parts of an electronic manuscript so that computers can distinguish between these parts. Second, it provides a logical way to represent special characters, symbols, and tabular material, using only the ASCII character set found on a standard American keyboard.

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

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

[7] They can be found at the URL `http://www.sil.org/sgml/gen-apps.html\#iso12083DTDs`.

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

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

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



R1 C1	R1 C2	R1 C3
R2 C1	R2 C2	R2 C3

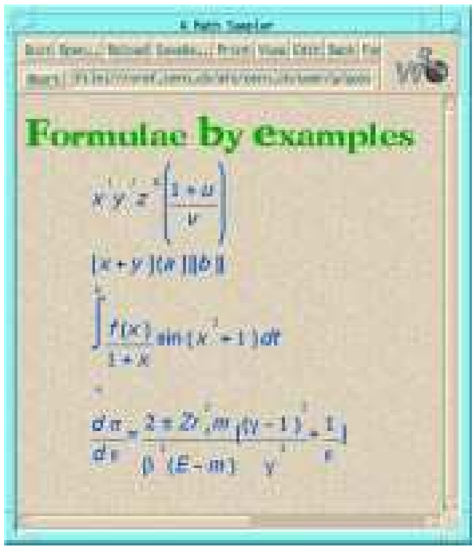
R12 C1	R1 C2	R1 C3
	R2 C2	R2 C3
R3 C1	*R3 C23*	

Head 1-2		**Head 3-4**	
Head 1	**Head 2**	**Head 3**	**Head 4**
R3 C1	R3 C2	R3 C3	R3 C4
R4 C1	R4 C2	R4 C3	R4 C4

		Background	
		Blue	**Yellow**
Text	**Red**	fair	good
	Green	bad	good

Figure 3: HTML3 example of tables (source and result with the Mosaic browser)

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



Formulae by examples

$$x^I y^J z^K \left(\frac{1+u}{v}\right)$$

$$[x+y](a]\|b\|$$

$$\int_a^b \frac{f(x)}{1+x} \sin(x^2+1)\,dt$$

$$\frac{d\sigma}{d\epsilon} = \frac{2\pi Z r_0^2 m}{\beta^2(E-m)}\left[\frac{(\gamma-1)^2}{\gamma^2}+\frac{1}{\epsilon}\right]$$

Figure 4: HTML3 example of simple mathematics (source and result with the arena browser)

The ISO/IEC 12083 table model

The ISO 12083 table model consists of the following elements (see Figure 5 for the relevant part of the DTD):

`<table>`	the table element;
`<np>`	number;
`<title>`	title;
`<tbody>`	table body;
`<head>`	head;
`<tsubhead>`	table subhead;
`<row>`	row;
`<tstub>`	table stub;
`<cell>`	cell.

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

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

Only the simple table model discussed above is part of the basic ISO/IEC 12083 DTD as distributed. There also exists a complex table model (AAP, 1989b) that allows the user to treat more complex tabular material.

The ISO/IEC 12083 mathematics model

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

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

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

superiors, inferiors
`<sup>`, `<inf>`;

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

fences, boxes, overlines and underlines
`<mark>`, `<fence>`, `<post>`, `<box>`, `<overline>`, `<undrline>`;

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

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

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

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

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

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

The complete DTD is shown in Appendix G on page 139, which shows the file `math.dtd` that is part of the ISO/IEC 12083 DTD set.

5 SGML Editors

Several solutions exist to enter SGML or HTML markup into a document, but an editor that is SGML-aware is probably the best solution. Several (mostly commercial) products exist (see Karney (1995a), Karney (1995b), and Ores (1995)), but in the remaining part of this section we shall have a look at a public domain solution based on the Emacs editor with the `psgml` application and on the Grif-based Symposia editor.

5.1 Emacs and `psgml`

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

Figure 6 shows the first HTML test example, to be discussed later (see example `test1.html` in Section 6.2.1). Both the `psgml` mode and the `nsgmls` program, discussed below, use a catalog file whose structure is defined by the SGML Open consortium to locate the SGML declarations and DTDs (see Appendix D on page 133). Thanks to the name of the DTD declared on the `<!DOCTYPE>` declaration and that catalog file, `psgml` loads the

[8] The `psgml` home page is at the URL `http://www.lysator.liu.se/projects/about_psgml.html`.

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

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

Figure 5: Part of the ISO 12083 DTD relating to simple tables

HTML2 DTD into memory and can then handle the HTML source file. In the Figure, all the elements that can occur at the position of the pointer are shown. Figure 7 shows the more important key combinations for quickly calling some functions. For instance, the sequence `C-c C-t` (`sgml-list-valid-tags`) was used to obtain the list in the lower part of Figure 6. As a last technical (but important) detail, in order to function properly, two variables should be defined in the `psgml` initialization file `psgml.el`, namely `sgml-system-path`, a list of directories used to look for system identifiers, and `sgml-public-map`, a mapping from public identifiers to file names.[9]

5.2 Symposia

At the Third International World Wide Web Conference "Technology, Tools and Applications",[10] which took place in Darmstadt, Germany, from 10–13 April 1995, Vincent Quint and collaborators discussed their authoring environment for SGML texts in general, and HTML on WWW in particular.[11] Their approach is based on the Grif editor, which can work with any DTD. They announced that a version with the HTML3 DTD will be made available freely under the name of Symposia. Grif (and Symposia) allow the user to enter text in a WYSIWYG way, but entered elements are validated against the DTD. An example is given in Figure 8, which shows us to be in insert mode in the first column on the first row of the table, where we input the word "text", whilst Figure 9 shows the generated SGML(HTML) source, hidden from the user, but available for any kind of treatment that one would like to do on the document.

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

[10] An overview of the papers is at the URL `http://www.igd.fhg.de/www/www95/papers/`.

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

6 SGML Utilities

As SGML is now actively used in many applications in the field of document production (see Section 1.2 and Karney (1995b)) several commercial and publicly available solutions are now available to increase the productivity, user-friendliness, and ease of using SGML systems. This section reviews a few of the more interesting publicly available tools.

6.1 Validating an SGML document with `nsgmls`

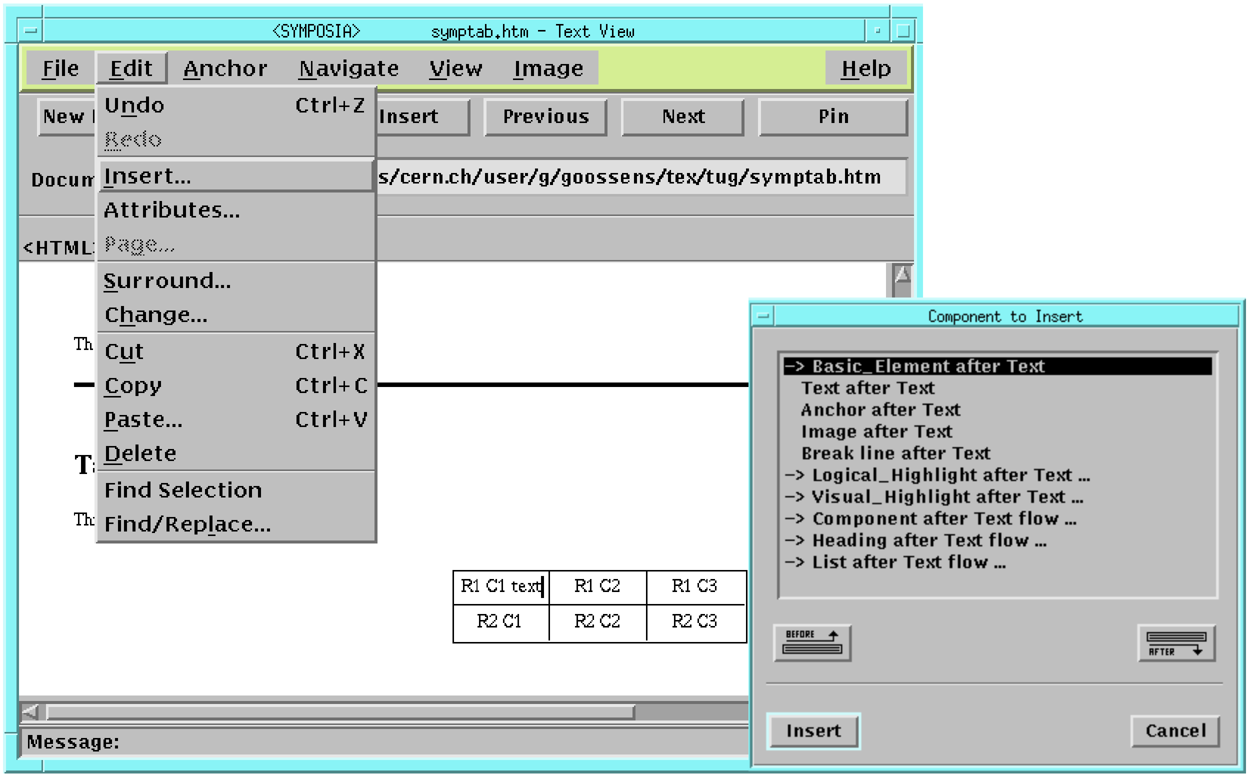
It is often important and useful to be able to validate an SGML (and hence HTML) document. This can, for instance, be achieved with the publicly available SGML parser `nsgmls`, which is part of SP,[12] a system developed by James Clark (`jjc@jclark.com`) and a successor to his older `sgmls`,[13] or by `arcsgml`, written by Charles Goldfarb (Goldfarb, considered by many to be the father of SGML, is also the author of "The SGML Handbook" (Goldfarb, 1990) describing the SGML Standard in great detail, a reference work that every serious SGML user should possess).

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

```
nsgmls [ -deglprsuvx ] [ -alinktype ]
       [ -ffile ] [ -iname ] [ -mfile ]
       [ -tfile ] [ -wwarning_type ]
       [ filename... ]
```

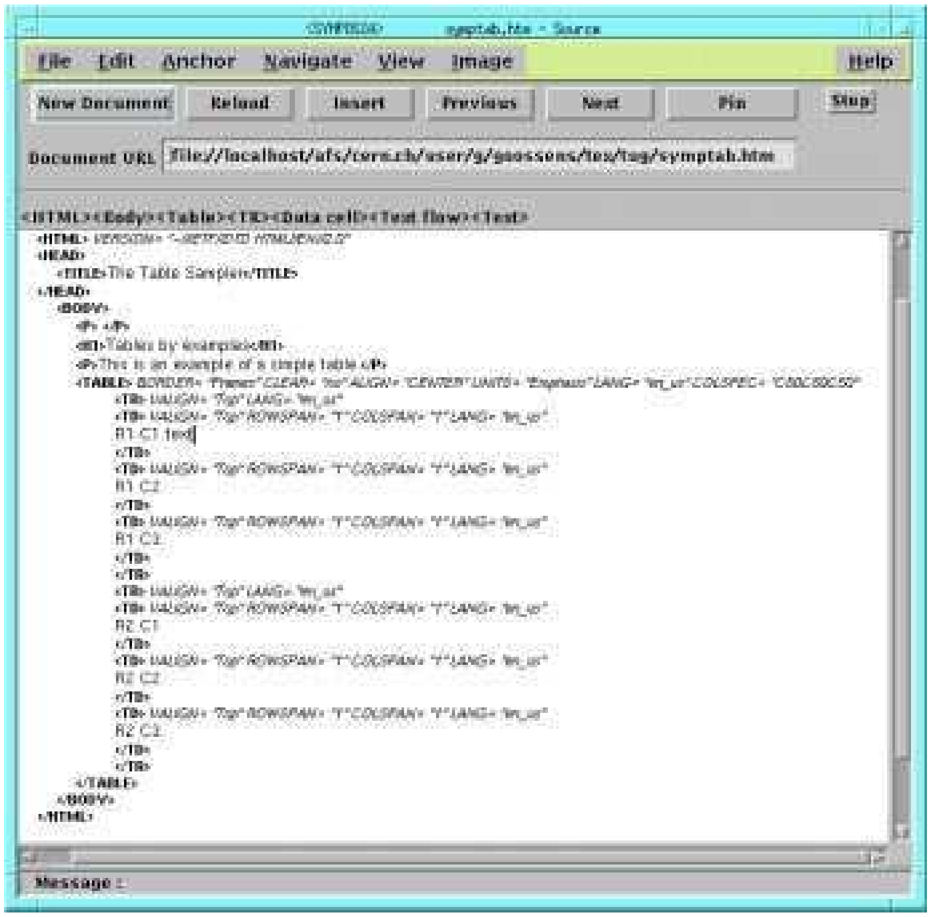
[12] SP is available at the URL `http://www.jclark.com/sp.html`. For more information about other publicly available SGML software, have a look at the the public SGML software list at the URL `http://www.sil.org/sgml/publicSW.html`. More generally, on the SGML Web Page at `http://www.sil.org/sgml/sgml.html`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

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



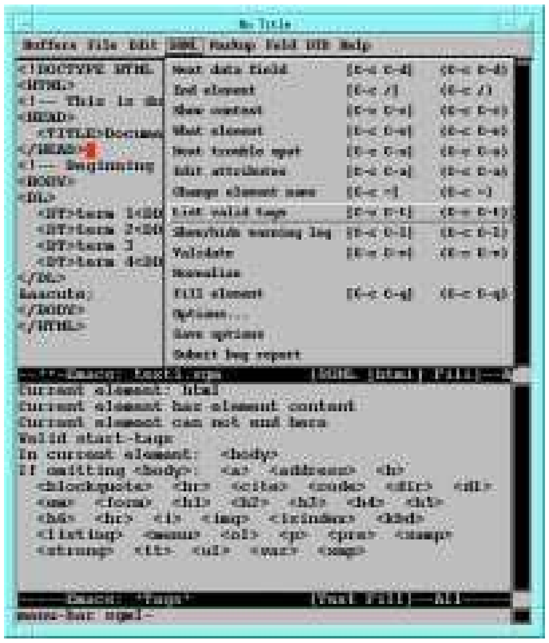
R1 C1 text	R1 C2	R1 C3
R2 C1	R2 C2	R2 C3

Figure 8: Inserting text in an SGML document with Symposia



```
<HTML><Body><Table><TR><Data cell><Text flow><Text>
<HTML> VERSION= "-//IETF//DTD HTML//EN//2.0"
<HEAD>
   <TITLE>The Table Sampler</TITLE>
</HEAD>
   <BODY>
      <P> </P>
      <H1>Tables by example</H1>
      <P>This is an example of a simple table.</P>
      <TABLE> BORDER= "Frame" CLEAR= "no" ALIGN= "CENTER" UNITS= "Emphasis" LANG= "en_us" COLSPEC= "C50C50C50"
            <TR> VALIGN= "Top" LANG= "en_us"
            <TD> VALIGN= "Top" ROWSPAN= "1" COLSPAN= "1" LANG= "en_us"
            R1 C1 text
            </TD>
            <TD> VALIGN= "Top" ROWSPAN= "1" COLSPAN= "1" LANG= "en_us"
            R1 C2
            </TD>
            <TD> VALIGN= "Top" ROWSPAN= "1" COLSPAN= "1" LANG= "en_us"
            R1 C3
            </TD>
            </TR>
            <TR> VALIGN= "Top" LANG= "en_us"
            <TD> VALIGN= "Top" ROWSPAN= "1" COLSPAN= "1" LANG= "en_us"
            R2 C1
            </TD>
            <TD> VALIGN= "Top" ROWSPAN= "1" COLSPAN= "1" LANG= "en_us"
            R2 C2
            </TD>
            <TD> VALIGN= "Top" ROWSPAN= "1" COLSPAN= "1" LANG= "en_us"
            R2 C3
            </TD>
            </TR>
      </TABLE>
   </BODY>
</HTML>
```

Figure 9: SGML source of the document shown in Figure 8


No Title
Buffers File Edit SGML Markup Fold DTD Help
Next data field [C-c C-d]
End element [C-c /]
Show context [C-c C-c]
What element [C-c C-w]
Next trouble spot [C-c C-o]
Edit attributes [C-c C-a]
Change element name [C-c =]
List valid tags [C-c C-t]
Show/hide warning log [C-c C-l]
Validate [C-c C-v]
Normalize
Fill element [C-c C-q]
Options...
Save options
Submit bug report

```
<!DOCTYPE HTML
<HTML>
<!-- This is th
<HEAD>
  <TITLE>Docume
</HEAD>
<!-- Beginning
<BODY>
<DL>
  <DT>term 1<DD
  <DT>term 2<DD
  <DT>term 3
  <DT>term 4<DD
</DL>
&aacute;
</BODY>
</HTML>
```

```
Current element: html
Current element has element content
Current element can not end here
Valid start-tags
In current element:  <body>
If omitting <body>:  <a>  <address>  <b>
  <blockquote>  <br>  <cite>  <code>  <dir>  <dl>
  <em>  <form>  <h1>  <h2>  <h3>  <h4>  <h5>
  <h6>  <hr>  <i>  <img>  <isindex>  <kbd>
  <listing>  <menu>  <ol>  <p>  <pre>  <samp>
  <strong>  <tt>  <ul>  <var>  <xmp>
```

Figure 6: Emacs in `psgml` mode

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

Figure 7: Emacs key-bindings with `psgml`

`nsgmls` needs at least four files to run:

- the catalog file, which describes how the SGML file's `<!DOCTYPE>` declaration is mapped to a filename (see below);
- the SGML declaration, defining the character set used by subsequent files, and the sizes of various internal limits, such as the permitted length of identifiers, as well as what features of SGML are used, such as tag minimization (see the start of Section 4 on page 106 and Appendix C on page 131);
- the DTD for the document type;
- an SGML or HTML document instance.

6.2 The `<!DOCTYPE>` declaration

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

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

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

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

The first string following the keyword `PUBLIC` is called a "public identifier", a name which is intended to be meaningful across systems and different user environments. Formally a public identifier is composed of several fields, separated by a double solidus, "`//`". The first part is an "owner identifier" (the first and third entries have a hyphen, `-`, meaning that these identifiers were not formally registered, and the organization who created the file was the IETF (the Internet Engineering Task Force); the second entry carries an ISO owner identifier. The second part of the public identifier (following the double solidus), is called the "text identifier".

The first word indicates the "public text class" (for example, DTD and ENTITIES), and is followed by the "public text description" (HTML, Latin 1, etc.), then, optionally, after another double solidus one finds the "public text language", a code from ISO Standard 639 (ISO (1988) — EN, for English in our case), and this can be followed by a "display version", if needed.

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

6.2.1 HTML examples

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

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

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

```
> nsgmls -m catalog sgml.decl test1.html
#SDA
AVERSION CDATA -//IETF//DTD HTML 2.0//EN
ASDAFORM CDATA Book
(HTML
(HEAD
ASDAFORM CDATA Ti
(TITLE
-Document test1.html
)TITLE
)HEAD
(BODY
ACOMPACT IMPLIED
ASDAFORM CDATA List
ASDAPREF CDATA Definition List:
(DL
ASDAFORM CDATA Term
(DT
-term 1
)DT
ASDAFORM CDATA LItem
(DD
-data 1\n
)DD
ASDAFORM CDATA Term
(DT
-term 2
)DT
ASDAFORM CDATA LItem
(DD
-data 2\n
)DD
ASDAFORM CDATA Term
(DT
-term 3\n
)DT
ASDAFORM CDATA Term
(DT
-term 4
)DT
ASDAFORM CDATA LItem
(DD
-data 4
)DD
ASDAFORM CDATA LItem
(DD
-data 4 bis
)DD
)DL
-\n\|[aacute]\|
)BODY
)HTML
C
```

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

`\\`	a \;
`\n`	a *record end*;
`|`	brackets internal SDATA entities;
`\nnn`	character whose octal code is nnn;
`(gi`	start of element whose generic identifier is gi, attributes for this element are specified with A commands;
`)gi`	end of element whose generic identifier is gi;
`-data`	data;
`&name`	reference to external data entity name;
`Aname val`	next element has an attribute name with specifier and value val (see Tables 2 and 3)

`#text`	application information (can only occur once);
`C`	signals that the document was a conforming document. It will always be the last command output.

For incorrect documents `nsgmls` shows an error:

```
<!DOCTYPE html PUBLIC
     "-//IETF//DTD HTML//EN">
<HTML>
<BODY>
  <P>text inside a paragraph
</BODY>
</HTML>
```

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

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

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

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

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

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

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

6.3 Prettyprinting

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

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

where the more interesting options are:

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

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

```
> html-pretty -i2 -n test1.html
<!DOCTYPE html PUBLIC
     "-//IETF//DTD HTML//EN">
<HTML>
  <!-- This is document doc1.sgm -->
  <HEAD>
    <TITLE>
      Document test HTML
    </TITLE>
  </HEAD>
  <!-- Beginning of body of document -->
  <BODY>
    <DL>
      <DT>
        term 1
      </DT>
      <DD>
        data 1
      </DD>
      <DT>
        term 2
      </DT>
      <DD>
        data 2
      </DD>
```

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

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

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

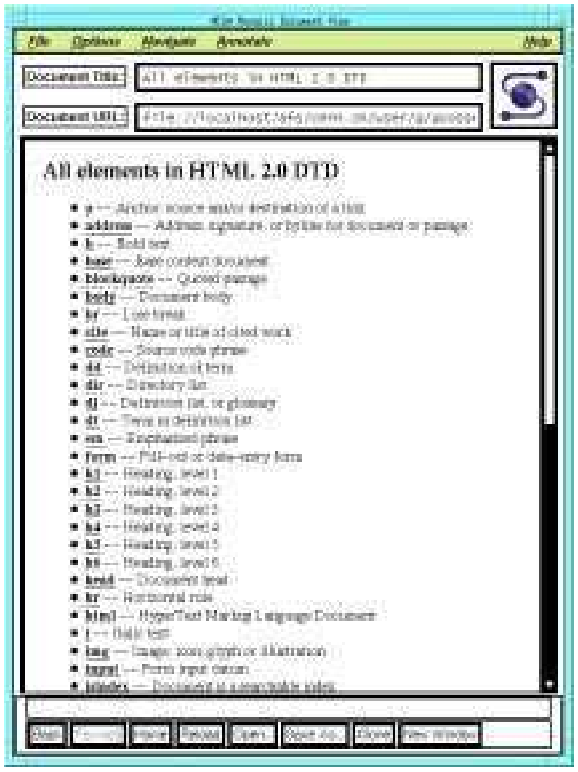
6.4 SGML document analysis tools

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

`dtd2html`	produces an HTML document starting from an SGML DTD that permits an easy hypertext navigation through the given DTD;
`dtddiff`	compares two DTDs and shows possible differences;
`dtdtree`	shows visually the hierarchical tree structure characterizing the relations between the various elements of a DTD;
`stripsgml`	strips a text from its SGML markup, and attempts to translate entity references by standard ASCII characters.

Let us first look at the `dtdtree` utility. When treating the HTML2 DTD, one obtains a visual representation that is very useful for understanding the relations that exist between the various HTML elements. For each element one explicitly sees the elements it can contain. Three points "`...`" indicate that the contents of the element has been shown previously. Lines containing entries between brackets signal a list of elements that can be included in — `(I)` and `(Ia)` — or are excluded from — `(X)` and `(Xa)` — the content model of the element. Figure 10 shows in four columns the (condensed) output generated by the `dtdtree` program when treating the HTML2 DTD. For more clarity most of the repeated blocks have been eliminated and replaced by the string `*\vbar**\vbar**\vbar` at the beginning of a line and a few lines have been cut to make them fit (marked with `***` at the end of the line).

[15] This system can be found at the URL `ftp://ftp.uci.edu/pub/dtd2html`.



All elements in HTML 2.0 DTD

file://localhost/afs/cern.ch/user/g/goosse

All elements in HTML 2.0 DTD

- a — Anchor; source and/or destination of a link
- address — Address, signature, or byline for document or passage
- b — Bold text
- base — Base context document
- blockquote — Quoted passage
- body — Document body
- br — Line break
- cite — Name or title of cited work
- code — Source code phrase
- dd — Definition of term
- dir — Directory list
- dl — Definition list, or glossary
- dt — Term in definition list
- em — Emphasized phrase
- form — Fill-out or data-entry form
- h1 — Heading, level 1
- h2 — Heading, level 2
- h3 — Heading, level 3
- h4 — Heading, level 4
- h5 — Heading, level 5
- h6 — Heading, level 6
- head — Document head
- hr — Horizontal rule
- html — HyperText Markup Language Document
- i — Italic text
- img — Image; icon, glyph or illustration
- input — Form input datum
- isindex — Document is a searchable index

Figure 11: Hypertext description of the elements of a DTD (HTML2) as presented by the HTML browser `mosaic`

6.4.1 Documenting a DTD

To document a DTD (and hence a particular SGML language instance) one can use the `dtd2html` utility, which generates, starting from the DTD in question and a file describing all document elements, a hypertext representation (in HTML) of all SGML language elements present in the DTD. This representation makes it easier for users of an SGML-based documentation system to obtain the information relating to an element they need for marking up their document. For example, in the case of HTML2, Figure 11 shows the representation as viewed by the HTML browser `mosaic`.

```
HTML
 |
 |_body
 |  |
 |  |_#PCDATA
 |  |_a
 |  |  | (X): a
 |  |  |
 |  |  |_#PCDATA
 |  |  |_b ...
 |  |  |_br ...
 |  |  |_cite ...
 |  |  |_code ...
 |  |  |_em ...
 |  |  |_h1 ...
 |  |  |_h2 ...
 |  |  |_h3 ...
 |  |  |_h4 ...
 |  |  |_h5 ...
 |  |  |_h6 ...
 |  |  |_i ...
 |  |  |_img ...
 |  |  |_kbd ...
 |  |  |_samp ...
 |  |  |_strong ...
 |  |  |_tt ...
 |  |  |_var ...
 |  |
 |  |_address
 |  |  |
 |  |  |_#PCDATA
 |  |  |_a ...
 |  |  |_b ...
 |  |  |_br ...
 |  |  |_cite ...
 |  |  |_code ...
 |  |  |_em ...
 |  |  |_i ...
 |  |  |_img ...
 |  |  |_kbd ...
 |  |  |_p ...
 |  |  |_samp ...
 |  |  |_strong ...
 |  |  |_tt ...
 |  |  |_var ...
 |  |
 |  |_b
 |  |  |
*|**|**| Like address
 |  |
 |  |_blockquote
 |  |  |
 |  |  |_#PCDATA
 |  |  |_a ...
 |  |  |_address ...
 |  |  |_b ...
 |  |  |_blockquote ...
 |  |  |_br ...
 |  |  |_cite ...
 |  |  |_code ...
 |  |  |_dir ...
 |  |  |_dl ...
 |  |  |_em ...
 |  |  |_form ...
 |  |  |_h1 ...
 |  |  |_h2 ...
 |  |  |_h3 ...
 |  |  |_h4 ...
 |  |  |_h5 ...
 |  |  |_h6 ...
 |  |  |_hr ...
 |  |  |_i ...
 |  |  |_img ...
 |  |  |_isindex ...
 |  |  |_kbd ...
 |  |  |_listing ...
 |  |  |_menu ...
 |  |  |_ol ...
 |  |  |_p ...
 |  |  |_pre ...
 |  |  |_samp ...
 |  |  |_strong ...
 |  |  |_tt ...
 |  |  |_ul ...
 |  |  |_var ...
 |  |  |_xmp ...
 |  |
```

```
 |  |_br
 |  |  |
 |  |  |_EMPTY
 |  |
 |  |_cite
 |  |  |
*|**|**| Like address
 |  |
 |  |_code
 |  |  |
*|**|**| Like address
 |  |
 |  |_dir
 |  |  | (X): ***
 |  |  |
 |  |  |_li
 |  |     | (Xa): ***
 |  |     |
*|**|*****| Like dd
 |  |
 |  |_dl
 |  |  |
 |  |  |_dd
 |  |  |  |
 |  |  |  |_#PCDATA
 |  |  |  |_a ...
 |  |  |  |_b ...
 |  |  |  |_blockquote ...
 |  |  |  |_br ...
 |  |  |  |_cite ...
 |  |  |  |_code ...
 |  |  |  |_dir ...
 |  |  |  |_dl ...
 |  |  |  |_em ...
 |  |  |  |_form ...
 |  |  |  |_i ...
 |  |  |  |_img ...
 |  |  |  |_isindex ...
 |  |  |  |_kbd ...
 |  |  |  |_listing ...
 |  |  |  |_menu ...
 |  |  |  |_ol ...
 |  |  |  |_p ...
 |  |  |  |_pre ...
 |  |  |  |_samp ...
 |  |  |  |_strong ...
 |  |  |  |_tt ...
 |  |  |  |_ul ...
 |  |  |  |_var ...
 |  |  |  |_xmp ...
 |  |  |
 |  |  |_dt
 |  |     |
 |  |     |_#PCDATA
 |  |     |_a ...
 |  |     |_b ...
 |  |     |_br ...
 |  |     |_cite ...
 |  |     |_code ...
 |  |     |_em ...
 |  |     |_i ...
 |  |     |_img ...
 |  |     |_kbd ...
 |  |     |_samp ...
 |  |     |_strong ...
 |  |     |_tt ...
 |  |     |_var ...
 |  |
 |  |_em
 |  |  |
*|**|**| Like h1
 |  |
 |  |_form
 |  |  | (I): ***
 |  |  | (X): form
 |  |  |
 |  |  |_#PCDATA
 |  |  |_a ...
 |  |  |_address ...
 |  |  |_b ...
 |  |  |_blockquote ...
 |  |  |_br ...
 |  |  |_cite ...
 |  |  |_code ...
 |  |  |_dir ...
 |  |  |_dl ...
 |  |  |_em ...
```

```
 |  |  |_h1 ...
 |  |  |_h2 ...
 |  |  |_h3 ...
 |  |  |_h4 ...
 |  |  |_h5 ...
 |  |  |_h6 ...
 |  |  |_hr ...
 |  |  |_i ...
 |  |  |_img ...
 |  |  |_input
 |  |  |  | (Ia): ***
 |  |  |  | (Xa): form
 |  |  |  |
 |  |  |  |_EMPTY
 |  |  |
 |  |  |_isindex ...
 |  |  |_kbd ...
 |  |  |_listing ...
 |  |  |_menu ...
 |  |  |_ol ...
 |  |  |_p ...
 |  |  |_pre ...
 |  |  |_samp ...
 |  |  |_select
 |  |  |  | (Ia): ***
 |  |  |  | (Xa): form
 |  |  |  |
 |  |  |  |_option
 |  |  |     | (Ia): ***
 |  |  |     | (Xa): form
 |  |  |     |
 |  |  |     |_#PCDATA
 |  |  |
 |  |  |_strong ...
 |  |  |_textarea
 |  |  |  | (Ia): ***
 |  |  |  | (Xa): form
 |  |  |  |
 |  |  |  |_#PCDATA
 |  |  |
 |  |  |_tt ...
 |  |  |_ul ...
 |  |  |_var ...
 |  |  |_xmp ...
 |  |
 |  |_h1
 |  |  |
 |  |  |_#PCDATA
 |  |  |_a ...
 |  |  |_b ...
 |  |  |_br ...
 |  |  |_cite ...
 |  |  |_code ...
 |  |  |_em ...
 |  |  |_i ...
 |  |  |_img ...
 |  |  |_kbd ...
 |  |  |_samp ...
 |  |  |_strong ...
 |  |  |_tt ...
 |  |  |_var ...
 |  |
 |  |_h2 to h6
 |  |
 |  |  |
*|**|**| Like h1
 |  |
 |  |_hr
 |  |  |
 |  |  |_EMPTY
 |  |
 |  |_i
 |  |  |
*|**|**| Like h1
 |  |
 |  |_img
 |  |  |
 |  |  |_EMPTY
 |  |
 |  |_isindex
 |  |  |
 |  |  |_EMPTY
 |  |
 |  |_kbd
 |  |  |
*|**|**| Like h1
 |  |
```

```
 |  |_listing
 |  |  |
 |  |  |_CDATA
 |  |
 |  |_menu
 |  |  | (X): ***
 |  |  |
 |  |  |_li ...
 |  |
 |  |_ol
 |  |  |
 |  |  |_li ...
 |  |
 |  |_p
 |  |  |
*|**|**| Like h1
 |  |
 |  |_pre
 |  |  |
 |  |  |_#PCDATA
 |  |  |_a ...
 |  |  |_b ...
 |  |  |_br ...
 |  |  |_cite ...
 |  |  |_code ...
 |  |  |_em ...
 |  |  |_hr ...
 |  |  |_i ...
 |  |  |_kbd ...
 |  |  |_samp ...
 |  |  |_strong ...
 |  |  |_tt ...
 |  |  |_var ...
 |  |
 |  |_samp
 |  |  |
*|**|**| Like h1
 |  |
 |  |_strong
 |  |  |
*|**|**| Like h1
 |  |
 |  |_tt
 |  |  |
*|**|**| Like h1
 |  |
 |  |_ul
 |  |  |
 |  |  |_li ...
 |  |
 |  |_var
 |  |  |
*|**|**| Like h1
 |  |
 |  |_xmp
 |     |
 |     |_CDATA
 |
 |_head
 |  |
 |  |_base
 |  |  |
 |  |  |_EMPTY
 |  |
 |  |_isindex ...
 |  |_link
 |  |  |
 |  |  |_EMPTY
 |  |
 |  |_meta
 |  |  |
 |  |  |_EMPTY
 |  |
 |  |_nextid
 |  |  |
 |  |  |_EMPTY
 |  |
 |  |_title
 |     |
 |     |_#PCDATA
 |
 |_plaintext
    |
    |_CDATA
```

Figure 10: Output of the `dtdtree` program for the HTML2 DTD

6.5 Searching and index entries

A search engine using regular expressions is available for use with the HTML2 DTD[16] (Figure 12), as well as an index containing more than 1100 words and phrases[17] (Figure 13).

6.5.1 Checking an HTML document

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

`htmlchek`	checks an HTML file for errors, and gives warnings about possible problems;
`makemenu`	makes a simple menu for HTML files, based on each file's `<TITLE>` tag; it can also make a simple table of contents based on the `<H1>`–`<H6>` heading tags;
`xtraclnk.pl`	`perl` procedure to extract links and anchors from HTML files and to isolate text contained inside the `<A>` and `<TITLE>` elements;
`dehtml`	removes all HTML markup from a document; is useful for spell checking;
`entify`	replaces 8-bit Latin-1 input by the corresponding 7-bit-safe entity references;

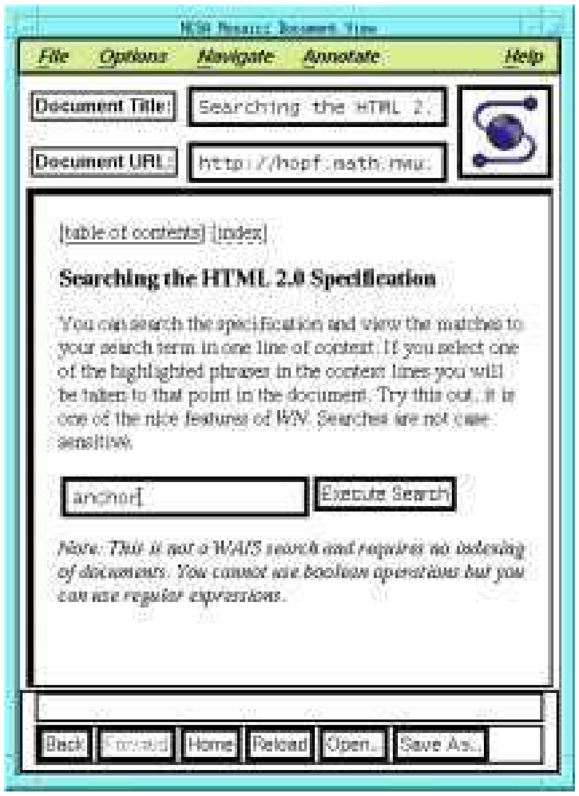
The syntax to use these programs is typically:

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

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

As an example we ran these scripts on the test files of section 6.2.1 with the results shown below, which are consistent with those obtained previously.

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


> awk -f htmlchek.awk test2.html
  Diagnostics for file "test2.html":
  <body> without preceding <head>...</head>
  Warning! at line 4 of file "test2.html"
```

[16] `http://hopf.math.nwu.edu/html2.0/dosearch.html`.
[17] `http://hopf.math.nwu.edu/html2.0/docindex.html`.
[18] The documentation is at the URL `http://uts.cc.utexas.edu/~churchh/htmlchek.html` and the tar file at `ftp://ftp.cs.buffalo.edu/pub/htmlchek/`.



Document Title: Searching the HTML 2.
Document URL: http://hopf.math.nwu.

[table of contents] [index]

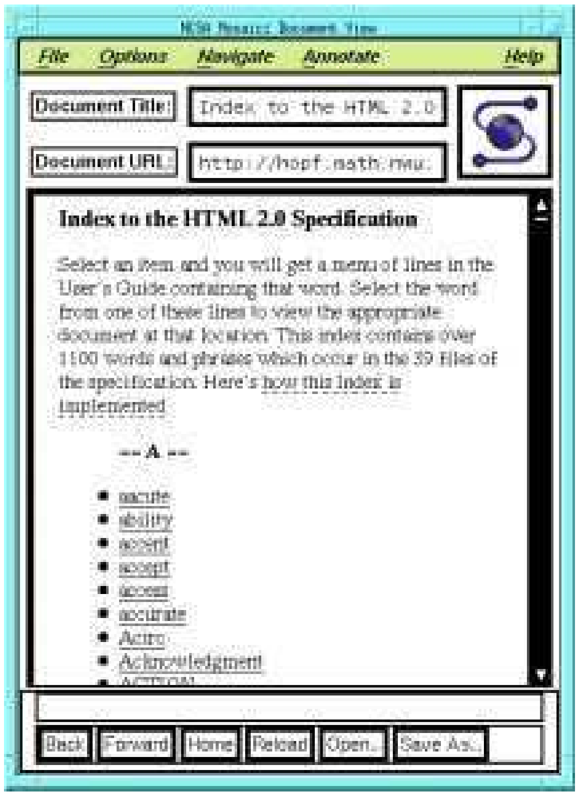
Searching the HTML 2.0 Specification

You can search the specification and view the matches to your search term in one line of context. If you select one of the highlighted phrases in the context lines you will be taken to that point in the document. Try this out, it is one of the nice features of WN. Searches are not case sensitive.

anchor | Execute Search

Note: This is not a WAIS search and requires no indexing of documents. You cannot use boolean operations but you can use regular expressions.

Figure 12: Searching the HTML2 DTD



Document Title: Index to the HTML 2.0
Document URL: http://hopf.math.nwu.

Index to the HTML 2.0 Specification

Select an item and you will get a menu of lines in the User's Guide containing that word. Select the word from one of these lines to view the appropriate document at that location. This index contains over 1100 words and phrases which occur in the 39 files of the specification. Here's how this index is implemented

-- A --

- aacute
- ability
- accent
- accept
- access
- accurate
- Acirc
- Acknowledgment

Figure 13: Index entries for the HTML2 DTD

```
  No <H1> in <body>...</body>
  Warning! at line 6 of file "test2.html"
  <HEAD> not used in document
  Warning! at END of file "test2.html"
  <TITLE> not used in document
  ERROR! at END of file "test2.html"
  Tag P occurred
  Tag HTML occurred
  Tag BODY occurred
  Tag !DOCTYPE occurred

> perl htmlchek.pl test3.html
  Diagnostics for file "test3.html":
  <LI> outside of list
  ERROR! at line 8 of file "test3.html"
  No <H1> in <body>...</body>
  Warning! at line 9 of file "test3.html"
  Tag !DOCTYPE occurred
  Tag BODY occurred
  Tag HEAD occurred
  Tag HTML occurred
  Tag LI occurred
  Tag TITLE occurred
```

7 DTD Transformations

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

7.1 sgmls.pl

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

7.2 SGMLS.pm and sgmlspl

David Megginson (University of Ottawa, Canada, dmeggins@aix1.uottawa.ca) has developed a more object-oriented approach for the translations, also based on the ESIS output of nsgmls and calling event-routines for each element found in the input stream. This package includes a default configuration for translating documents marked up according to the Doc-Book DTD into HTML or LaTeX markup.

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

7.3 Conversion from Doc-Book to HTML3

The translation program generates events for each primitive in the source document and these events are handled by calling a corresponding routine. These routines then produce the corresponding HTML/LaTeX output. Thanks to its object-oriented flavour the overall architecture provides solid ground for DTD translations. The following listing gives an idea of how the conversion is implemented. In the example below two elements are translated into LaTeX. When a tag is found that can be translated, the corresponding string is produced.

```
## Program listings appear in verbatim

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

## Class names appear in typewriter.

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

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

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

sgml('</TITLE>', sub {
    my $element = shift;
    my $data = pop_output;
    if ($element->in(ARTHEADER)) {
        $title = $data;
    } elsif ($element->in(SECT1) ||
         $element->in(IMPORTANT)) {
        output "\n\n\\section{$data}\n";
        output "\\label{$id}\n" if $id;
        output "\n";
    } elsif ($element->in(TABLE)) {
        output "\\caption{$data}\n";
        output "\\label{$id}\n" if $id;
        output "\n";
```

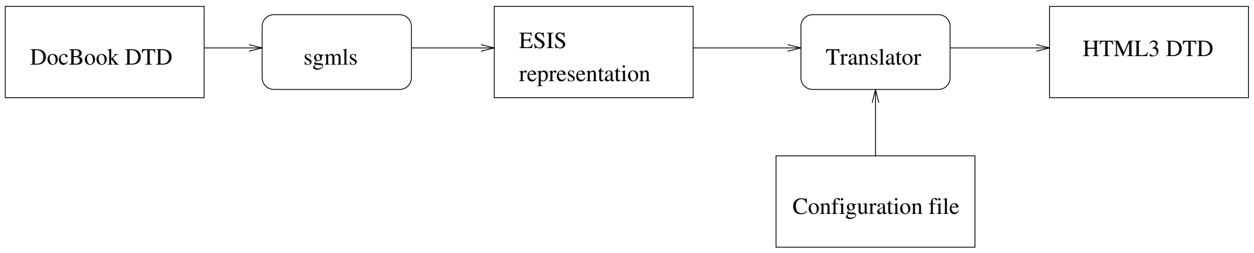


Figure 14: Schematic overview of the `Doc-Book` to HTML conversion process

```
    } else {
        die "No TITLE allowed in "
       . $element->parent->name . "\n";
    }
});
```

A conversion example of an extract from the `Doc-Book` DTD manual is given in Appendix H on page 143. It shows part of the original `Doc-Book` document markup, how it is presented in the ESIS format, and finally its translation into HTML3. Figure 14 shows the principle of the translation process.

7.4 Commercial solutions

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

8 Other Standards in the Area of Electronic Documents

SGML is part of a vast project conceived by the International Standards Organization (ISO) to develop a model to describe the complete process of creating, exchanging, editing and viewing or printing of electronic documents. This model consists of several standards, some already adopted, others still under discussion (see Goossens and van Herwijnen (1992) and Goossens and van Herwijnen (1992a)).

SGML (Standard Generalized Markup Language)

ISO 8879, the Standard described in this article, is concerned with the creation and editing of documents. A complementary standard is ISO 9069 (ISO, 1988a), SDIF, for "SGML Document Interchange Format". ISO/IEC 10744, the Hytime Standard, presents a formalism for the representation of hypermedia documents. The Hytime language (Goldfarb (1991), ISO (1992)) allows the descriptions of situations that are time dependent (for example CD-I).

DSSSL (Document Style Semantics and Specification Language)

International Standard ISO/IEC 10179 (ISO, 1995a), was adopted at the beginning of 1995. It presents a framework to express the concepts and actions necessary for transforming a structurally marked up document into its final physical form. Although this Standard is primarily targeted at document handling, it can also define other layouts, such as those needed for use with databases.[19]

SPDL (Standard Page Description Language)

International Standard ISO/IEC 10180 (ISO, 1995) defines a formalism for the description of documents in their final, completely typeset, unrevisable form.[20] The structure of the language and its syntax strongly resemble the PostScript language, which is not surprising since PostScript has become the *de facto* standard page description language.

Fonts

To exchange documents one must also define a font standard. ISO/IEC 9541 (ISO, 1991) describes a method for naming and grouping glyphs or glyph collections independently of a particular font language (such as PostScript or Truetype).

Acknowledgments

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

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

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

References

Association of American Publishers, Electronic Manuscript Series. *Reference Manual on Electronic Manuscript Preparation and Markup (Version 2).* Association of American Publishers, EPSIG, Dublin, OH, USA, 1989.

Association of American Publishers, Electronic Manuscript Series. *Author's Guide to Electronic Manuscript Preparation and Markup (Version 2).* Association of American Publishers, EPSIG, Dublin, OH, USA, 1989.

Association of American Publishers, Electronic Manuscript Series. *Markup of tabular material (Version 2).* Association of American Publishers, EPSIG, Dublin, OH, USA, 1989.

Association of American Publishers, Electronic Manuscript Series. *Markup of mathematical formulas (Version 2).* Association of American Publishers, EPSIG, Dublin, OH, USA, 1989.

Goldfarb, Charles F. HyTime: A standard for structured hypermedia interchange. *IEEE Computer*, pages 81–84, August 1991.

Goldfarb, Charles F. *The SGML Handbook.* Oxford University Press, 1990.

Goossens, Michel and Eric van Herwijnen. Introduction à SGML, DSSSL et SPDL. *Cahiers GUTenberg*, 12, pages 37–56, December 1991.

Goossens, Michel and Eric van Herwijnen. Scientific Text Processing. *International Journal of Modern Physics C*, vol. 3(3), pages 479–546, June 1992.

van Herwijnen, Eric. *Practical SGML (Second Edition).* Wolters-Kluwer Academic Publishers, Boston, 1994.

International Organization or Standardization. *Information processing — Text and office systems — Standard Generalized Markup Language (SGML).* ISO 8879:1986(E), ISO Geneva, 1986.

International Organization for Standardization, *Code for the presentation of names of languages.* ISO 639:1988 (E/F), ISO Geneva, 1988.

International Organization for Standardization, *Information processing — SGML support facilities — SGML Document Interchange Format (SDIF).* ISO 9069:1988, ISO Geneva, 1988.

International Organization for Standardization. *Information Technology — Font information interchange (three parts).* ISO/IEC 9541-1,2,3, ISO Geneva, 1991 and 1993.

International Organization for Standardization. *Information Technology — Hypermedia/Time-based Structuring Language (Hytime).* ISO/IEC 10744:1992, ISO Geneva, 1992.

International Organization for Standardization. *Information Technology — Text Communication — Standard Page Description Language (SPDL).* ISO/IEC 10180, ISO Geneva, 1995.

International Organization for Standardization. *Information processing — Text and office systems — Document Style Semantics and Specification Language (DSSSL).* ISO/IEC 10179.2, ISO Geneva, 1995.

Karney, James "SGML and HTML Tag Masters." *PC Magazine*, **14** (3), pages 144–162, 1995.

Karney, James "SGML: It's Still à la Carte." *PC Magazine*, **14** (3), pages 168–171, 1995.

Ores, Pauline. "Hypertext Publishing — Edit Trial." *PC Magazine*, **14** (3), pages 132–143, 1995.

Vignaud, Dominique. *L'édition structurée des documents.* Éditions du Cercle de la Librairie, Paris, 1990.

◇ Michel Goossens and Janne Saarela
CERN, CN Division, CH-1211 Geneva 23, Switzerland
Email: `goossens@cern.ch` and `saarela@cern.ch`

Appendices

B The DTD of the HTML2 Language

```
<!--    html.dtd

        Document Type Definition for the HyperText Markup Language
                 (HTML DTD)

        $Id: html.dtd,v 1.25 1995/03/29 18:53:13 connolly Exp $

        Author: Daniel W. Connolly <connolly@w3.org>
        See Also: html.decl, html-0.dtd, html-1.dtd
          http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html
-->

<!ENTITY % HTML.Version
        "-//IETF//DTD HTML 2.0//EN"

        -- Typical usage:

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


<!--============ Feature Test Entities ========================-->

<!ENTITY % HTML.Recommended "IGNORE"
        -- Certain features of the language are necessary for
           compatibility with widespread usage, but they may
           compromise the structural integrity of a document.
           This feature test entity enables a more prescriptive
           document type definition that eliminates
           those features.
        -->

<![ %HTML.Recommended [
        <!ENTITY % HTML.Deprecated "IGNORE">
]]>

<!ENTITY % HTML.Deprecated "INCLUDE"
        -- Certain features of the language are necessary for
           compatibility with earlier versions of the specification,
           but they tend to be used an implemented inconsistently,
           and their use is deprecated. This feature test entity
           enables a document type definition that eliminates
           these features.
        -->

<!ENTITY % HTML.Highlighting "INCLUDE"
        -- Use this feature test entity to validate that a
           document uses no highlighting tags, which may be
           ignored on minimal implementations.
        -->

<!ENTITY % HTML.Forms "INCLUDE"
        -- Use this feature test entity to validate that a document
           contains no forms, which may not be supported in minimal
           implementations
        -->

<!--============== Imported Names ==============================-->

<!ENTITY % Content-Type "CDATA"
        -- meaning an internet media type
           (aka MIME content type, as per RFC1521)
        -->

<!ENTITY % HTTP-Method "GET | POST"
        -- as per HTTP specification, in progress
        -->

<!ENTITY % URI "CDATA"
        -- The term URI means a CDATA attribute
           whose value is a Uniform Resource Identifier,
           as defined by
        "Universal Resource Identifiers" by Tim Berners-Lee
        aka RFC 1630

        Note that CDATA attributes are limited by the LITLEN
        capacity (1024 in the current version of html.decl),
        so that URIs in HTML have a bounded length.

        -->


<!--========= DTD "Macros" =====================-->

<!ENTITY % heading "H1|H2|H3|H4|H5|H6">

<!ENTITY % list " UL | OL | DIR | MENU " >


<!--======= Character mnemonic entities =================-->


<!ENTITY % ISOlat1 PUBLIC
  "-//IETF//ENTITIES Added Latin 1 for HTML//EN" "iso-lat1.gml">

%ISOlat1;

<!ENTITY amp CDATA "&#38;"     -- ampersand          -->
<!ENTITY gt CDATA "&#62;"      -- greater than       -->
<!ENTITY lt CDATA "&#60;"      -- less than          -->
<!ENTITY quot CDATA "&#34;"    -- double quote       -->


<!--========= SGML Document Access (SDA) Parameter Entities =====-->

<!-- HTML 2.0 contains SGML Document Access (SDA) fixed attributes
in support of easy transformation to the International Committee
for Accessible Document Design (ICADD) DTD
         "-//EC-USA-CDA/ICADD//DTD ICADD22//EN".
ICADD applications are designed to support usable access to
structured information by print-impaired individuals through
Braille, large print and voice synthesis.  For more information on
SDA & ICADD:
        - ISO 12083:1993, Annex A.8, Facilities for Braille,
          large print and computer voice
        - ICADD ListServ
          <ICADD%ASUACAD.BITNET@ARIZVM1.ccit.arizona.edu>
        - Usenet news group bit.listserv.easi
        - Recording for the Blind, +1 800 221 4792
-->

<!ENTITY % SDAFORM  "SDAFORM  CDATA  #FIXED"
          -- one to one mapping        -->
<!ENTITY % SDARULE  "SDARULE  CDATA  #FIXED"
          -- context-sensitive mapping -->
<!ENTITY % SDAPREF  "SDAPREF  CDATA  #FIXED"
          -- generated text prefix     -->
<!ENTITY % SDASUFF  "SDASUFF  CDATA  #FIXED"
          -- generated text suffix     -->
<!ENTITY % SDASUSP  "SDASUSP  NAME   #FIXED"
          -- suspend transform process -->


<!--========== Text Markup =====================-->

<![ %HTML.Highlighting [

<!ENTITY % font " TT | B | I ">

<!ENTITY % phrase "EM | STRONG | CODE | SAMP | KBD | VAR | CITE ">

<!ENTITY % text "#PCDATA | A | IMG | BR | %phrase | %font">

<!ELEMENT (%font;|%phrase) - - (%text)*>
<!ATTLIST ( TT | CODE | SAMP | KBD | VAR )
        %SDAFORM; "Lit"
        >
<!ATTLIST ( B | STRONG )
        %SDAFORM; "B"
        >
<!ATTLIST ( I | EM | CITE )
        %SDAFORM; "It"
        >

<!-- <TT>       Typewriter text                         -->
<!-- <B>        Bold text                               -->
<!-- <I>        Italic text                             -->

<!-- <EM>       Emphasized phrase                       -->
<!-- <STRONG>   Strong emphais                          -->
<!-- <CODE>     Source code phrase                      -->
<!-- <SAMP>     Sample text or characters               -->
<!-- <KBD>      Keyboard phrase, e.g. user input        -->
<!-- <VAR>      Variable phrase or substituable         -->
<!-- <CITE>     Name or title of cited work             -->

<!ENTITY % pre.content "#PCDATA | A | HR | BR | %font | %phrase">

]]>

<!ENTITY % text "#PCDATA | A | IMG | BR">

<!ELEMENT BR    - O EMPTY>
<!ATTLIST BR
        %SDAPREF; "&#RE;"
        >

<!-- <BR>       Line break      -->


<!--========= Link Markup ======================-->

<![ %HTML.Recommended [
        <!ENTITY % linkName "ID">
]]>

<!ENTITY % linkName "CDATA">

<!ENTITY % linkType "NAME"
        -- a list of these will be specified at a later date -->

<!ENTITY % linkExtraAttributes
        "REL %linkType #IMPLIED
        REV %linkType #IMPLIED
        URN CDATA #IMPLIED
        TITLE CDATA #IMPLIED
        METHODS NAMES #IMPLIED
        ">

<![ %HTML.Recommended [
        <!ENTITY % A.content   "(%text)*"
        -- <H1><a name="xxx">Heading</a></H1>
                is preferred to
           <a name="xxx"><H1>Heading</H1></a>
        -->
]]>

<!ENTITY % A.content   "(%heading|%text)*">

<!ELEMENT A     - - %A.content -(A)>
<!ATTLIST A
        HREF %URI #IMPLIED
        NAME %linkName #IMPLIED
        %linkExtraAttributes;
        %SDAPREF; "<Anchor: #AttList>"
        >
<!-- <A>                Anchor; source/destination of link      -->
<!-- <A NAME="...">     Name of this anchor                     -->
<!-- <A HREF="...">     Address of link destination             -->
<!-- <A URN="...">      Permanent address of destination        -->
<!-- <A REL=...>        Relationship to destination             -->
<!-- <A REV=...>        Relationship of destination to this     -->
<!-- <A TITLE="...">    Title of destination (advisory)         -->
<!-- <A METHODS="...">  Operations on destination (advisory)    -->


<!--========== Images ==========================-->

<!ELEMENT IMG    - O EMPTY>
<!ATTLIST IMG
        SRC %URI;  #REQUIRED
        ALT CDATA #IMPLIED
        ALIGN (top|middle|bottom) #IMPLIED
        ISMAP (ISMAP) #IMPLIED
        %SDAPREF; "<Fig><?SDATrans Img: #AttList>#AttVal(Alt)</Fig>"
        >

<!-- <IMG>              Image; icon, glyph or illustration      -->
<!-- <IMG SRC="...">    Address of image object                 -->
<!-- <IMG ALT="...">    Textual alternative                     -->
<!-- <IMG ALIGN=...>    Position relative to text               -->
<!-- <IMG ISMAP>        Each pixel can be a link                -->

<!--========== Paragraphs=======================-->

<!ELEMENT P     - O (%text)*>
<!ATTLIST P
        %SDAFORM; "Para"
        >

<!-- <P>        Paragraph       -->


<!--========== Headings, Titles, Sections ===============-->

<!ELEMENT HR    - O EMPTY>
<!ATTLIST HR
        %SDAPREF; "&#RE;&#RE;"
        >

<!-- <HR>       Horizontal rule -->

<!ELEMENT ( %heading )  - -  (%text;)*>
<!ATTLIST H1
        %SDAFORM; "H1"
        >
<!ATTLIST H2
        %SDAFORM; "H2"
        >
<!ATTLIST H3
        %SDAFORM; "H3"
        >
<!ATTLIST H4
        %SDAFORM; "H4"
        >
<!ATTLIST H5
        %SDAFORM; "H5"
        >
<!ATTLIST H6
        %SDAFORM; "H6"
        >

<!-- <H1>       Heading, level 1 -->
<!-- <H2>       Heading, level 2 -->
<!-- <H3>       Heading, level 3 -->
<!-- <H4>       Heading, level 4 -->
<!-- <H5>       Heading, level 5 -->
<!-- <H6>       Heading, level 6 -->


<!--========== Text Flows ======================-->

<![ %HTML.Forms [
        <!ENTITY % block.forms "BLOCKQUOTE | FORM | ISINDEX">
]]>

<!ENTITY % block.forms "BLOCKQUOTE">

<![ %HTML.Deprecated [
        <!ENTITY % preformatted "PRE | XMP | LISTING">
]]>

<!ENTITY % preformatted "PRE">

<!ENTITY % block "P | %list | DL
        | %preformatted
        | %block.forms">

<!ENTITY % flow "(%text|%block)*">

<!ENTITY % pre.content "#PCDATA | A | HR | BR">
<!ELEMENT PRE - - (%pre.content)*>
<!ATTLIST PRE
```

```
        WIDTH NUMBER #implied
        %SDAFORM; "Lit"
        >

<!-- <PRE>              Preformatted text               -->
<!-- <PRE WIDTH=...>    Maximum characters per line     -->

<![ %HTML.Deprecated [

<!ENTITY % literal "CDATA"
        -- historical, non-conforming parsing mode where
           the only markup signal is the end tag
           in full
        -->

<!ELEMENT (XMP|LISTING) - -  %literal>
<!ATTLIST XMP
        %SDAFORM; "Lit"
        %SDAPREF; "Example:&#RE;"
        >
<!ATTLIST LISTING
        %SDAFORM; "Lit"
        %SDAPREF; "Listing:&#RE;"
        >

<!-- <XMP>              Example section         -->
<!-- <LISTING>          Computer listing        -->

<!ELEMENT PLAINTEXT - O %literal>
<!-- <PLAINTEXT>        Plain text passage      -->

<!ATTLIST PLAINTEXT
        %SDAFORM; "Lit"
        >
]]>


<!--========== Lists ==================-->

<!ELEMENT DL    - -  (DT | DD)+>
<!ATTLIST DL
        COMPACT (COMPACT) #IMPLIED
        %SDAFORM; "List"
        %SDAPREF; "Definition List:"
        >

<!ELEMENT DT    - O (%text)*>
<!ATTLIST DT
        %SDAFORM; "Term"
        >

<!ELEMENT DD    - O %flow>
<!ATTLIST DD
        %SDAFORM; "LItem"
        >

<!-- <DL>               Definition list, or glossary    -->
<!-- <DL COMPACT>       Compact style list              -->
<!-- <DT>               Term in definition list         -->
<!-- <DD>               Definition of term              -->

<!ELEMENT (OL|UL) - -  (LI)+>
<!ATTLIST OL
        COMPACT (COMPACT) #IMPLIED
        %SDAFORM; "List"
        >
<!ATTLIST UL
        COMPACT (COMPACT) #IMPLIED
        %SDAFORM; "List"
        >
<!-- <UL>               Unordered list                  -->
<!-- <UL COMPACT>       Compact list style              -->
<!-- <OL>               Ordered, or numbered list       -->
<!-- <OL COMPACT>       Compact list style              -->


<!ELEMENT (DIR|MENU) - -  (LI)+ -(%block)>
<!ATTLIST DIR
        COMPACT (COMPACT) #IMPLIED
        %SDAFORM; "List"
        %SDAPREF; "<LHead>Directory</LHead>"
        >
<!ATTLIST MENU
        COMPACT (COMPACT) #IMPLIED
        %SDAFORM; "List"
        %SDAPREF; "<LHead>Menu</LHead>"
        >

<!-- <DIR>              Directory list                  -->
<!-- <DIR COMPACT>      Compact list style              -->
<!-- <MENU>             Menu list                       -->
<!-- <MENU COMPACT>     Compact list style              -->

<!ELEMENT LI    - O %flow>
<!ATTLIST LI
        %SDAFORM; "LItem"
        >

<!-- <LI>               List item                       -->

<!--========== Document Body ===================-->

<![ %HTML.Recommended [
        <!ENTITY % body.content "(%heading|%block|HR|ADDRESS|IMG)*"
        -- <h1>Heading</h1>
           <p>Text ...
                is preferred to
           <h1>Heading</h1>
           Text ...
        -->
]]>

<!ENTITY % body.content "(%heading | %text | %block |
                                 HR | ADDRESS)*">

<!ELEMENT BODY O O  %body.content>

<!-- <BODY>     Document body   -->

<!ELEMENT BLOCKQUOTE - - %body.content>
<!ATTLIST BLOCKQUOTE
        %SDAFORM; "BQ"
        >

<!-- <BLOCKQUOTE>       Quoted passage  -->

<!ELEMENT ADDRESS - - (%text|P)*>
<!ATTLIST  ADDRESS
        %SDAFORM; "Lit"
        %SDAPREF; "Address:&#RE;"
        >

<!-- <ADDRESS>  Address, signature, or byline   -->


<!--======= Forms ====================-->

<![ %HTML.Forms [

<!ELEMENT FORM - - %body.content -(FORM) +(INPUT|SELECT|TEXTAREA)>
<!ATTLIST FORM
        ACTION %URI #IMPLIED
        METHOD (%HTTP-Method) GET
        ENCTYPE %Content-Type; "application/x-www-form-urlencoded"
        %SDAPREF; "<Para>Form:</Para>"
        %SDASUFF; "<Para>Form End.</Para>"
        >

<!-- <FORM>                     Fill-out or data-entry form     -->
<!-- <FORM ACTION="...">        Address for completed form      -->
<!-- <FORM METHOD=...>          Method of submitting form       -->
<!-- <FORM ENCTYPE="...">       Representation of form data     -->

<!ENTITY % InputType "(TEXT | PASSWORD | CHECKBOX |
                        RADIO | SUBMIT | RESET |
                        IMAGE | HIDDEN )">
<!ELEMENT INPUT - O EMPTY>
<!ATTLIST INPUT
        TYPE %InputType TEXT
        NAME CDATA #IMPLIED
        VALUE CDATA #IMPLIED
        SRC %URI #IMPLIED
        CHECKED (CHECKED) #IMPLIED
        SIZE CDATA #IMPLIED
        MAXLENGTH NUMBER #IMPLIED
        ALIGN (top|middle|bottom) #IMPLIED
        %SDAPREF; "Input: "
        >

<!-- <INPUT>                  Form input datum               -->
<!-- <INPUT TYPE=...>         Type of input interaction      -->
<!-- <INPUT NAME=...>         Name of form datum             -->
<!-- <INPUT VALUE="...">      Default/initial/selected value -->
<!-- <INPUT SRC="...">        Address of image               -->
<!-- <INPUT CHECKED>          Initial state is "on"          -->
<!-- <INPUT SIZE=...>         Field size hint                -->
<!-- <INPUT MAXLENGTH=...>    Data length maximum            -->
<!-- <INPUT ALIGN=...>        Image alignment                -->

<!ELEMENT SELECT - - (OPTION+) -(INPUT|SELECT|TEXTAREA)>
<!ATTLIST SELECT
        NAME CDATA #REQUIRED
        SIZE NUMBER #IMPLIED
        MULTIPLE (MULTIPLE) #IMPLIED
        %SDAFORM; "List"
        %SDAPREF;
        "<LHead>Select #AttVal(Multiple)</LHead>"
        >

<!-- <SELECT>                 Selection of option(s)         -->
<!-- <SELECT NAME=...>        Name of form datum             -->
<!-- <SELECT SIZE=...>        Options displayed at a time    -->
<!-- <SELECT MULTIPLE>        Multiple selections allowed    -->

<!ELEMENT OPTION - O (#PCDATA)*>
<!ATTLIST OPTION
        SELECTED (SELECTED) #IMPLIED
        VALUE CDATA #IMPLIED
        %SDAFORM; "LItem"
        %SDAPREF;
        "Option: #AttVal(Value) #AttVal(Selected)"
        >

<!-- <OPTION>                 A selection option             -->
<!-- <OPTION SELECTED>        Initial state                  -->
<!-- <OPTION VALUE="...">     Form datum value for this option-->

<!ELEMENT TEXTAREA - - (#PCDATA)* -(INPUT|SELECT|TEXTAREA)>
<!ATTLIST TEXTAREA
        NAME CDATA #REQUIRED
        ROWS NUMBER #REQUIRED
        COLS NUMBER #REQUIRED
        %SDAFORM; "Para"
        %SDAPREF; "Input Text -- #AttVal(Name): "
        >

<!-- <TEXTAREA>               An area for text input         -->
<!-- <TEXTAREA NAME=...>      Name of form datum             -->
<!-- <TEXTAREA ROWS=...>      Height of area                 -->
<!-- <TEXTAREA COLS=...>      Width of area                  -->

]]>


<!--======= Document Head ======================-->

<![ %HTML.Recommended [
        <!ENTITY % head.extra "META* & LINK*">
]]>

<!ENTITY % head.extra "NEXTID? & META* & LINK*">

<!ENTITY % head.content "TITLE & ISINDEX? & BASE? &
                         (%head.extra)">

<!ELEMENT HEAD O O  (%head.content)>

<!-- <HEAD>     Document head   -->

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

```
<!ATTLIST TITLE
        %SDAFORM; "Ti"    >

<!-- <TITLE>    Title of document -->

<!ELEMENT LINK - O EMPTY>
<!ATTLIST LINK
        HREF %URI #REQUIRED
        %linkExtraAttributes;
        %SDAPREF; "Linked to : #AttVal (TITLE) (URN) (HREF)>"    >

<!-- <LINK>             Link from this document                 -->
<!-- <LINK HREF="...">  Address of link destination             -->
<!-- <LINK URN="...">   Lasting name of destination             -->
<!-- <LINK REL=...>     Relationship to destination             -->
<!-- <LINK REV=...>     Relationship of destination to this     -->
<!-- <LINK TITLE="..."> Title of destination (advisory)         -->
<!-- <LINK METHODS="..."> Operations allowed (advisory)         -->

<!ELEMENT ISINDEX - O EMPTY>
<!ATTLIST ISINDEX
        %SDAPREF;
   "<Para>[Document is indexed/searchable.]</Para>">

<!-- <ISINDEX>          Document is a searchable index          -->

<!ELEMENT BASE - O EMPTY>
<!ATTLIST BASE
        HREF %URI; #REQUIRED     >

<!-- <BASE>             Base context document                   -->
<!-- <BASE HREF="...">  Address for this document               -->

<!ELEMENT NEXTID - O EMPTY>
<!ATTLIST NEXTID
        N %linkName #REQUIRED     >

<!-- <NEXTID>           Next ID to use for link name            -->
<!-- <NEXTID N=...>     Next ID to use for link name            -->

<!ELEMENT META - O EMPTY>
<!ATTLIST META
        HTTP-EQUIV  NAME    #IMPLIED
        NAME        NAME    #IMPLIED
        CONTENT     CDATA   #REQUIRED    >

<!-- <META>                     Generic Metainformation         -->
<!-- <META HTTP-EQUIV=...>      HTTP response header name       -->
<!-- <META NAME=...>            Metainformation name            -->
<!-- <META CONTENT="...">       Associated information          -->

<!--======= Document Structure =================-->

<![ %HTML.Deprecated [
        <!ENTITY % html.content "HEAD, BODY, PLAINTEXT?">
]]>
<!ENTITY % html.content "HEAD, BODY">

<!ELEMENT HTML O O  (%html.content)>
<!ENTITY % version.attr "VERSION CDATA #FIXED '%HTML.Version;'">

<!ATTLIST HTML
        %version.attr;
        %SDAFORM; "Book"
        >

<!-- <HTML>                     HTML Document   -->
```

C The HTML2 SGML Declaration

```
<!SGML  "ISO 8879:1986"
--
        SGML Declaration for HyperText Markup Language (HTML).

```

```
--

CHARSET
         BASESET  "ISO 646:1983//CHARSET
                   International Reference Version
                   (IRV)//ESC 2/5 4/0"
         DESCSET  0   9   UNUSED
                  9   2   9
                  11  2   UNUSED
                  13  1   13
                  14  18  UNUSED
                  32  95  32
                  127 1   UNUSED
     BASESET   "ISO Registration Number 100//CHARSET
                ECMA-94 Right Part of
                Latin Alphabet Nr. 1//ESC 2/13 4/1"

         DESCSET  128  32   UNUSED
                  160  96    32

CAPACITY        SGMLREF
                TOTALCAP        150000
                GRPCAP          150000

SCOPE    DOCUMENT
SYNTAX
         SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
                 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 127
         BASESET  "ISO 646:1983//CHARSET
                   International Reference Version
                   (IRV)//ESC 2/5 4/0"
         DESCSET  0 128 0
         FUNCTION
                  RE          13
                  RS          10
                  SPACE       32
                  TAB SEPCHAR  9


         NAMING   LCNMSTRT ""
                  UCNMSTRT ""
                  LCNMCHAR ".-"
                  UCNMCHAR ".-"
                  NAMECASE GENERAL YES
                           ENTITY  NO
         DELIM    GENERAL  SGMLREF
                  SHORTREF SGMLREF
         NAMES    SGMLREF
         QUANTITY SGMLREF
                  ATTSPLEN 2100
                  LITLEN   1024
                  NAMELEN  72    -- somewhat arbitrary; taken from
                                internet line length conventions --
                  PILEN    1024
                  TAGLEN   2100
                  GRPGTCNT 150
                  GRPCNT   64

FEATURES
  MINIMIZE
    DATATAG  NO
    OMITTAG  YES
    RANK     NO
    SHORTTAG YES
  LINK
    SIMPLE   NO
    IMPLICIT NO
    EXPLICIT NO
  OTHER
    CONCUR   NO
    SUBDOC   NO
    FORMAL   YES
  APPINFO    "SDA"  -- conforming SGML Document Access application
                    --
>
<!--
        $Id: html.decl,v 1.14 1995/02/10 22:20:05 connolly Exp $

        Author: Daniel W. Connolly <connolly@hal.com>

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

D The SGML Open HTML Catalog File

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

```
        -- catalog: SGML Open style entity catalog for HTML --
        -- $Id: catalog,v 1.2 1994/11/30 23:45:18 connolly Exp $ --

        -- Ways to refer to Level 2: most general to most specific --
PUBLIC  "-//IETF//DTD HTML//EN"                 /afs/cern.ch/user/j/jsaarela/sgml/dtds/html.dtd
PUBLIC  "-//IETF//DTD HTML 2.0//EN"             /afs/cern.ch/user/j/jsaarela/sgml/dtds/html.dtd
PUBLIC  "-//IETF//DTD HTML Level 2//EN"         /afs/cern.ch/user/j/jsaarela/sgml/dtds/html.dtd
PUBLIC  "-//IETF//DTD HTML 2.0 Level 2//EN"     /afs/cern.ch/user/j/jsaarela/sgml/dtds/html.dtd

        -- Ways to refer to Level 1: most general to most specific --
PUBLIC  "-//IETF//DTD HTML Level 1//EN"         /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-1.dtd
PUBLIC  "-//IETF//DTD HTML 2.0 Level 1//EN"     /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-1.dtd

        -- Ways to refer to Level 0: most general to most specific --
PUBLIC  "-//IETF//DTD HTML Level 0//EN"         /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-0.dtd
PUBLIC  "-//IETF//DTD HTML 2.0 Level 0//EN"     /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-0.dtd


        -- Ways to refer to Strict Level 2: most general to most specific --
PUBLIC  "-//IETF//DTD HTML Strict//EN"                  /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-s.dtd
PUBLIC  "-//IETF//DTD HTML 2.0 Strict//EN"              /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-s.dtd
PUBLIC  "-//IETF//DTD HTML Strict Level 2//EN"          /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-s.dtd
PUBLIC  "-//IETF//DTD HTML 2.0 Strict Level 2//EN"      /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-s.dtd

        -- Ways to refer to Strict Level 1: most general to most specific --
PUBLIC  "-//IETF//DTD HTML Strict Level 1//EN"          /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-1s.dtd
PUBLIC  "-//IETF//DTD HTML 2.0 Strict Level 1//EN"      /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-1s.dtd

        -- Ways to refer to Strict Level 0: most general to most specific --
PUBLIC  "-//IETF//DTD HTML Strict Level 0//EN"          /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-0s.dtd
PUBLIC  "-//IETF//DTD HTML 2.0 Strict Level 0//EN"      /afs/cern.ch/user/j/jsaarela/sgml/dtds/html-0s.dtd

        -- PUBLIC entity sets --
PUBLIC  "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML" /afs/cern.ch/user/j/jsaarela/sgml/dtds/iso-lat1.gml
PUBLIC  "-//IETF//ENTITIES Added Latin 1 for HTML//EN"  /afs/cern.ch/user/j/jsaarela/sgml/dtds/iso-lat1.gml
PUBLIC  "-//IETF//ENTITIES icons for HTML//EN"          /afs/cern.ch/user/j/jsaarela/sgml/dtds/htmlicons.gml
PUBLIC  "-//IETF//ENTITIES Math and Greek for HTML//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/iso-grk1.gml

        -- HTML3 --
PUBLIC  "-//IETF//DTD HTML 3.0//EN//"           /afs/cern.ch/user/j/jsaarela/sgml/dtds/html3.dtd

        -- ISO 12083 --
PUBLIC  "ISO 12083:1994//DTD Mathematics//EN"           /afs/cern.ch/user/j/jsaarela/sgml/dtds/math.dtd
PUBLIC  "-//ISO 12083:1994//DTD Mathematics//EN"                /afs/cern.ch/user/j/jsaarela/sgml/dtds/math.dtd
PUBLIC  "ISO 12083:1994//DTD Book//EN"                  /afs/cern.ch/user/j/jsaarela/sgml/dtds/book.dtd
PUBLIC  "-//ISO 12083:1994//DTD Book//EN"                       /afs/cern.ch/user/j/jsaarela/sgml/dtds/book.dtd

        -- DocBook DTD --
PUBLIC  "-//HaL and O'Reilly//DTD DocBook//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/docbook.2.2.1.dtd

        -- General --
PUBLIC "ISO 8879:1986//ENTITIES Publishing//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/iso-pub.gml
PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/iso-num.gml

SGMLDECL "/afs/cern.ch/user/j/jsaarela/sgml/dtds/html3.decl"
```

E The ISO-Latin1 Entity Set

To have an idea of how character entity sets are defined in practice, below is shown the file corresponding to Latin1 (standard ISO/IEC 8859-1), available as SGML public entity set `ISOlat1` with ISO 8879.

```
<!-- (C) International Organization for Standardization 1986
     Permission to copy in any form is granted for use with
     conforming SGML systems and applications as defined in
     ISO 8879, provided this notice is included in all copies.
-->
<!-- Character entity set. Typical invocation:
     <!ENTITY % ISOlat1 PUBLIC
       "ISO 8879-1986//ENTITIES Added Latin 1//EN">
     %ISOlat1;
-->
<!ENTITY aacute SDATA "[aacute]"--=small a, acute accent-->
<!ENTITY Aacute SDATA "[Aacute]"--=capital A, acute accent-->
<!ENTITY acirc  SDATA "[acirc ]"--=small a, circumflex accent-->
<!ENTITY Acirc  SDATA "[Acirc ]"--=capital A, circumflex accent-->
<!ENTITY agrave SDATA "[agrave]"--=small a, grave accent-->
<!ENTITY Agrave SDATA "[Agrave]"--=capital A, grave accent-->
<!ENTITY aring  SDATA "[aring ]"--=small a, ring-->
<!ENTITY Aring  SDATA "[Aring ]"--=capital A, ring-->
<!ENTITY atilde SDATA "[atilde]"--=small a, tilde-->
<!ENTITY Atilde SDATA "[Atilde]"--=capital A, tilde-->
<!ENTITY auml   SDATA "[auml  ]"--=small a, dieresis or umlaut mark-->
<!ENTITY Auml   SDATA "[Auml  ]"--=capital A, dieresis or umlaut mark-->
<!ENTITY aelig  SDATA "[aelig ]"--=small ae diphthong (ligature)-->
<!ENTITY AElig  SDATA "[AElig ]"--=capital AE diphthong (ligature)-->
<!ENTITY ccedil SDATA "[ccedil]"--=small c, cedilla-->
<!ENTITY Ccedil SDATA "[Ccedil]"--=capital C, cedilla-->
<!ENTITY eth    SDATA "[eth   ]"--=small eth, Icelandic-->
<!ENTITY ETH    SDATA "[ETH   ]"--=capital Eth, Icelandic-->
<!ENTITY eacute SDATA "[eacute]"--=small e, acute accent-->
<!ENTITY Eacute SDATA "[Eacute]"--=capital E, acute accent-->
<!ENTITY ecirc  SDATA "[ecirc ]"--=small e, circumflex accent-->
<!ENTITY Ecirc  SDATA "[Ecirc ]"--=capital E, circumflex accent-->
<!ENTITY egrave SDATA "[egrave]"--=small e, grave accent-->
<!ENTITY Egrave SDATA "[Egrave]"--=capital E, grave accent-->
<!ENTITY euml   SDATA "[euml  ]"--=small e, dieresis or umlaut mark-->
<!ENTITY Euml   SDATA "[Euml  ]"--=capital E, dieresis or umlaut mark-->
<!ENTITY iacute SDATA "[iacute]"--=small i, acute accent-->
<!ENTITY Iacute SDATA "[Iacute]"--=capital I, acute accent-->
<!ENTITY icirc  SDATA "[icirc ]"--=small i, circumflex accent-->
<!ENTITY Icirc  SDATA "[Icirc ]"--=capital I, circumflex accent-->
<!ENTITY igrave SDATA "[igrave]"--=small i, grave accent-->
<!ENTITY Igrave SDATA "[Igrave]"--=capital I, grave accent-->
<!ENTITY iuml   SDATA "[iuml  ]"--=small i, dieresis or umlaut mark-->
<!ENTITY Iuml   SDATA "[Iuml  ]"--=capital I, dieresis or umlaut mark-->
<!ENTITY ntilde SDATA "[ntilde]"--=small n, tilde-->
<!ENTITY Ntilde SDATA "[Ntilde]"--=capital N, tilde-->
<!ENTITY oacute SDATA "[oacute]"--=small o, acute accent-->
<!ENTITY Oacute SDATA "[Oacute]"--=capital O, acute accent-->
<!ENTITY ocirc  SDATA "[ocirc ]"--=small o, circumflex accent-->
<!ENTITY Ocirc  SDATA "[Ocirc ]"--=capital O, circumflex accent-->
<!ENTITY ograve SDATA "[ograve]"--=small o, grave accent-->
<!ENTITY Ograve SDATA "[Ograve]"--=capital O, grave accent-->
<!ENTITY oslash SDATA "[oslash]"--=small o, slash-->
<!ENTITY Oslash SDATA "[Oslash]"--=capital O, slash-->
<!ENTITY otilde SDATA "[otilde]"--=small o, tilde-->
<!ENTITY Otilde SDATA "[Otilde]"--=capital O, tilde-->
<!ENTITY ouml   SDATA "[ouml  ]"--=small o, dieresis or umlaut mark-->
<!ENTITY Ouml   SDATA "[Ouml  ]"--=capital O, dieresis or umlaut mark-->
<!ENTITY szlig  SDATA "[szlig ]"--=small sharp s, German (sz ligature)-->
<!ENTITY thorn  SDATA "[thorn ]"--=small thorn, Icelandic-->
<!ENTITY THORN  SDATA "[THORN ]"--=capital THORN, Icelandic-->
<!ENTITY uacute SDATA "[uacute]"--=small u, acute accent-->
<!ENTITY Uacute SDATA "[Uacute]"--=capital U, acute accent-->
<!ENTITY ucirc  SDATA "[ucirc ]"--=small u, circumflex accent-->
<!ENTITY Ucirc  SDATA "[Ucirc ]"--=capital U, circumflex accent-->
<!ENTITY ugrave SDATA "[ugrave]"--=small u, grave accent-->
<!ENTITY Ugrave SDATA "[Ugrave]"--=capital U, grave accent-->
<!ENTITY uuml   SDATA "[uuml  ]"--=small u, dieresis or umlaut mark-->
<!ENTITY Uuml   SDATA "[Uuml  ]"--=capital U, dieresis or umlaut mark-->
<!ENTITY yacute SDATA "[yacute]"--=small y, acute accent-->
<!ENTITY Yacute SDATA "[Yacute]"--=capital Y, acute accent-->
<!ENTITY yuml   SDATA "[yuml  ]"--=small y, dieresis or umlaut mark-->

```

F The HTML3 DTD — Tables and Mathematics Parts

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

```
<!--====================== Captions =====================================-->

<!ELEMENT CAPTION - - (%text;)+ -- table or figure caption -->
<!ATTLIST CAPTION
        %attrs;
        align (top|bottom|left|right) #IMPLIED
        >
<!--====================== Tables =======================================-->

<!--
    Tables and figures can be aligned in several ways:

    bleedleft   flush left with the left (window) border
    left        flush left with the left text margin
    center      centered (text flow is disabled for this mode)
    right       flush right with the right text margin
    bleedright  flush right with the right (window) border
    justify     when applicable the table/figure should stretch
                to fill space between the text margins

    Note: text will flow around the table or figure if the browser
    judges there is enough room and the alignment is not centered
    or justified. The table or figure may itself be part of the
    text flow around some earlier figure. You can in this case use
    the clear or needs attributes to move the new table or figure
    down the page beyond the obstructing earlier figure. Similarly,
    you can use the clear or needs attributes with other elements
    such as headers and lists to move them further down the page.
-->

<!ENTITY % block.align
        "align  (bleedleft|left|center|right|bleedright|justify) center">

<!--
    The HTML 3.0 table model has been chosen for its simplicity
    and the ease in writing filters from common DTP packages.

    By default the table is automatically sized according to the
    cell contents and the current window size. Specifying the columns
    widths using the colspec attribute allows browsers to start
    displaying the table without having to wait for last row.

    The colspec attribute is a list of column widths and alignment
    specifications. The columns are listed from left to right with
    a capital letter followed by a number, e.g. COLSPEC="L20 C8 L40".
    The letter is L for left, C for center, R for right alignment of
    cell contents. J is for justification, when feasible, otherwise
    this is treated in the same way as L for left alignment.
    Column entries are delimited by one or more space characters.

    The number specifies the width in em’s, pixels or as a
    fractional value of the table width, as according to the
    associated units attribute. This approach is more compact
    than used with most SGML table models and chosen to simplify
    hand entry. The width attribute allows you to specify the
    width of the table in pixels, em units or as a percentage
    of the space between the current left and right margins.

    To assist with rendering to speech, row and column headers
    can be given short names using the AXIS attribute. The AXES
    attribute is used to explicitly specify the row and column
    names for use with each cell. Otherwise browsers can follow
    up columns and left along rows (right for some languages)
    to find the corresponding header cells.

    Table content model: Braille limits the width of tables,
    placing severe limits on column widths. User agents need
    to render big cells by moving the content to a note placed
    before the table. The cell is then rendered as a link to
    the corresponding note.

    To assist with formatting tables to paged media, authors
    can differentiate leading and trailing rows that are to
    be duplicated when splitting tables across page boundaries.
    The recommended way is to subclass rows with the CLASS attribute
    For example: <TR CLASS=Header>, <TR CLASS=Footer> are used for
    header and footer rows. Paged browsers insert footer rows at
    the bottom of the current page and header rows at the top of
    the new page, followed by the remaining body rows.
-->

<!ELEMENT TABLE - - (CAPTION?, TR*) -- mixed headers and data -->
<!ATTLIST TABLE
        %attrs;
        %needs; -- for control of text flow --
        border (border) #IMPLIED -- draw borders --
        colspec CDATA   #IMPLIED -- column widths and alignment --
        units  (em|pixels|relative) em -- units for column widths --
        width NUMBER    #IMPLIED -- absolute or percentage width --
        %block.align;  -- horizontal alignment --
        nowrap (nowrap) #IMPLIED -- don't wrap words --
        >

<!ENTITY % cell "TH | TD">
<!ENTITY % vertical.align "top|middle|bottom|baseline">

<!--
    Browsers should tolerate an omission of the first <TR>
    tag as it is implied by the context. Missing trailing
    <TR>s implied by rowspans should be ignored.

    The alignment attributes act as defaults for rows
    overriding the colspec attribute and being in turn
    overridden by alignment attributes on cell elements.
    Use valign=baseline when you want to ensure that text
    in different cells on the same row is aligned on the
    same baseline regardless of fonts. It only applies
    when the cells contain a single line of text.
-->

<!ELEMENT TR - O (%cell)* -- row container -->
<!ATTLIST TR
        %attrs;
        align  (left|center|right|justify) #IMPLIED
        valign (%vertical.align) top -- vertical alignment --
        nowrap (nowrap) #IMPLIED -- don't wrap words --
        >

<!--
    Note that table cells can include nested tables.
    Missing cells are considered to be empty, while
    missing rows should be ignored, i.e. if a cell
    spans a row and there are no further TR elements
    then the implied row should be ignored.
-->

<!ELEMENT (%cell) - O %body.content>
<!ATTLIST (%cell)
        %attrs;
        colspan NUMBER    1      -- columns spanned --
        rowspan NUMBER    1      -- rows spanned --
        align  (left|center|right|justify) #IMPLIED
        valign (%vertical.align) top -- vertical alignment --
        nowrap (nowrap) #IMPLIED -- don't wrap words --
        axis CDATA #IMPLIED -- axis name, defaults to element content --
        axes CDATA #IMPLIED -- comma separated list of axis names --
        >

<!--================ Entities for math symbols ============================-->

<!-- ISO subset chosen for use with the widely available Adobe math font -->

<!ENTITY % HTMLmath PUBLIC
  "-//IETF//ENTITIES Math and Greek for HTML//EN">
%HTMLmath;

<!--======================== Math  =======================================-->

<!-- Use &thinsp; &emsp; etc for greater control of spacing. -->

<!-- Subscripts and Superscripts

  <SUB> and <SUP> are used for subscripts and superscripts.

                                        i j
      X <SUP>i</SUP>Y<SUP>j</SUP>  is   X  Y

  i.e. the space following the X disambiguates the binding.
  The align attribute can be used for horizontal alignment,
  e.g. to explicitly place an index above an element:
                                              i
        X<sup align=center>i</sup>  produces  X

Short references are defined for superscripts, subscripts and boxes
to save typing when manually editing HTML math, e.g.

      x^2^    is mapped to   x<sup>2</sup>
      y_z_    is mapped to   y<sub>z</sub>
      {a+b}   is mapped to   <box>a + b</box>

Note that these only apply within the MATH element and can't be
used in normal text!
-->
<!ENTITY REF1   STARTTAG   "SUP">
<!ENTITY REF2   ENDTAG     "SUP">
<!ENTITY REF3   STARTTAG   "SUB">
<!ENTITY REF4   ENDTAG     "SUB">
<!ENTITY REF5   STARTTAG   "BOX">
<!ENTITY REF6   ENDTAG     "BOX">

<!USEMAP MAP1   MATH>
<!USEMAP MAP2   SUP>
<!USEMAP MAP3   SUB>
<!USEMAP MAP4   BOX>

<!SHORTREF MAP1 "^" REF1
                "_" REF3
                "{" REF5 >

<!SHORTREF MAP2 "^" REF2
                "_" REF3
                "{" REF5 >

<!SHORTREF MAP3 "_" REF4
                "^" REF1
                "{" REF5 >

<!SHORTREF MAP4 "}" REF6
                "^" REF1
                "_" REF3
                "{" REF5 >

<!--
 The inclusion of %math and exclusion of %notmath is used here
 to alter the content model for the B, SUB and SUP elements,
 to limit them to formulae rather than general text elements.
-->

<!ENTITY % mathvec "VEC|BAR|DOT|DDOT|HAT|TILDE" -- common accents -->
<!ENTITY % mathface "B|T|BT" -- control of font face -->
<!ENTITY % math "BOX|ABOVE|BELOW|%mathvec|ROOT|SQRT|ARRAY|SUB|SUP|%mathface">
<!ENTITY % formula "#PCDATA|%math">

<!ELEMENT MATH - - (#PCDATA)* -(%notmath) +(%math)>
<!ATTLIST MATH
        id      ID      #IMPLIED
        model   CDATA   #IMPLIED>

<!-- The BOX element acts as brackets. Delimiters are optional and
     stretch to match the height of the box. The OVER element is used
     when you want a line between numerator and denominator. This line
     is suppressed with the alternative ATOP element. CHOOSE acts like
     ATOP but adds enclosing round brackets as a convenience for binomial
     coefficients. Note the use of { and } as shorthand for <BOX> and
     </BOX> respectively:

                           1 + X
     {1 + X<OVER>Y}  is   _______
                             Y

                                 a + b
     {a + b<ATOP>c - d} is
                                 c - d

     The delimiters are represented using the LEFT and RIGHT
     elements as in:

```

```
     {[<LEFT>x + y<RIGHT>]}   is   [ x + y ]
     {(<LEFT>a<RIGHT>]}       is   (a]
     {||<LEFT>a<RIGHT>||}     is   || a ||

     Use &lbrace; and &rbrace; for "{" and "}" respectively as
     these symbols are used as shorthand for BOX, e.g.

     {&lbrace;<LEFT>a+b<RIGHT>&rbrace;}  is  {a+b}

     You can stretch definite integrals to match the integrand, e.g.

     {&int;<SUB>a</SUB><SUP>b</SUP><LEFT>{f(x)<over>1+x} dx}

            b
            /  f(x)
            | ----- dx
            / 1 + x
            a

     Note the complex content model for BOX is a work around
     for the absence of support for infix operators in SGML.

     You can get oversize delimiters with the SIZE attribute,
     for example <BOX SIZE=large>(<LEFT>...<RIGHT>)</BOX>

     Note that the names of common functions are recognized
     by the parser without the need to use "&" and ";" around
     them, e.g. int, sum, sin, cos, tan, ...
-->

<!ELEMENT BOX - - ((%formula)*, (LEFT, (%formula)*)?,
                   ((OVER|ATOP|CHOOSE), (%formula)*)?,
                   (RIGHT, (%formula)*)?)>
<!ATTLIST BOX
        size  (normal|medium|large|huge) normal -- oversize delims -->

<!ELEMENT (OVER|ATOP|CHOOSE|LEFT|RIGHT) - O EMPTY>

<!-- Horizontal line drawn ABOVE contents
     The symbol attribute allows authors to supply
     an entity name for an accent, arrow symbol etc.
     Generalisation of LaTeX's overline command.
 -->

<!ELEMENT ABOVE - - (%formula)+>
<!ATTLIST ABOVE symbol ENTITY #IMPLIED>

<!-- Horizontal line drawn BELOW contents
     The symbol attribute allows authors to
     supply an entity name for an arrow symbol etc.
     Generalisation of LaTeX's underline command.
 -->

<!ELEMENT BELOW - - (%formula)+>
<!ATTLIST BELOW symbol ENTITY #IMPLIED>

<!-- Convenience tags for common accents:
     vec, bar, dot, ddot, hat and tilde
-->

<!ELEMENT (%mathvec) - - (%formula)+>

<!--
  T and BT are used to designate terms which should
  be rendered in an upright font (& bold face for BT)
-->

<!ELEMENT (T|BT) - - (%formula)+>
<!ATTLIST (T|BT) class NAMES #IMPLIED>

<!-- Roots  e.g. <ROOT>3<OF>1+x</ROOT> -->

<!ELEMENT ROOT - - ((%formula)+, OF, (%formula)+)>
<!ELEMENT OF - O (%formula)* -- what the root applies to -->

<!ELEMENT SQRT - - (%formula)* -- square root convenience tag -->

<!-- LaTeX like arrays. The COLDEF attribute specifies
     a single capital letter for each column determining
     how the column should be aligned, e.g. coldef="CCC"
```

```

        "L"     left
        "C"     center
        "R"     right

     An optional separator letter can occur between columns
     and should be one of + - or =, e.g. "C+C+C+C=C".
     Whitespace within coldef is ignored. By default, the
     columns are all centered.

     The ALIGN attribute alters the vertical position of the
     array as compared with preceding and following expressions.

     Use LDELIM and RDELIM attributes for delimiter entities.
     When the LABELS attribute is present, the array is
     displayed with the first row and the first column as
     labels displaced from the other elements. In this case,
     the first element of the first row should normally be
     left blank.

     Use &vdots; &cdots; and &ddots; for vertical, horizontal
     and diagonal ellipsis dots. Use &dotfill; to fill an array
     cell with horizontal dots (e.g. for a full row).
     Note &ldots; places the dots on the baseline, while &cdots;
     places them higher up.
-->

<!ELEMENT ARRAY - - (ROW)+>
<!ATTLIST ARRAY
        align (top|middle|bottom) middle -- vertical alignment --
        coldef  CDATA   #IMPLIED  -- column alignment and separator --
        ldelim  NAMES   #IMPLIED  -- stretchy left delimiter --
        rdelim  NAMES   #IMPLIED  -- stretchy right delimiter --
        labels (labels) #IMPLIED  -- TeX's \bordermatrix style -->

<!ELEMENT ROW - O (ITEM)*>
<!ELEMENT ITEM - O (%formula)*>
<!ATTLIST ITEM
        align   CDATA  #IMPLIED  -- override coldef alignment --
        colspan NUMBER 1         -- merge columns as per TABLE --
        rowspan NUMBER 1         -- merge rows as per TABLE -->
```

G The ISO/IEC 12083 Mathematics DTD

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

```
<!-- This is the ISO12083:1994 document type definition for Mathematics    -->

<!-- Copyright: (C) International Organization for Standardization 1994.
Permission to copy in any form is granted for use with conforming SGML
systems and applications as defined in ISO 8879:1986, provided this notice
is included in all copies.                                                 -->

<!-- ==================================================================== -->
<!--            PUBLIC DOCUMENT TYPE DEFINITION SUBSET                -->
<!-- ==================================================================== -->

<!--
This DTD is included by the Book and Article DTDs of ISO12083:1994.
As it is a separate entity it may also be included by other DTDs.

Since there is no consensus on how to describe the semantics of formulas,
it only describes their presentational or visual structure. Since, however,
there is a strong need for such description (especially within the
print-disabled community), it is recommended that the following
declaration be added where there is a requirement for a consistent,
standardized mechanism to carry semantic meanings for the SGML
elements declared throughout this part of this International Standard:

<!ENTITY % SDAMAP       "SDAMAP   NAME    #IMPLIED"           >

and that the attribute represented by %SDAMAP; be made available for
all elements which may require a semantic association, or, in the simpler
case, be added to all elements in this DTD.                    -->
```

```



<!-- ===================================================================== -->
<!--  Parameter entities describing the possible contents of formulas.     -->
<!-- ===================================================================== -->

<!ENTITY % p.trans "bold|italic|sansser|typewrit|smallcap|roman"
                   -- character transformations                            -->
<!ENTITY % m.math "fraction|subform|sup|inf|top|bottom|middle|fence|mark|
   post|box|overline|undrline|radical|array|hspace|vspace|break|markref|
   #PCDATA" -- mathematical formula elements                               -->



<!-- ===================================================================== -->
<!-- Accessible Document and other Parameter Entities
     If this DTD is not imbedded by a ISO12083:1994 Book or Article,
     the comment delimiters should be removed.                             -->
<!-- ===================================================================== -->

<!--ENTITY % SDAFORM      "SDAFORM   CDATA    #FIXED"                      -->
<!--ENTITY % SDARULE      "SDARULE   CDATA    #FIXED"                      -->
<!--ENTITY % SDAPREF      "SDAPREF   CDATA    #FIXED"                      -->
<!--ENTITY % SDASUFF      "SDASUFF   CDATA    #FIXED"                      -->
<!--ENTITY % SDASUSP      "SDASUSP   NAME     #FIXED"                      -->



<!-- ===================================================================== -->
<!-- This entity is for an attribute to indicate which alphabet is
     used in the element (formula, dformula). You may change this to
     a notation attribute, where the notation could describe a
     keyboard mapping. Please modify the set as necessary.
     If this DTD is not imbedded by a ISO12083:1994 Book or Article,
     the comment delimiters should be removed.                             -->
<!-- ===================================================================== -->

<!-- ENTITY % a.types "(latin|greek|cyrillic|hebrew|kanji) latin"         -->


<!-- ===================================================================== -->
<!-- character transformations                                             -->
<!-- ===================================================================== -->

<!--      ELEMENT             MIN  CONTENT             EXPLANATIONS      -->
<!ELEMENT bold                - -  (%p.trans;|#PCDATA)* -- bold          -->
<!ELEMENT italic              - -  (%p.trans;|#PCDATA)* -- italic        -->
<!ELEMENT sansser             - -  (%p.trans;|#PCDATA)* -- sans serif    -->
<!ELEMENT typewrit            - -  (%p.trans;|#PCDATA)* -- typewriter    -->
<!ELEMENT smallcap            - -  (%p.trans;|#PCDATA)* -- small caps    -->
<!ELEMENT roman               - -  (%p.trans;|#PCDATA)* -- roman         -->


<!-- ===================================================================== -->
<!-- Fractions                                                             -->
<!-- ===================================================================== -->

<!--      ELEMENT             MIN  CONTENT              EXPLANATIONS     -->
<!ELEMENT fraction            - -  (num, den)      -- fraction           -->
<!ELEMENT num                 - -  (%p.trans;|%m.math;)* -- numerator    -->
<!ELEMENT den                 - -  (%p.trans;|%m.math;)* -- denominator  -->
<!--      ELEMENT   NAME      VALUE           DEFAULT                      -->
<!ATTLIST fraction  shape     (built|case) #IMPLIED
                    align     (left|center|right)
                                              center
                    style     (single|double|triple|dash|dot|bold|blank|none)
                                              single                       >



<!-- ===================================================================== -->
<!-- Superiors, inferiors, accents, over and under                         -->
<!-- ===================================================================== -->

<!--      ELEMENT             MIN  CONTENT              EXPLANATIONS     -->
<!ELEMENT sup                 - -  (%p.trans;|%m.math;)* -- superior     -->
<!ELEMENT inf                 - -  (%p.trans;|%m.math;)* -- inferior     -->
<!--      ELEMENT   NAME      VALUE           DEFAULT                      -->
<!ATTLIST sup       location  (pre|post)      post
```

```
                    arrange    (compact|stagger)
                                                compact                      >
<!ATTLIST inf       location   (pre|post) post
                    arrange    (compact|stagger) compact                     >


<!-- ===================================================================== -->
<!-- Embellishments                                                        -->
<!-- ===================================================================== -->

<!--      ELEMENT              MIN  CONTENT               EXPLANATIONS     -->
<!ELEMENT top                  - -  (%p.trans;|%m.math;)*
                                                    -- top embellishment   -->
<!ELEMENT middle               - -  (%p.trans;|%m.math;)*
                                                   -- middle, or "through" -->
<!ELEMENT bottom               - -  (%p.trans;|%m.math;)*
                                                   -- bottom embellishment -->
<!--      ELEMENT   NAME       VALUE            DEFAULT                    -->
<!ATTLIST top       align      (left|center|right)
                                                center
                    sizeid     ID               #IMPLIED
                                                -- to pass on the height   -->
<!ATTLIST middle    align      (left|center|right)
                                                center
                    sizeid     ID               #IMPLIED
                                                -- to pass on the height   -->
<!ATTLIST bottom    align      (left|center|right)
                                                center
                    sizeid    ID               #IMPLIED
                                                -- to pass on the height   -->


<!-- The subform element is defined later                                  -->



<!-- ===================================================================== -->
<!-- Fences, boxes, overlines and underlines                               -->
<!-- ===================================================================== -->

<!--      ELEMENT              MIN  CONTENT               EXPLANATIONS     -->
<!ELEMENT mark                 - O  EMPTY                                    >
<!ELEMENT fence                - -  (%p.trans;|%m.math;)* -- fence         -->
<!ELEMENT post                 - O  EMPTY            -- post               -->
<!ELEMENT box                  - -  (%p.trans;|%m.math;)* -- box           -->
<!ELEMENT overline             - -  (%p.trans;|%m.math;)* -- overline      -->
<!ELEMENT undrline             - -  (%p.trans;|%m.math;)* -- underline     -->
<!--      ELEMENT   NAME       VALUE            DEFAULT                    -->
<!ATTLIST mark      id         ID               #REQUIRED                    >
<!ATTLIST fence     lpost      CDATA            "|" -- left post          --
                    rpost      CDATA            "|" -- right post         --
                    style      (single|double|triple|dash|dot|bold|blank|none)
                                                single
                    sizeid     ID               #IMPLIED
                                                -- to pass on the height   --
                    sizeref    IDREF            #IMPLIED
                                                -- to pick up a height     -->
<!ATTLIST post      post       CDATA            "|"
                    style      (single|double|triple|dash|dot|bold|blank|none)
                                                single
                    sizeid     ID               #IMPLIED
                                                -- to pass on the height   --
                    sizeref    IDREF            #IMPLIED
                                                -- to pick up a height     -->
<!ATTLIST box       style      (single|double|triple|dash|dot|bold|blank|none)
                                                single                       >
<!ATTLIST overline  type       CDATA            "-" -- embellishment type --
                    style      (single|double|triple|dash|dot|bold|blank|none)
                                                single
                    start      IDREF            #IMPLIED
                    end        IDREF            #IMPLIED                     >

<!ATTLIST undrline  type       CDATA            "_" -- embellishment
                                                                       type --
                    style      (single|double|triple|dash|dot|bold|blank|none)
                                                single
                    start      IDREF            #IMPLIED
                    end        IDREF            #IMPLIED                     >
```

```
<!-- ===================================================================== -->
<!-- Labelled arrows                                                       -->
<!-- ===================================================================== -->

<!--      ELEMENT             MIN  CONTENT              EXPLANATIONS     -->
<!ELEMENT subform             - -  (%p.trans;|%m.math;)* -- base element  -->
<!--      ELEMENT   NAME       VALUE            DEFAULT                    -->
<!ATTLIST subform   sizeid     ID               #IMPLIED
                                                -- to pass on a width, or
                                                a height                   --
                    sizeref    IDREF            #IMPLIED
                                                -- to pick up a width      -->


<!-- ===================================================================== -->
<!-- Roots                                                                 -->
<!-- ===================================================================== -->

<!--      ELEMENT             MIN  CONTENT              EXPLANATIONS     -->
<!ELEMENT radical             - -  (radix?, radicand) -- root or radical  -->
<!ELEMENT radix               - -  (%p.trans;|%m.math;)* -- radix         -->
<!ELEMENT radicand            O O  (%p.trans;|%m.math;)* -- radicand      -->


<!-- ===================================================================== -->
<!-- Arrays                                                                -->
<!-- ===================================================================== -->

<!--      ELEMENT             MIN  CONTENT              EXPLANATIONS     -->
<!ELEMENT array               - -  (arrayrow+|arraycol+) -- array         -->
<!ELEMENT arrayrow            - O  (arraycel+)      -- array row          -->
<!ELEMENT arraycol            - O  (arraycel+)      -- array column       -->
<!ELEMENT arraycel            - O  (%p.trans;|%m.math;)* -- array cell    -->

<!--      ELEMENT   NAME       VALUE            DEFAULT                    -->
<!ATTLIST array     rowalign   NMTOKENS         #IMPLIED -- row alignment  --
                    colalign   NMTOKENS         #IMPLIED -- column
                                                                 alignment --
                    rowsep     NMTOKENS         #IMPLIED -- row separators --
                    colsep     NMTOKENS         #IMPLIED -- column
                                                                separators -->


<!-- ===================================================================== -->
<!-- Spacing                                                               -->
<!-- ===================================================================== -->

<!--      ELEMENT             MIN  CONTENT              EXPLANATIONS     -->
<!ELEMENT hspace              - O EMPTY            -- horizontal spacing  -->
<!ELEMENT vspace              - O EMPTY            -- vertical   spacing  -->
<!ELEMENT break               - O EMPTY            -- turn line, break    -->
<!ELEMENT markref             - O EMPTY            -- hmark reference     -->

<!--      ELEMENT   NAME       VALUE            DEFAULT                    -->
<!ATTLIST hspace    space      CDATA            "1 mm"
                                                    -- units as required   -->
<!ATTLIST vspace    space      CDATA            "1 mm"
                                                    -- units as required   -->
<!ATTLIST markref   refid      IDREF            #REQUIRED
                    direct     (hor|ver)        hor
                                                 -- horizontal or vertical -->


<!-- ===================================================================== -->
<!-- the formula elements                                                  -->
<!-- ===================================================================== -->

<!--      ELEMENT             MIN  CONTENT              EXPLANATIONS     -->
<!ELEMENT formula             - -  (%p.trans;|%m.math;)*
                                                        -- in-line formula -->
<!ELEMENT dformula            - -  (%p.trans;|%m.math;)*
                                                        -- display formula -->
<!ELEMENT dformgrp            - -  (formula|dformula)+
                                                  -- display-formula group -->

<!--      ELEMENT   NAME       VALUE            DEFAULT                    -->
<!ATTLIST formula   id         ID               #IMPLIED
                    alphabet   %a.types;
--                  %SDAPREF;      "<?SDATRANS>Inline formula"    --
--                  %SDASUSP;       "SUSPEND"        --
```

```
>
<!ATTLIST dformula  id         ID              #IMPLIED
                    num        CDATA           #IMPLIED
                    align      (left|center|right)
                                               center
                    alphabet   %a.types;
--                  %SDAPREF;      "<?SDATRANS>Display formula" --
--                  %SDASUSP;       "SUSPEND"        --
>
<!ATTLIST dformgrp  id         ID              #IMPLIED
                    num        CDATA           #IMPLIED
                    align      (left|center|right)
                                               center
--                  %SDAPREF;      "<?SDATRANS>Display formula group" --

>
```

H Example of a Conversion of the DocBook DTD to HTML3

H.1 The original document marked up in the Doc-Book DTD

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

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

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

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

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

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

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

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

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

H.2 ESIS representation of the source document

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

```
AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
ALABEL IMPLIED
ARENDERAS IMPLIED

(SECT1

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
APAGENUM IMPLIED

(TITLE

-How to Get the DocBook DTD
Online

)TITLE

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED

(PARA

-You can find the DocBook DTD
and its documentation \nonline
in the Davenport archive \n(

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(FILENAME

-/pub/davenport/docbook

)FILENAME

-) at \n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(FILENAME

-ftp.ora.com

)FILENAME

- (198.112.208.13).

)PARA

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED

(PARA

-This sample session shows how
to retrieve the DTD\nand its
documentation:\n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
```

```
sline ends and leading white
space must be preserved in
output

NLINESPECIFIC

AFORMAT NOTATION LINESPECIFIC
AWIDTH IMPLIED

(SCREEN

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
ACLASS TOKEN PROMPT
AMOREINFO TOKEN NONE

(SYSTEMITEM

-%

)SYSTEMITEM

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(USERINPUT

-ftp ftp.ora.com

)USERINPUT

-\n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(COMPUTEROUTPUT

-Connected to amber.ora.com.

)COMPUTEROUTPUT

-\n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(COMPUTEROUTPUT

-220 amber FTP server (Version
wu-2.4(1) Fri Apr 15 14:14:30
EDT 1994) ready.

)COMPUTEROUTPUT

-\n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(COMPUTEROUTPUT

-Name (ftp.ora.com:terry):

)COMPUTEROUTPUT

AID IMPLIED
```

```
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(USERINPUT

-anonymous

)USERINPUT

-\n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(COMPUTEROUTPUT

-331 Guest login ok, send your
complete e-mail address as
password.

)COMPUTEROUTPUT

-\n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(COMPUTEROUTPUT

-Password:

)COMPUTEROUTPUT

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED

(LINEANNOTATION

-\|[larr  ]\| type e-mail
address

)LINEANNOTATION

-\n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
ACLASS TOKEN PROMPT
AMOREINFO TOKEN NONE

(SYSTEMITEM

-ftp\|[gt    ]\|

)SYSTEMITEM

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(USERINPUT

-cd pub/davenport/docbook

)USERINPUT
```

```
)SCREEN

-\nThe DocBook DTD and related
ASCII files are in\na file
named

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(FILENAME

-docbook.N.shar

)FILENAME

-, where

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED

(EMPHASIS

-N

)EMPHASIS

-\nis the current revision
number:\n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AFORMAT NOTATION LINESPECIFIC
AWIDTH IMPLIED

(SCREEN

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
ACLASS TOKEN PROMPT
AMOREINFO TOKEN NONE

(SYSTEMITEM

-ftp\|[gt    ]\|

)SYSTEMITEM

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(USERINPUT

-get docbook.2.2.1.shar

)USERINPUT

)SCREEN

-\nMost of these files\nalso
exist separately and may be
ftp'd individually.

)PARA

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
```

```
(PARA

-The

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(COMMAND

-get

)COMMAND

- command will put this ASCII
shar \nfile on your system.
You must later unpack \nit on

your system:\n

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AFORMAT NOTATION LINESPECIFIC
AWIDTH IMPLIED

(SCREEN

AID IMPLIED
ALANG IMPLIED
AREMAP IMPLIED
AROLE IMPLIED
AXREFLABEL IMPLIED
AMOREINFO TOKEN NONE

(USERINPUT

-sh docbook.2.2.1.shar

)USERINPUT

)SCREEN

)PARA
```

H.3 HTML3 output

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

```
<HTML>
<HEAD>
<TITLE>How to Get the DocBook DTD Online</TITLE>
</HEAD>
<BODY>
<H1>How to Get the DocBook DTD Online</H1>

You can find the DocBook DTD and its documentation online in the
Davenport archive (/pub/davenport/docbook) at ftp.ora.com
(198.112.208.13).<P>This sample session shows how to retrieve
the DTD and its documentation:

<pre>
%<i>ftp ftp.ora.com</i>
Connected to amber.ora.com.
220 amber FTP server (Version wu-2.4(1) Fri Apr 15 14:14:30 EDT 1994) ready.
Name (ftp.ora.com:terry): <i>anonymous</i>
331 Guest login ok, send your complete e-mail address as password.
Password:  type e-mail address
ftp&gt;<i>cd pub/davenport/docbook</i>
</pre>

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

<pre>
ftp&gt;<i>get docbook.2.2.1.shar</i>
</pre>

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

<pre>
<i>sh docbook.2.2.1.shar</i>
</pre>
</BODY>
</HTML>
```

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 2$_\varepsilon$ because it uses undocumented LATEX internals. (*Tanmoy Bhattacharya*)

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

hyper.dtx Similar to hyperlatex, but for LATEX 2$_\varepsilon$ (*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`

LaTeX, hypertext and PDF, *or* the entry of TeX into the world of hypertext[*]

Yannis Haralambous and Sebastian Rahtz

1 The relationship between hypertext and LaTeX

Unlike *hyper*market, *hyper*tension and *hyper*-activity, where the prefix *hyper-* expresses high quantity, and excess, *hyper*text is not a giant text, but a text with an internal structure that viewers can exploit to allow for non-linear navigation through the document.

[*] This paper is based on one published by Yannis in *Cahiers GUTenberg* no. 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.

There is thus a relationship between the notion of hypertext and the mark-up system of LaTeX: both add structure to a document. For example, the LaTeX 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 TeX in screen interaction. TeX 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. Since TeX is a tool for typographical composition, it does not use a screen other than for previewing, and screen display is seldom considered the final aim of a TeX compilation.

This lack of interest in TeX 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.[1]

We claim that, for effectively the first time in its existence, TeX is becoming seriously useful for creating documents whose aim is to be read on the screen. In fact, LaTeX 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 LaTeX document into an hypertext document. It is worth insisting that such a document keeps all the typographical quality of LaTeX, and can be printed exactly in the same way as before.

2 Overview

TeX and 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 visualisation 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 TeX community, and one would be effectively limited to CM fonts, which are the only fonts (almost) guaranteed to be present on all 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 TeX system.

Finally, hypertext links are not catered for in the syntax of the DVI format;[2] 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 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 (visualisation, 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). TeX will produce a `.dvi` file (it also uses macro files, and font metrics). If the LaTeX format is used, and the *hyperref* package has been loaded (as the last package), the cross reference commands, bibliographic citations and indexing will produce hypertext links, included in the `.dvi` file with the help of `\special` commands.[3]

Another possible starting point is an HTML file (in the top left of the diagram). An HTML file can be easily converted to LaTeX, and the hypertext links in the former can be kept in the latter.

[1] Except for the lucky ones amongst us who can use operating systems with Display PostScript.

[2] The description of the DVI format can be found on CTAN in the directory CTANdviware/driv-standard/level-0/dvistd0.tex

[3] 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.

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. Either an extended version of `dvips` by Mark Doyle, called `dvihps` (DVI to HyperPostScript), keeps the hypertext links contained in the `.dvi` file, or we can write Acrobat `pdfmark` code directly. 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 visualise a PDF document we can use Adobe Acrobat Reader, which is freely available for Macintosh, Windows, DOS and some Unix platforms. This program allows us to browse and navigate the document and print it on any printer supported by the host system. Recent versions of the free PostScript interpreter, GhostScript (later than 3.51) are also able to display and print PDF files.

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 LaTeX. A PDF document is a faithful copy of the printed document (it can even be phototypeset to produce a professional quality result with colour 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 LaTeX document can be exploited by a wide application domain: the most striking example is the voice 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 LaTeX

The HTML mark-up 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 LaTeX is a much richer language than HTML, and so the conversion from HTML to LaTeX is essentially trivial.[4] The conversion has to do some simple jobs:

1. convert certain tags straight to a LaTeX environments, such as `<CITE>` and `</CITE>` going to `\begin{quotation}` and `\end{quotation}`;
2. convert other tags to LaTeX commands with arguments, such as `<em>` and `</em>` going to `\emph{...}`;
3. replace a very few tags with simple 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 LaTeX, such as `<STRONG>`; the appropriate action for these is to convert them to new LaTeX environments, and provide appropriate definitions in a style file. Thus

   ```
   <strong>Very important!</strong>
   ```

   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 LaTeX by `\hyperdef{}{keyword}{}{...}`, where ... is the chosen text.

[4] Going in the other direction is much harder (see [2])!

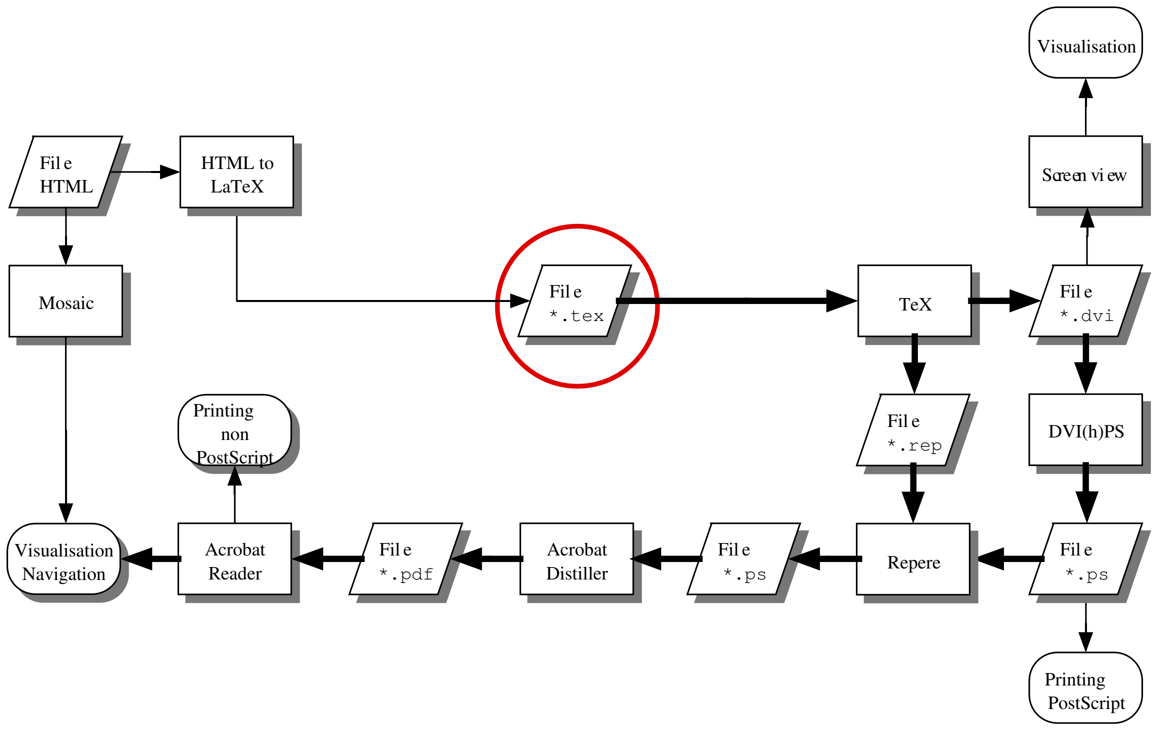


Figure 1: Flow diagram for processing hypertext LaTeX files

(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 LaTeX 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 LaTeX mark-up would be `\hyperref{address}{}{}{...}`.

(d) Linking to an image, which in HTML would be `<IMG SRC="address">`; in LaTeX this is converted into `\hyperimage{address}`

4 LaTeX to DVI

Let us be clear from the start: more or less any valid LaTeX 2_ε document can produce a electronic equivalent, by the simple addition of

```
\usepackage{hyperref}
```

at the *end* of the document preamble (this is very important, to give the package a fighting chance of being the last one to redefine underlying macros). This loads Sebastian Rahtz' hyperref package, which redefines the following LaTeX 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 LaTeX mark-up, in which case the "raw" commands `\hypertarget` and `\hyperlink` and `\hyperimage` can be used.

4.1 The HyperTeX specification and the hyperref package

The hyperref package derives from and builds on the work of the HyperTeX project, described in the World Wide Web document `http://xxx.lanl.gov/hypertex/`. It aims to extend the functionality of all the LaTeX cross-referencing commands (including the table of contents) to produce `\special` commands which are parsed by DVI processors conforming to the HyperTeX guidelines (i.e., `xhdvi` and `dvihps`); it also provides general hypertext links, including those to external documents.

The HyperTeX specification[5] 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 TeX file—the TeX 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 LaTeX 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 AMSLaTeX can cause problems.

[5] This description is derived from Arthur Smith's documentation.

The package supports the following options:

draft makes the low-level macros have no effect;

colorlinks colours the links and anchors (this needs the standard LaTeX 2$_\varepsilon$ color package). The colors can be changed by redefining two macros; the default setting is:

```
\def\LinkColor{red}
\def\AnchorColor{blue}
```

nocolorlinks turns off colouring, 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;

nativepdf does not emit standard HyperTeX `\special` commands, but produces `pdfmark` code directly as raw PostScript;

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 behaviour, 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 behaviour;

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, Haralambous developed a post-processor for the output of `dvihps` which creates the necessary extra material. The program, *repere*, is written in Flex, and can be compiled on most platforms with a Flex implementation and a C compiler.

The *repere* 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 LaTeX, and running `dvihps`, the `.rep` file is prepended and appended to the PostScript file, and the result run through *repere*. For a file `foo.tex`, the sequence would be (for Unix, or other systems with pipes:[6])

```
latex foo
latex foo
dvihps -z foo -o footemp.ps
cat foo.rep footemp.ps foo.rep \
  | repere > foo.ps
```

This would result in a PostScript file, `foo.ps` which can be given to Adobe Distiller which will produce the table of contents. The *repere* program works by writing `pdfmark` commands for Distiller.

The trickiest part of the operation is the conversion of the encoding of the 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 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 *repere* 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 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. The vast majority 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?

[6] DOS or VMS users will have to use copy/append commands to create a temporary file.

Adobe solved this problem with the *Multiple Master* technology; this is similar to the principles of METAFONT,[7] 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 embedded 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 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 TeX, fonts can be included in PK bitmap format. The copyright problem does not arise, since only bitmap representations are included in the PDF file.[8] Unfortunately, Acrobat Reader does not display such bitmap fonts

[7] Compared to METAFONT, Multiple Master fonts are in fact quite simplistic.

[8] 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.

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 LaTeX 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 'écouté' is represented as

```
e<acute accent>coute<acute accent>
```

in the PDF document, then a search for *écouté*, where é is an 8-bit character, will not be successful.

The solution to this problem is to use PostScript fonts encoded in the LaTeX 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 ź 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 pointing out that LaTeX 2_ε will automatically transform 7-bit input mark-up like `\'e` into the 8-bit single character on output, if T1 encoding is used.

5 DVI to (hyper)PostScript

Like TeX, `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.[9]

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 LaTeX 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 LaTeX 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.

If we know we *only* want PDF, rather than having a portable hypertext `dvi` file, it is probably easier to use the `nativepdf` option of hyperref. This produces simple `pdfmark` PostScript, with a greater degree of flexibility than `dvihps` currently offers. The user has full control over all the parameters of the `pdfmark`, allowing:

- changing of the color and style of frames around active areas;
- varying the type of link (i.e., to full page, zooms etc.);

[9] It is to be expected that the functionality will be merged back into the 'real' `dvips` by Tom Rokicki in due course.

- access to the trivial commands like 'Next Page' and 'Previous Page' without resorting to LaTeX macro programming;
- setting startup code (i.e., to start document in full-screen mode);
- setting window title etc.

With the release of Acrobat 2.1 in September 1995, the `dvihps` program and the native `pdfmark` feature in hyperref have been enhanced to support the 'plug-in' which allows for access to World Wide Web browsers from within Acrobat. This allows the author to define URLs in the text as Web links, and have them launch a browser when activated. Distiller 2.1 is required to understand the appropriate `pdfmark` commands.

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 useful way to preview any PostScript file, although the processing is rather slow. There is another way of creating PDF, the PDFwriter printer driver which is part of Acrobat Exchange; this allows users of any conventional Windows or Mac word-processing program to 'print' directly from their application to a PDF file. However, this has no possibilities for automatic creation of hypertext links via `pdfmark`, so we do not consider it very useful in the current context.

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 or key combinations. Figure 2 shows a typical Acrobat screen, with page thumbnails on the left side for help in navigation, and marked hyperlink areas in the main text The LaTeX 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). Figure 3 shows how hyperref displays the table of contents, with each line as an active area, framed in a black box; Figure 4 shows a different style of marking areas, this time in the bibliography. In the latter case the links are 'back references' to where the reference is cited in the text. These are derived automatically by the hyperref package.

7.1 Some information on the PDF format

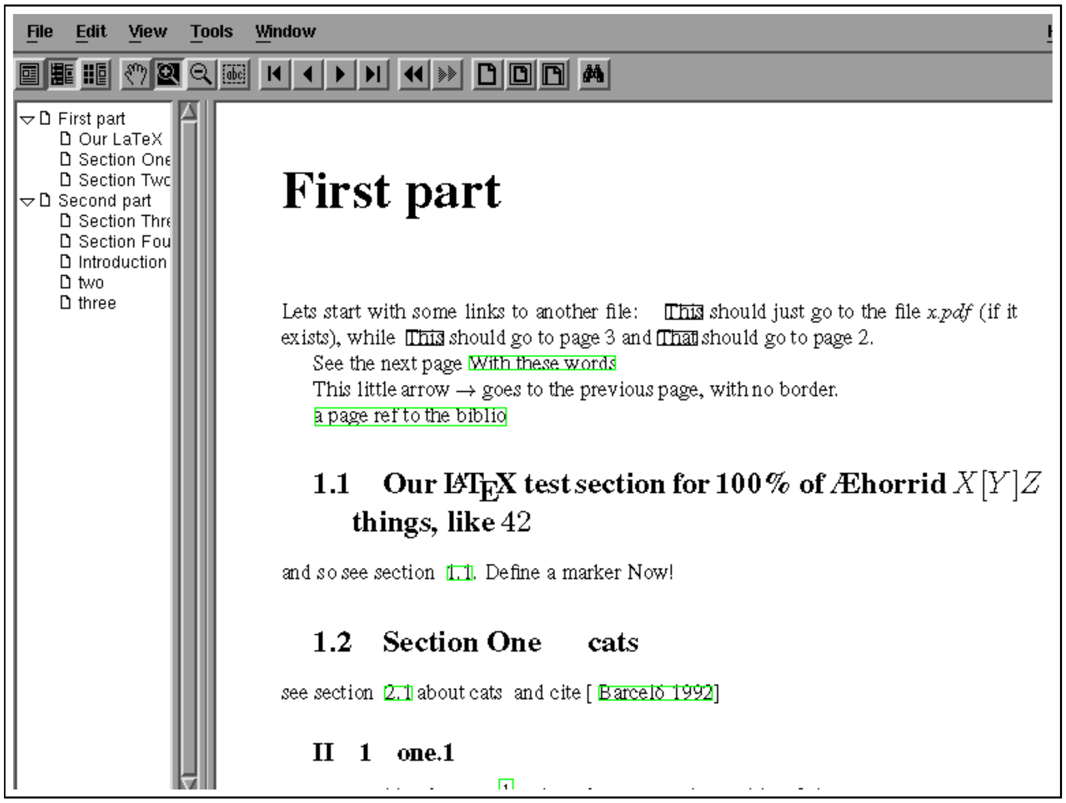
The PDF format is as difficult (or easy!) for average user to read as the PostScript language. However, it is interesting to know a bit of its structure, to perform, if needed, minor modifications to the presentation file (the PDF format is still quite new and we desperately lack tools to modify PDF 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 body is composed of objects: each page is an object; the links, the notes, the marks, the font codes, the font descriptors, and the systems for colour 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. 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.

We will describe here only some of the 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, Figure 5 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.



First part

Lets start with some links to another file: This should just go to the file *x.pdf* (if it exists), while This should go to page 3 and That should go to page 2.

See the next page With these words

This little arrow → goes to the previous page, with no border.

a page ref to the biblio

1.1 Our LaTeX test section for 100% of Æhorrid $X[Y]Z$ things, like 42

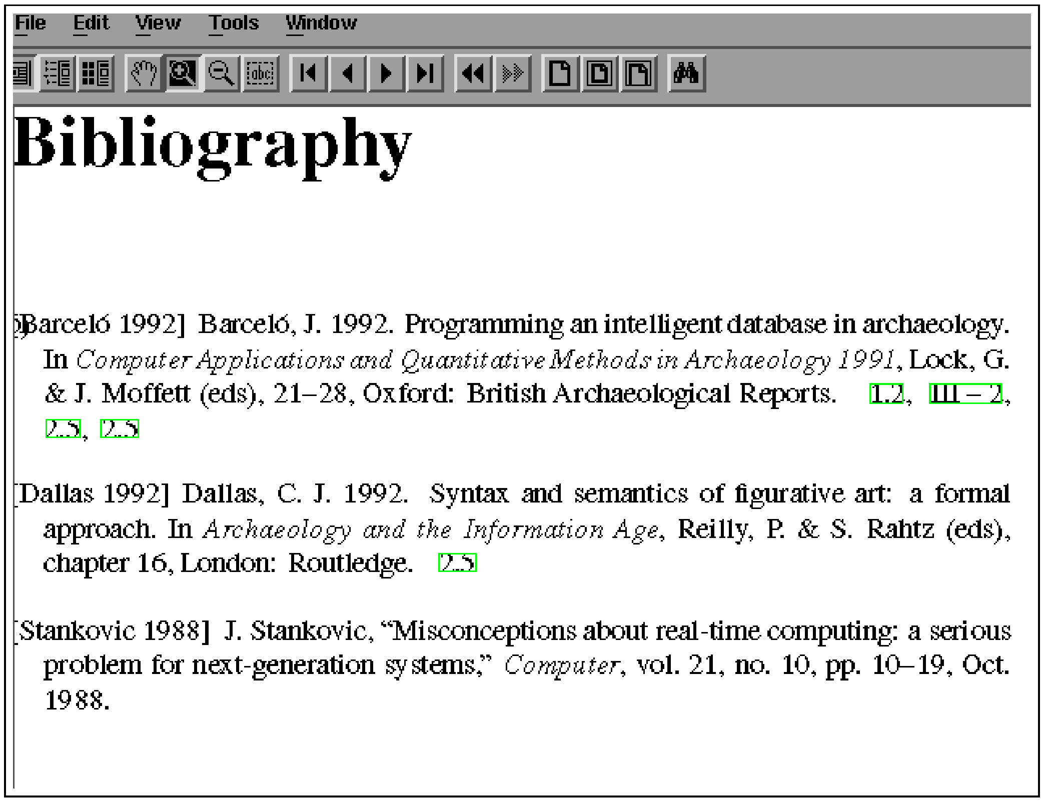
and so see section 1.1. Define a marker Now!

1.2 Section One cats

see section 2.1 about cats and cite [Barceló 1992]

II 1 one.1

Figure 2: PDF file being displayed with Acrobat Reader



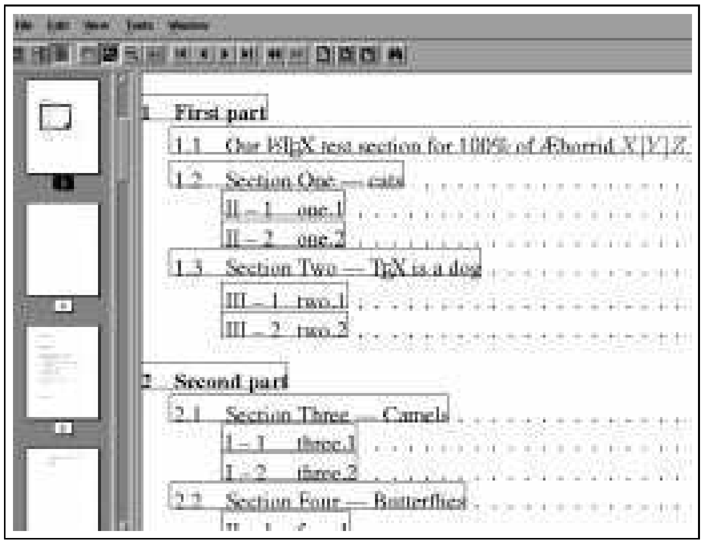
Bibliography

[Barceló 1992] Barceló, J. 1992. Programming an intelligent database in archaeology. In *Computer Applications and Quantitative Methods in Archaeology 1991*, Lock, G. & J. Moffett (eds), 21–28, Oxford: British Archaeological Reports. 1.2, III – 2, 2.5, 2.5

[Dallas 1992] Dallas, C. J. 1992. Syntax and semantics of figurative art: a formal approach. In *Archaeology and the Information Age*, Reilly, P. & S. Rahtz (eds), chapter 16, London: Routledge. 2.5

[Stankovic 1988] J. Stankovic, "Misconceptions about real-time computing: a serious problem for next-generation systems," *Computer*, vol. 21, no. 10, pp. 10–19, Oct. 1988.

Figure 3: The LaTeX table of contents in Acrobat



1 First part
1.1 One LaTeX test section for 100% of Æborrid X|Y|Z
1.2 Section One — cats
II – 1 one.1
II – 2 one.2
1.3 Section Two — TeX is a dog
III – 1 two.1
III – 2 two.2
2 Second part
2.1 Section Three — Camels
I – 1 three.1
I – 2 three.2
2.2 Section Four — Butterflies

Figure 4: Bibliography, showing back-referencing

The number 251984 is the offset from the start of the file of the beginning of the cross-reference table. The extract in Figure 6 shows an object in the main part of the file with the uncompressed version of some text being displayed

A hypertext link is an object of type 'Annot'; an example is

```
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 coloured 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 favourite 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. 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 LaTeX 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

```
%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
0000021336 00000 n
......
0000211955 00000 n
0000251877 00000 n
trailer
<<
/Size 43
/Root 41 0 R
/Info 42 0 R
/ID [<a7b776d0fb5478b29f5739c089a2c83f><a7b776d0fb5478b29f5739c089a2c83f>]
>>
startxref
251984
%%EOF
```

Figure 5: Extract from a PDF file

```
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
...
```

Figure 6: A PDF object

and richer electronic documents, using TEX typography as a solid foundation.

Obtaining the programs

The HyperTEX project, whose standards form the basis of the work described in this article, should be visited on the World Wide Web at `http://xxx.lanl.gov/hypertex/`.

The hyperref package can be obtained from any of the CTAN (Comprehensive TEX Archive Network) archives, from the directory CTANmacros/latex/contrib/supported/hyperref. The *repere* and *recticrt* programs are supplied in source form (Flex code) and as compiled MSDOS binaries. The source of `dvihps` is available in CTANdviware/dvihps in the CTAN archives, and an MSDOS binary is also stored in the `hyperref` directory. Michael Mehlich has written another LATEX 2$_\varepsilon$ package for encapsulating hypertext functionality in LATEX output, to the same HyperTEX standards as hyperref, with comparable functionality. This is available on CTAN in CTANmacros/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 CTANfonts/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.

References

[1] Tim Bientz and Richard Cohn (Adobe Systems Inc.). *Portable Document Format Reference Manual*, Addison Wesley 1993.

[2] Michel Goossens and Janne Saarela. *From LATEX to HTML and back*, *TUGboat*, 16(3), 1995.

[3] T.V. Raman. *An audio view of TEX documents*, *TUGboat*, 13(4), 1992.

◇ Yannis Haralambous
187, rue Nationale, 59000 Lille, France
Email: `haralambous@univ-lille1.fr`

◇ Sebastian Rahtz
Elsevier Science Ltd, The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB
Email: `s.rahtz@elsevier.co.uk`

From LaTeX to HTML and back

Michel Goossens and Janne Saarela

Abstract

Both LaTeX and HTML are languages that can express the structure of a document, and similarities between these two systems are shown. A detailed study is made of the LaTeX2HTML program, written by Nikos Drakos, that is today the most complete utility for translating LaTeX code into HTML, providing a quasi-automatic translation for most elements. A discussion of a few other tools for translating between HTML and LaTeX concludes the article.

1 Similarities between LaTeX and HTML

HTML and LaTeX are both generic markup systems, and a comparison between tags for structural elements in both cases is shown in Table 1. In most cases the differences are trivial, seeming to indicate that, at first approximation, translating between these two systems should not prove too difficult.

The translation programs described in this article use these similarities, but in order to exploit the richness of the LaTeX language as compared to HTML (especially HTML2, which has no support for tables or mathematics), an *ad hoc* approach has to be adopted. To handle correctly LaTeX commands that have no equivalent in HTML, such elements can either be transformed into bitmap or PostScript pictures (an approach taken by `LaTeX2HTML`), or the user can specify how the given element should be handled in the target language.

2 Converting LaTeX into HTML

Before discussing the `LaTeX2HTML` program, we want to mention a few other programs. First there is `l2x`,[1] written by Henning Schulzrinne (Berlin, Germany), which translates LaTeX into various other formats. This program is written in C and calls a Tcl function (Ousterhout, 1994) for each LaTeX command.

A converter `html.tcl` is available for translating LaTeX files into HTML, by writing, for instance:

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

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

Schwarzkopf (Schwarzkopf, 1995) has developed `Hyperlatex`,[2] a package written in the GNU Emacs lisp language to translate documents marked up in (a subset of) LaTeX into HTML.

Another interesting tool is `tex2RTF`,[3] a utility to convert from LaTeX to four other formats, including HTML. It does a relatively good job for a sub-set of LaTeX commands, but, as with the Tcl approach of `l2x`, it cannot handle more complex structures, such as mathematical expressions and tables.

Finally, although not directly relevant to LaTeX, `texihtml`[4] translates `texinfo` sources[5] into HTML.

3 The LaTeX2HTML Converter—Generalities

`LaTeX2HTML` is a program written in the `perl` programming language[6] (Schwartz, 1993; Till, 1995; Wall and Schwartz, 1991) by Nikos Drakos.[7]. It transforms a LaTeX document into a series of HTML files linked in a way that reflects the structure of the original document.

3.1 What LaTeX2HTML is and What it is Not

`LaTeX2HTML` is a conversion tool that allows documents written in LaTeX to become part of the World Wide Web. In addition, it offers an easy migration path towards authoring complex hypermedia documents using familiar word-processing concepts.

`LaTeX2HTML` replicates the basic structure of a LaTeX document as a set of interconnected HTML files which can be explored using automatically generated navigation panels. The cross-references, citations, footnotes, the table of contents and the lists of figures and tables are also translated into hypertext links. Formatting information which has equivalent "tags" in HTML (lists, quotes, paragraph breaks, type styles, etc.) is also converted appropriately. The remaining heavily formatted items such as mathematical equations, pictures or tables are

[1] See the URL `http://info.cern.ch/hypertext/WWW/Tools/l2x.html`.

[2] The documentation is available at the URL `http://hobak.postech.ac.kr/~otfried/html/hyperlatex.html`. Otfried Schwarzkopf can be reached via email at `otfried@vision.postech.ac.kr`.

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

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

[5] `texinfo` is a TeX based markup language used for all gnu project related documentation.

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

[7] The documentation is at the URL `http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html`. One can also join the `LaTeX2HTML` mailing list by sending a message to `latex2html-request@mcs.anl.gov` with as only contents line: `subscribe`.

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

Table 1: Comparison of structural elements in HTML and LaTeX

converted to images placed automatically at the correct positions in the final HTML document.

LaTeX2HTML extends LaTeX by supporting arbitrary hypertext links and symbolic cross-references between evolving remote documents. It also allows the specification of *conditional text* and the inclusion of raw HTML commands. These hypermedia extensions to LaTeX are available as new commands and environments from within a LaTeX document.

3.2 Overview

The main characteristics of the LaTeX2HTML translator are summarized in this section.

- a document is broken into one or more components as specified by the user;
- optional, customizable navigation panels can be added to each generated page to link other parts of the document, or external documents;
- inline and displayed equations are handled as images;
- tables and figures, and all other arbitrary environments are passed on to LaTeX and included as images; these images are included inline or made available via hypertext links;
- figures or tables can be arbitrarily scaled and shown either as inlined images or "thumbnails";
- output can be generated to cope with the possibilities of various kind of browsers (for example, line browsers);
- definitions of new commands, environments, and theorems are given even when they are in external files;
- footnotes, tables of contents, lists of figures and tables, bibliographies, and the index are handled correctly;
- LaTeX cross-references and citations are transformed into hyperlinks, which work just as well inside a (sub)document as between several documents (located anywhere on the Internet);
- most LaTeX accented national characters are translated into their ISO-Latin-1 equivalent;
- hypertext links to arbitrary Internet services are recognized;
- programs running arbitrary scripts can be invoked (at the LaTeX level);
- a conditional text mechanism allows material to be included in the HTML or printed (DVI) versions only;
- similarly raw HTML material can be present in the LaTeX document (such as for specifying interactive forms);
- *common* LaTeX commands (i.e.,, those defined in the LaTeX Reference manual (Lamport, 1994)) are handled gracefully;

- the user can define (in `perl`) functions to translate (un)known LaTeX commands in a customized way.

3.3 Using `LaTeX2HTML`

To use `LaTeX2HTML`, simply type

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

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

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

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

`-split` *num* The default is 8.
: Stop splitting sections into separate files at this depth. A value of 0 will put the document into a single HTML file.

`-link` *num* The default is 4.
: Stop revealing child nodes at each node at this depth. (A node is characterized by the sequence part-chapter-section-subsection-...). A value of 0 will show no links to child nodes, a value of 1 will show only the immediate descendents, etc. A value at least as big as that of the `-split` option will produce a table of contents for the tree structure, rooted at each given node.

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

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

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

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

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

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

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

`-no_navigation`
: Do not put navigation links in each page.

`-top_navigation`
: Put navigation links at the top of each page.

`-bottom_navigation`
: Put navigation links at the bottom of each page *as well* as at the top.

`-auto_navigation`
: Put navigation links at the top of each page. If the page exceeds `$WORDS_IN_PAGE` number of words (the default is 450) then put one at the bottom of the page also.

`-index_in_navigation`
: When an index exists, put a link to the index page in the navigation panel.

`-contents_in_navigation`
: When a table of contents exists, put a link to that table in the navigation panel.

`-next_page_in_navigation`
: Put a link to the next logical page in the navigation panel.

`-previous_page_in_navigation`
: Put a link to the previous logical page in the navigation panel.

`-info` *string*
: Generate a new section *About this document ...* containing information about the document being translated. The default is for generating such a section with information on the original document, the date, the user and the translator. If *string* is empty (or has the value 0), this section is not created. If *string* is non-empty, it will replace the default information in the contents of the *About this document ...* section.

`-dont_include` *file1 file2 ...*
: Do not include the specified file(s) *file1*, *file2*, etc. Such files can be package files that contain raw TeX commands that the translator cannot handle.

`-reuse`
: Images generated during a previous translation process should be reused as far as possible. With this option enabled the user is no longer asked whether to reuse the old directory, delete its contents or quit. Images which depend on context (for example, numbered tables or equations) cannot be reused and are always regenerated.

`-no_reuse`
: Do *not* reuse images generated during a previous translation. This enables the initial interactive session during which the user is asked whether to reuse the old directory, delete its contents or quit.

`-init_file` *file*
: Load the `perl` initialization script *file*. It will be loaded after `$HOME/.latex2html-init` (if it exists). It can be used to change default options.

`-no_images`
: Do not produce inlined images. If needed, the missing images can be generated "off-line" by restarting `LaTeX2HTML` with the `-images_only` option.

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

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

`-h` Print the list of options.

3.4 Simple Use of `LaTeX2HTML`

To show the procedure for translating a LaTeX document into HTML, let us first look at a simple example, namely the file shown in Figure 1.[8] After running this file through LaTeX (twice, to resolve the cross-references) one obtains the output shown in Figure 2.

This same LaTeX source document is now run through `LaTeX2HTML` with the command

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

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

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

as explained in Section 3.9.

The log messages generated by `LaTeX2HTML` are shown below.

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

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

Loading /usr/local/lib/latex2html/styles/makeidx.perl
....
Reading ...
Processing macros ....+.....
Reading babel.aux ..............................
Translating ...0/8.............1/8....2/8....3/8
....4/8............5/8...........6/8..........
7/8.....8/8.....
Writing image file ...

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

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

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

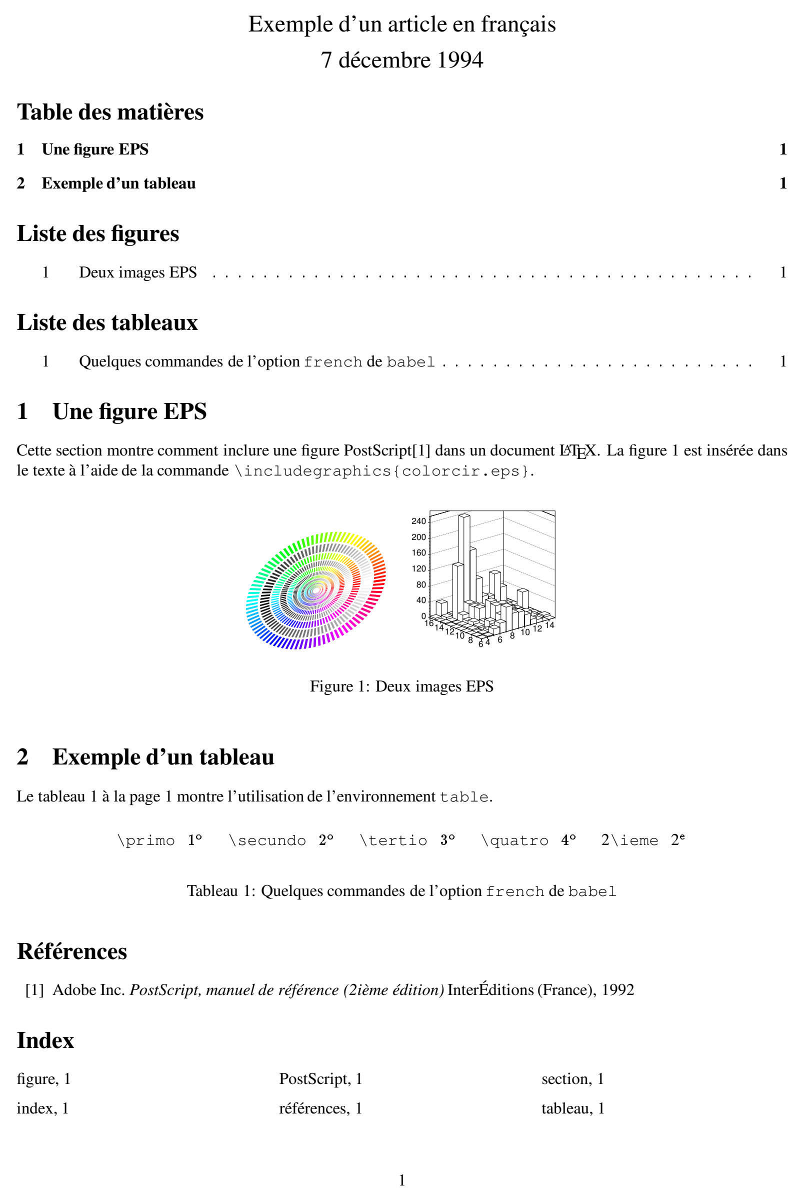
The results are shown in Figure 3. The main document is shown in the middle at the top. Numbered arrows indicate the secondary documents that are produced and to which point in the main document they are linked. The document also contains a table of contents that is not shown explicitly, since its contents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

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

[8] This one-page example is chosen because it is discussed in detail in Chapter 9 of Goossens et al. (1994) and at the same time shows how `LaTeX2HTML` handles non-English documents.

```
\documentclass{article}
\usepackage{makeidx}
\usepackage[dvips]{graphicx}
\usepackage[french]{babel}
\makeindex
\begin{document}
\begin{center}\Large
  Exemple d'un article en fran\c{c}ais\\[2mm]\today
\end{center}
\tableofcontents
\listoffigures
\listoftables
\section{Une figure EPS}
\index{section}
Cette section montre comment inclure une figure PostScript\cite{bib-PS}
dans un document \LaTeX. La figure~\ref{Fpsfig} est ins\'er\'ee dans le
texte \`a l'aide de la commande \verb!\includegraphics{colorcir.eps}!.
\index{figure}\index{PostScript}
\begin{figure}
\centering
  \begin{tabular}{c@{\qquad}c}
    \includegraphics[width=3cm]{colorcir.eps} &
    \includegraphics[width=3cm]{tac2dim.eps}
  \end{tabular}
  \caption{Deux images EPS}\label{Fpsfig}
\end{figure}
\section{Exemple d'un tableau}
Le tableau~\ref{tab:exa} \`a la page \pageref{tab:exa}
montre l'utilisation de l'environnement \texttt{table}.
\begin{table}
\centering
  \begin{tabular}{cccccc}
    \Lcs{primo} \primo & \Lcs{secundo} \secundo & \Lcs{tertio}
    \tertio & \Lcs{quatro} \quatro & 2\Lcs{ieme}\ 2\ieme
  \end{tabular}
  \caption{Quelques commandes de l'option \texttt{french}
           de \texttt{babel}}\label{tab:exa}\index{tableau}
\end{table}
\begin{thebibliography}{99}
\index{r\'ef\'erences}
\bibitem{bib-PS}
Adobe Inc. \emph{PostScript, manuel de r\'ef\'erence
(2i\`eme \'edition)} Inter\'Editions (France), 1992
\end{thebibliography}
\printindex
\index{index}
\end{document}
```

Figure 1: Example of a LaTeX document



Exemple d'un article en français

7 décembre 1994

Table des matières

Liste des figures

Liste des tableaux

1 Une figure EPS

Cette section montre comment inclure une figure PostScript[1] dans un document LaTeX. La figure 1 est insérée dans le texte à l'aide de la commande `\includegraphics{colorcir.eps}`.

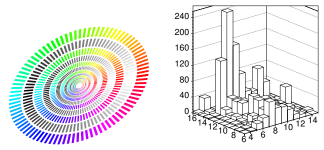


Figure 1: Deux images EPS

2 Exemple d'un tableau

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

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

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

Références

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

Index

1

Figure 2: Output generated by LaTeX from document shown in Figure 1

Each of the default values can be modified by redefining the corresponding `perl` variables in the initialization file `.latex2html.init`, as described in Section 3.5.4.

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

- ❶ the list of figures, containing a hyperlink pointing to document ❸ (containing the figure in question);
- ❷ the list of tables, containing a hyperlink pointing to document ❹ (containing the table in question);
- ❸ the first section, containing some text, a figure, and a hyperlink (`[1]`) pointing to an entry in the bibliography (document ❺);
- ❹ the second section, also containing some text and a table;
- ❺ the bibliographic references;
- ❻ the index, containing keywords that provide hyperlinks pointing to entry points in the various documents;
- ❼ an explanatory note detailing the procedure by which the document was translated into HTML. This text can be customized with the help of the option `-desc` (see Section 3.6.3).

3.5 Extending and Customizing the Translator

As the translator only partially covers the set of LaTeX commands and because new LaTeX commands can be defined arbitrarily using low level TeX commands, the translator should be flexible enough to allow end users to specify how they want particular commands to be translated.

3.5.1 Adding Support for Packages

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

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

Presently, the user can ask that particular commands and their arguments be ignored or passed on to LaTeX for processing (the default behavior for unrecognized commands is for their arguments remains in the HTML text). Commands passed to LaTeX are converted to images that are either "inlined" in the main document or are accessible via hypertext links. Simple extensions using the commands below may be included in the system initialization file `LATEX2HTMLDIR/latextohtml.config`, or in the customization initialization file `.latex2html-init` in the user's home directory or in the directory where the files to be converted reside.

3.5.2 Directing the Translator to Ignore Commands

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

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

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

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

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

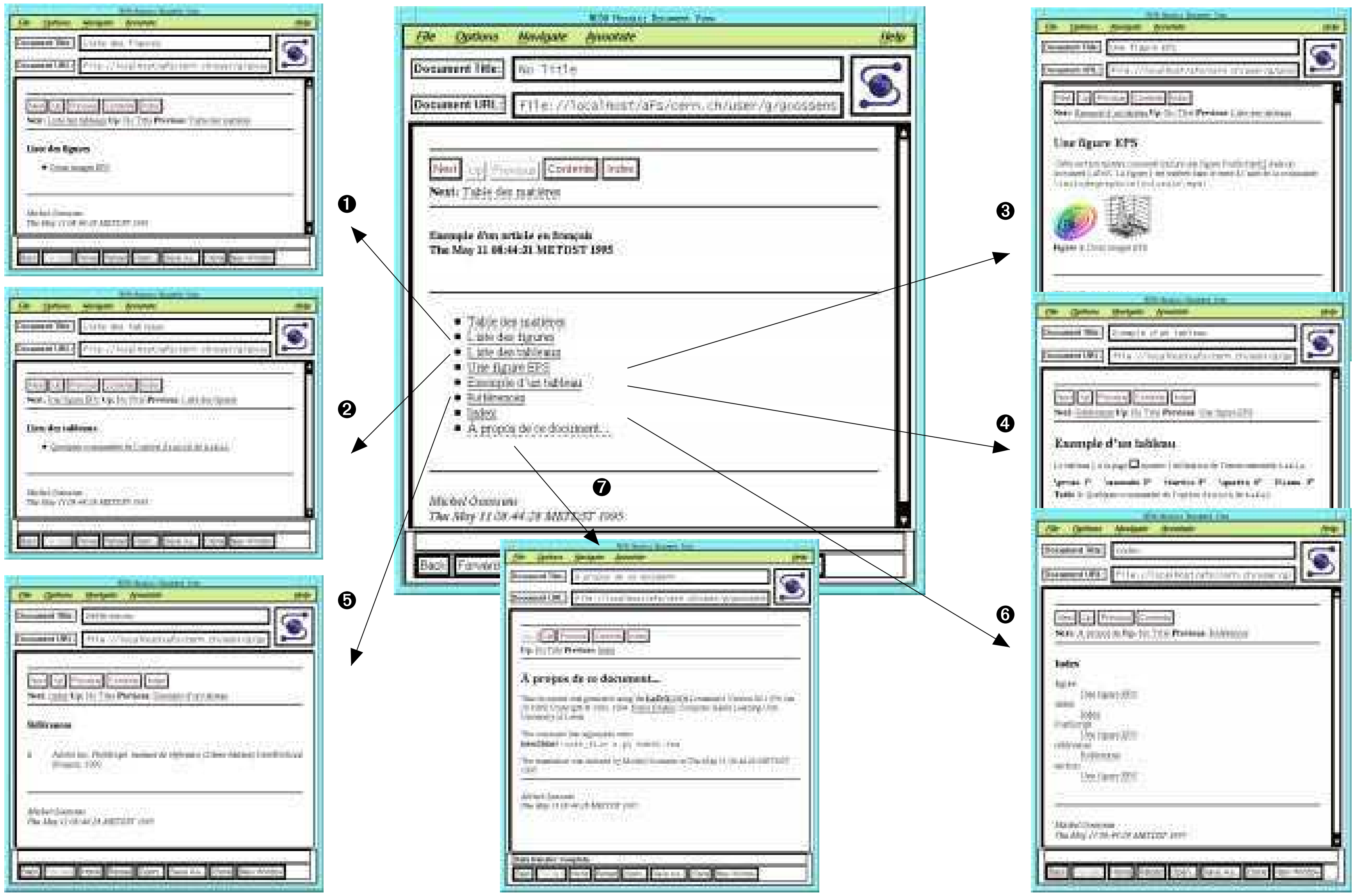
```
&ignore_commands( <<_IGNORED_CMDS_);
documentclass # [] # {}
linebreak# []
pagebreak# []
center
<add your commands here>
_IGNORED_CMDS_
```

3.5.3 Asking the Translator to Pass Commands to LaTeX

Commands that should be passed on to LaTeX for processing because there is no direct translation to HTML may be specified in the `.latex2html-init` file as input to the `process_commands_in_tex` subroutine. The format is the same as that for specifying commands to be ignored. Here is an example:

```
&process_commands_in_tex (<<_RAW_ARG_CMDS_);
fbox # {}
```

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


❶
❷
❸
❹
❺
❻
❼
File Options Navigate Annotate Help
Document Title:
Document URL:
Next Up Previous Contents Index
Back Forward

No Title

File://localhost/afs/cern.ch/user/g/goossens

Next: Table des matières

Exemple d'un article en français
Thu May 11 08:44:21 METDST 1995

- Table des matières
- Liste des figures
- Liste des tableaux
- Une figure EPS
- Exemple d'un tableau
- Références
- Index
- À propos de ce document...

Michel Goossens
Thu May 11 08:44:28 METDST 1995

Liste des figures

Liste des tableaux

Une figure EPS

Exemple d'un tableau

Références

Index

À propos de ce document...

Figure 3: The HTML structure generated from the LaTeX source of Figure 1 as visualized with the Mosaic browser

```
framebox # [] # [] # {}
<add your commands here>
_RAW_ARG_CMDS_
```

3.5.4 Customizing LaTeX2HTML

Besides honoring the options specified on the command line, LaTeX2HTML reads two standard files that can be used to customize its behavior. The first file, latextohtml.config, is a system-wide file (usually in the directory /usr/local/lib/latex2html). It contains the definitions for a complete installation, i.e.,, those common for all users, and specifies where certain external utility programs needed by LaTeX2-HTML are to be found on the system (such as LaTeX, dvips, gs, pmbplus). Moreover, in this file important perl variables are initialized to their default values. At the end of the file one has the possibility of specifying those LaTeX commands or environments that should be ignored, and those that should be passed on to LaTeX to be transformed into images for inclusion in the HTML file.

The second file, .latex2html-init, allows the user to customize LaTeX2HTML on an individual level. LaTeX2HTML will normally look for this file in the user's home directory (variable $HOME on Unix). The file can contain the same information as the global configuration file latextohtml.config and is thus the ideal place to overwrite default values or to specify in the perl language how certain specific LaTeX commands should be handled. It should be noted that the standard distribution of LaTeX2HTML already contains a few files with definitions for translations of supplementary LaTeX commands introduced by certain extension packages, such as german.perl, french.perl, html.perl and makeidx.perl. To help the user, the distribution comes with an example file dot.latex2html-init that can serve as a model for writing one's own .latex2html-init.

3.5.5 Creating a Customization File .latex2html-init

Before discussing examples of commands that can be put in the .latex2html-init customization file, it should be emphasized once more that this file, as well as all other files that are part of the LaTeX2HTML system, contain only perl instructions, and that one should thus have at least a basic understanding of this language before trying to edit any of these files.

Figures 4 and 5 show the contents of the example initialization file dot.latex2html-init. Its first parts initialize most of the perl variables used by the LaTeX2HTML system by setting them equal to their default values (as defined in the system-wide initialization file latex2html.config). Values that need not to be changed can be deleted from the file. When studying the various system variables, note the correspondence between the perl variables and the options of the latex2html described in Section 3.3).

Examples

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

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

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

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

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

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

Finally, we specify—as described in Sections 3.5.2 and 3.5.3—a list of commands to be ignored and passed to LaTeX.

```
### Commands to ignore ##############

&ignore_commands( <<_IGNORED_CMDS_);
documentclass # [] # {}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 5: `dot.latex2html-init` file (part 2). The navigation panel `perl` code is shown in Figure 9.

```
usepackage # [] # {}
mbox
makebox# [] # []
_IGNORED_CMDS_

### Commands to pass on to LaTeX{} ##

&process_commands_in_tex (<<_RAW_ARG_CMDS_);
includegraphics # [] # [] # {}
rotatebox # {} # {}
_RAW_ARG_CMDS_

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

We notice that the mandatory argument of the `\mbox` and `\makebox` commands is not specified, so that it will end up in the text, while the optional arguments of the `\makebox` command will disappear. In the case of the framed box commands `\fbox` and `\framebox`, both mandatory and optional arguments are passed on to LaTeX.

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

3.6 Hypertext Extensions

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

3.6.1 Hyperlinks in LaTeX

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

`\htmladdnormallink{`*text*`}{`⟨*URL*⟩`}`

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

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

`\htmladdnormallinkfoot{`*text*`}{`⟨*URL*⟩`}`

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

`\htmladdimg{`⟨*URL*⟩`}`

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

3.6.2 Cross-References between Living Documents

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

The mechanism is an extension of the simple `\label`-`\ref` pairs that can be used only within a single document. A symbolic label defined with a `\label` command can be referred to using a `\ref` command. When processed by LaTeX, each `\ref` command is replaced by the section number at which the corresponding `\label` occurred but when processed by `LaTeX2HTML` each `\ref` is replaced by a hypertext link to the place where the corresponding `\label` occurred. The new commands, detailed below, show how it is possible to refer to symbolic labels defined with `\label` in other external documents. These references to external symbolic labels are then translated into hyperlinks pointing to the external document.

Cross-references between documents are established with the commands:

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

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

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

3.6.3 Example of a Composite Document

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

We start with the LaTeX source document shown in Figure 1 and divide it, for demonstration purposes, into four sub-documents, shown in Figure 6, namely a master file (`ex20.tex`) and three secondary files (`ex21.tex`, `ex22.tex` and `ex2bib.tex`). We must first run all these files through LaTeX and then `LaTeX2HTML`, *in the correct order.* As we use cross-references to refer to document elements in external documents (with the commands `\externalref` and `\externallabels`) we first treat the secondary files before tackling the master file `ex20.tex`.

By default, `LaTeX2HTML` writes the files that it creates into a subdirectory with the same name as the original file, for example, after execution of the command:

```
> latex2html ex20.tex
```

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

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

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

1;      # Mandatory last line
```

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

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

In addition to our customization file `moninit.pl` (loaded with option `-init_file`), we also specify a title for the document (option `-t`), and add a description about the document (option `-info`). The

```
Top directory (TeX source files)
================================

569  ex20.tex
721  ex21.tex
627  ex22.tex
322  ex2bib.tex

Subdirectories (generated HTML files)
=====================================

       ex20
       ----
 1187  ex20.html
  109  images.pl
   93  labels.pl

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

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

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

Note the presence of the files `labels.pl` that contain information associating the symbolic names used on the `\label` commands in the original LaTeX source documents with the physical documents. The other files are one or more HTML files that were created by the translation process. `GIF` images are generated for all environments that `LaTeX2HTML` cannot translate gracefully into HTML. In this case the relevant part of the LaTeX code is first copied into a file `images.tex` that is run through LaTeX, which places each such environment on a separate page, so that the `dvips` program can produce a PostScript picture for each that is then (in principle) translated into `GIF` by using the Ghostscript program (see Section A.1 for more information about all these programs)

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

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

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

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

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

Master file (`ex2.tex`)

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

File containing images (`ex21.tex`)

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

File containing the table (`ex22.tex`)

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

File with the bibliography (`ex2bib.tex`)

Figure 6: Example of a composite document (LaTeX files)

result can be seen in the upper left corner of Figure 8.

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

```
This is LaTeX2HTML Version 0.6.4 (Tues Aug 30 1994)
          by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

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

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

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

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

Doing section links ....
Done.
```

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

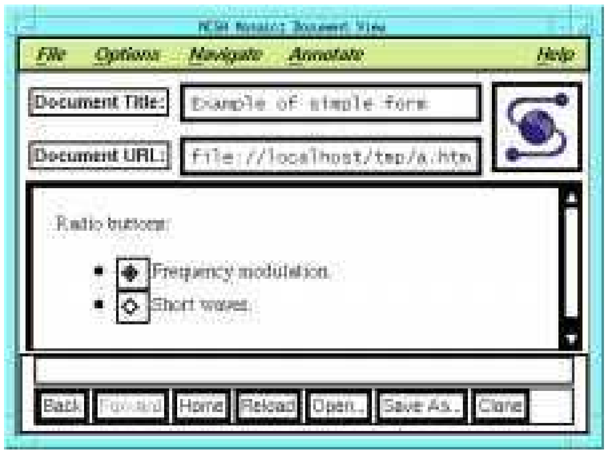
3.7 Including arbitrary HTML Markup

Raw HTML tags and text may be included using the rawtext environment. An interesting use of this feature is in the creation of interactive electronic forms. from within a LaTeX document. When producing the paper (DVI) version of a document the rawhtml environment is ignored.

Here is an example:

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

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

The result of running this electronic form with Mosaic would yield something like:



Document Title: Example of simple form

Document URL: file://localhost/tmp/a.htm

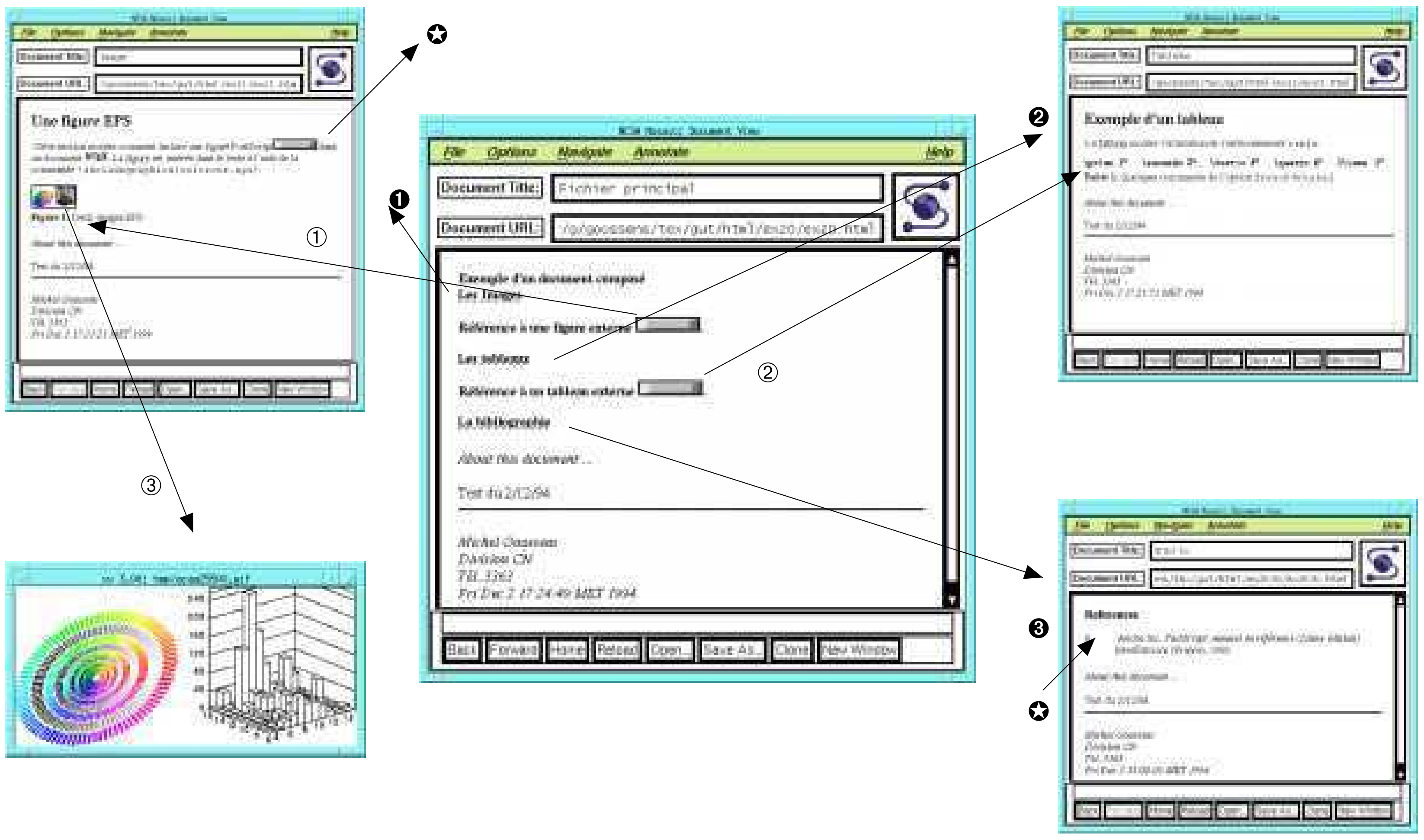
Radio buttons:

- Frequency modulation.
- Short waves.

3.7.1 Conditional Text

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

This would be useful in, for example, adding a long description of a multi-media resource to the



NCSA Mosaic: Document View

Document Title: Fichier principal

Document URL: /g/goossens/tex/gut/html/ex20/ex20.html

Exemple d'un document composé

Les Images

Référence à une figure externe

Les tableaux

Référence à un tableau externe

La bibliographie

About this document ...

Test du 2/12/94

Michel Goossens
Division CN
Tél. 3163
Fri Dec 2 17:24:49 MET 1994

Une figure EPS

Exemple d'un tableau

Références

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

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

paper version of a document. Such a description would be redundant in the electronic version, as the user can have direct access to this resource.

Using LaTeX commands involving counters (for example, numbered figures or equations) in conditional texts may cause problems with the values of the counters in the electronic version.

3.7.2 Cross-References shown as "Hyperized" Text

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

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

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

The meaning of the four arguments is as follows:

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

```
Example of the use of hyperref, with a
\hyperref
{reference to conditional text}
{reference to conditional text
  (see Section }
{ for more information)}
{sec:latexonly}
as an example.
```

Here is how it will be printed:

> Example of the use of hyperref, with a reference to conditional text (see Section 3.7.1 for more information) as an example.

In the hypertext version what would appear is:

> Example of the use of hyperref, with a <u>reference to conditional text</u> as an example.

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

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

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

3.7.3 Customizing the Navigation Panel

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

A simple mechanism for appending customized buttons to the navigation panel is provided by the command, `\htmladdtonavigation`. This takes one argument, which `LaTeX2HTML` appends to the navigation panel:

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

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

It is also possible to completely specify what is to appear in the navigation panel and its order of appearance. As each section is processed, `LaTeX2HTML` assigns relevant information to a number of global variables. These variables are used by the subroutine `navigation_panel` where the navigation panel is constructed as a string consisting of these variables and some formatting information.

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

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

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

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

`NEXT_TITLE`	next section;
`PREVIOUS_TITLE`	previous section;
`UP_TITLE`	"parent" section;
`NEXT_GROUP_TITLE`	next "group" section;
`PREVIOUS_GROUP_TITLE`	previous "group" section.

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

The number of words in each textual link is set by the `WORDS_IN_NAVIGATION_PANEL_TITLES` variable, which may also be changed in the configuration files. Figure 9 shows an example of a navigation panel.

3.8 Image Conversion

`LaTeX2HTML` converts equations, special accents, external PostScript files, and LaTeX environments it cannot directly translate into inlined images. It is possible to control the final appearance of such images, both for inline and display-type constructs.

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

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

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

`\htmlimage{`*options*`}`

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

`scale=`*scale-factor*
: the scale factor for the final image;

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

`thumbnail=`*reduction-factor*
: a small inlined image will be generated and placed in the caption; its size depends on the specification *reduction-factor* that downsizes the image by that amount. Note that this option implies `external`.

`map=`*image-map-URL*
: turns the inlined image into an active image map.[10]

An example is the following LaTeX code:

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

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

3.9 Internationalization

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

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

`LaTeX2HTML` will produce "Table des matières" instead of "Table of Contents".

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

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

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

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

```
sub navigation_panel {

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

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

    "<BR>\n" .          # Line break

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

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

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

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

Figure 9: Example definition of a navigation panel. (Note that "." is the `perl` string concatenation operator and "`#`" signifies a comment).

3.10 Known Problems

Users of `LaTeX2HTML` should take note of the following shortcomings of the translator.

- *Unrecognized commands and environments.*
  Unrecognized *commands* are ignored and any arguments are left in the text. Unrecognized *environments* are passed to LaTeX and the result is included in the document as one or more inlined images. Users can take care of this by providing information to `LaTeX2HTML` on how to handle such cases, either by deciding to ignore them (see Section 3.5.2 on page 179), or by defining a `perl` procedure (see Appendix C on page 204).
- *Cross-references.*
  References in environments that are passed to LaTeX for processing (such as `\cite` or `\ref` commands), are not processed correctly. On the other hand, `\label` commands are handled satisfactorily.
- *Order-sensitive commands.*
  Commands affecting global parameters during the translation that are sensitive to the order in which they are processed may cuase problems. In particular, counter manipulation with commands such as `\newcounter`, `\setcounter`, `\stepcounter` should be watched.
- *Index.*
  `LaTeX2HTML` generates its own index by saving the arguments of the `\index` command. The contents of the `theindex` environment are ignored.
- *New definitions.*
  New definitions (with the `\def`, `\newcommand`, `\newenvironment`, `\newtheorem` commands) do not work as expected if they are defined more than once. Indeed, only the last definition will be used throughout the document.
- *Scope of declarations and environments.*
  `LaTeX2HTML` processes sections as independent units. Thus, when the scope of a declaration or environment crosses section boundaries, the output may not be as expected.

4 HTML3 Extensions to `LaTeX2HTML`

4.1 The `math2html` Program

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

The LaTeX2HTML translator solves the problem of presenting mathematics in HTML by converting each mathematical sentence into a bitmap image. Although simple and straighforward, this approach seems a little unreasonable in general, since in many cases an article of a few pages can generate many hundreds of bitmap images, which have to be stored with the document, kept up to date, and transmitted with the document over the Internet, thus wasting an enormous amount of bandwidth. Therefore, a clear need for a translator from LaTeX mathematics into HTML3's primitive mathematics was considered an important goal. Thanks to the increased displaying capabilities of HTML3 complyable browsers, most inline mathematics and a fair proportion of display equations can be translated into HTML3 source code and hence transmitted in textual format together with the rest of the document, doing away with well over 90% of the images that are created in the HTML2 case where only bitmap images are generated. In addition, mathematics text can be searched for keywords as the rest of the document, thus increasing the value of the HTML document.

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

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

4.1.1 A few Examples

The HTML3 extensions translate quite a large fraction of not-too-complex LaTeX math constructs (for as far as they can be handled by the HTML3 DTD, of course).

A first explicit example is the code representing the differential cross-section of δ-ray production. The original LaTeX code and its result as typeset by LaTeX are shown in parts (a) and (b) of Figure 10, while the result of the translation by math2html of the LaTeX source in (a) into HTML3 is shown in (c), yielding the output with the Arena browser shown in (d). Part of the tree constructed by math2html when parsing this LaTeX input is shown in Figure 17 on page 213.

Multi-line mathematical constructs, such as arrays (array and eqnarray environments), are also handled without too many problems, and the present limits of the translation are due more to shortcomings of the HTML3 browser Arena (which is, after all, merely a beta-test version), than to intrinsic limitations in the approach. In Figure 11 we show the LaTeX source and result as seen with Arena of two multi-line environments.

4.1.2 Writing Convertible LaTeX

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

- Do not write the base of a superscript or a subscript outside the mathematics markup, i.e.,, `a$^2$` is not converted correctly but creates a bitmap image. The correct way is to write it `$a^2$` or `$\mathrm{a}^$` depending on whether or not one wants the letter "a" in math italic or in a roman font. When you leave the base outside of the math markup (the $ signs) the text between the mathematics delimiters is passed to the math2html translator and the latter does not know where to place the mathematics start (`<math>`) tag.
- Do not write nested array/tabular environments. The math2html translator cannot create an HTML3 counterpart for that markup since the HTML3 table model does not allow nested tables. Keeping the tables simple (not nested, for example) will improve their reusability.

4.2 Tables to HTML3 Conversion

Hennecke recently developed some code for treating LaTeX's tabular environment with LaTeX2HTML by translating it into HTML3-compliant tables. His patches[11] allow LaTeX2HTML to translate most LaTeX tables reasonably well. There are a few things it cannot do, but mainly because HTML tables are not quite as powerful as LaTeX tables. Most importantly, HTML tables are quite limited when it comes to borders, since they are not nearly as flexible in specifying borders as LaTeX tables. In his implementation, when a LaTeX table has a border anywhere, the resulting HTML table will have borders around all cells. LaTeX commands inside cells are treated

[11] Available from the URL ftp://ftp.crc.ricoh.com/pub/www/l2h/tables.tar.gz. The author Marcus E. Hennecke can be reached by email at marcush@crc.ricoh.com.

(a) LaTeX source that has to be translated:

```
\frac{d\sigma}{d\epsilon}=\frac{2\pi Z r_0^2m}{\beta^2(E-m)}\left[\frac{(\gamma-1)^2}
{\gamma^2}+\frac{1}{\epsilon}\left(\frac{1}{\epsilon}-\frac{2\gamma-1}{\gamma^2}\right)
+\frac{1}{1-\epsilon}\left(\frac{1}{1-\epsilon}\frac{2\gamma-1}{\gamma^2}\right)\right]
```

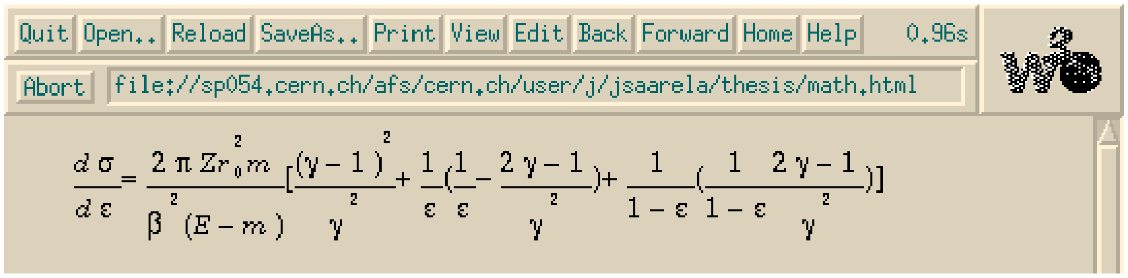
(b) Result of the above source as typeset with LaTeX:

$$\frac{d\sigma}{d\epsilon}=\frac{2\pi Z r_0^2m}{\beta^2(E-m)}\left[\frac{(\gamma-1)^2}{\gamma^2}+\frac{1}{\epsilon}\left(\frac{1}{\epsilon}-\frac{2\gamma-1}{\gamma^2}\right)+\frac{1}{1-\epsilon}\left(\frac{1}{1-\epsilon}\frac{2\gamma-1}{\gamma^2}\right)\right]$$

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

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

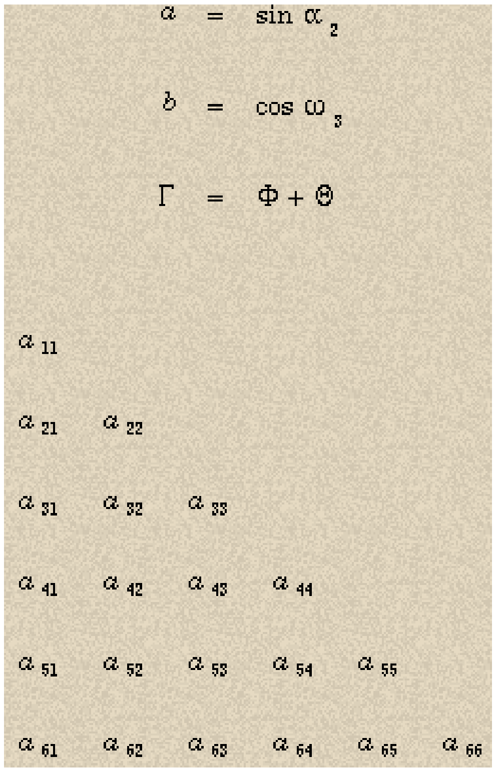
(d) Result of viewing of the HTML3 code of (c) with the `arena` browser:



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

Figure 10: Example of transforming LaTeX code to HTML3 with `math2html`

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

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



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

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

Figure 11: How `math2html` translates LaTeX multi-line mathematics into HTML3

as they should and declarations are limited in scope to the cell in which they appear (just as in LaTeX itself).

His additions can be placed in the `LaTeX2HTML` `perl` code itself, or in the customization files. In any case to leave open the possibility of generating tables with and without this feature turned on, a new command line option `-html_level` can be used to specify the level of HTML to be produced.

4.2.1 Examples

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

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

The result is seen at the top of Figure 12.

Math can also be handled (in this case it will be translated into images). With a little bit if "handwork" it could be translated into native HTML3:

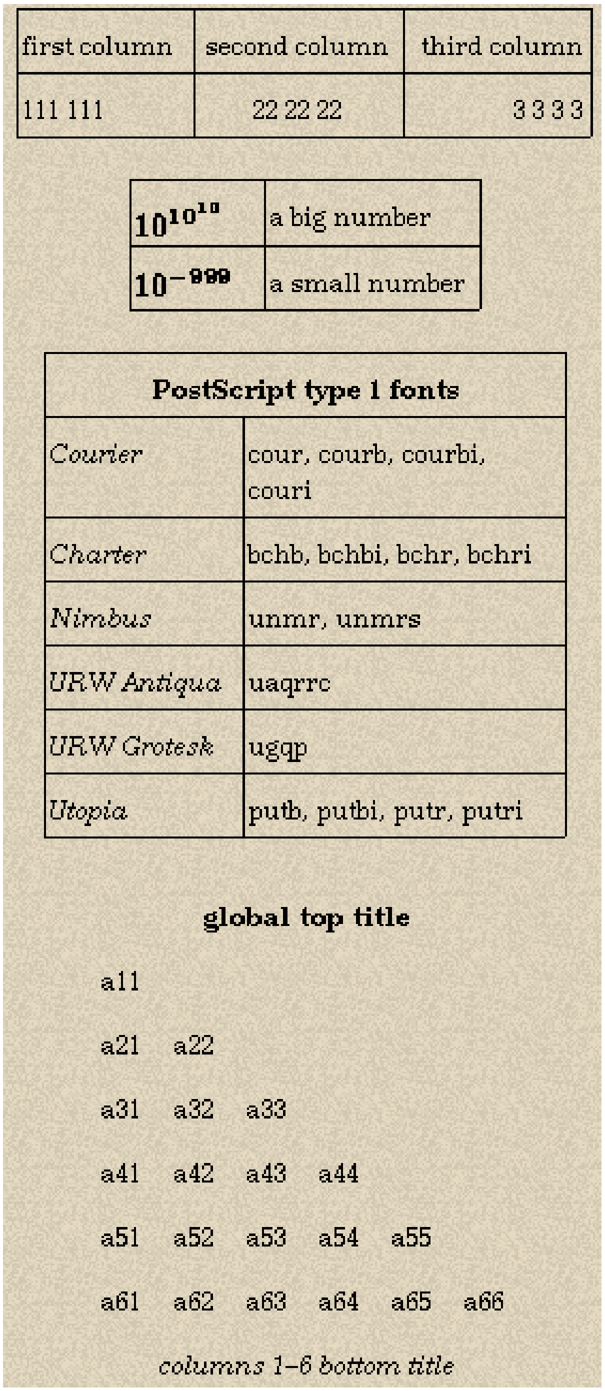
```
\begin{tabular}{|ll|}           \hline
$10^{10^{10}}$& a big number  \\\hline
$10^{-999}$   & a small number\\\hline
\end{tabular}
```

The result is seen in the second table from the top in Figure 12.

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

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

The result is seen in the third table from the top of Figure 12. Note that, even though only verti-



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

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

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

global top title

a11					
a21	a22				
a31	a32	a33			
a41	a42	a43	a44		
a51	a52	a53	a54	a55	
a61	a62	a63	a64	a65	a66

columns 1–6 bottom title

Figure 12: Four examples of `tabular` environments translated automatically to HTML3 as viewed with the Arena browser

cal rules were specified in the `tabular`'s preamble, rules are drawn everywhere. This is because the `BORDER` attribute of the `<TABLE>` tag in HTML3 has only one value, i.e.,, borders are present or absent everywhere.

Our final example has also a few `\multicolumn` commands, but also shows that non-specified cells are treated gracefully (this can be compared to the example in Section 4.1.1, where a similar table was built as an array inside math mode):

```
\begin{tabular}{cccccc}
\multicolumn{6}{c}{\bf global top title}\\
a11                           \\
a21 & a22                     \\
a31 & a32 & a33               \\
a41 & a42 & a43 & a44         \\
a51 & a52 & a53 & a54 & a55      \\
a61 & a62 & a63 & a64 & a65 & a66    \\
\multicolumn{6}{c}%
            {\em columns 1-6 bottom title}
\end{tabular}
```

The result is seen as the bottom table of Figure 12. As no vertical nor horizontal rules were specified in the input, the resulting table has no borders.

5 Caml based LaTeX to HTML Translation

Xavier Leroy (INRIA, France) developed a LaTeX to HTML translator based on the `Caml` language.[12]

What was needed was to translate a 200-page technical document (the reference manual and user's documentation for their implementation of the `Caml Light` programming language). This manual was written in LaTeX and contained some rather non-standard environments and macros written directly in TeX. Parts of the document were automatically generated: syntactic definitions (typeset from BNF descriptions) and descriptions of library functions (extracted from commented source code).

5.1 Why not just use `LaTeX2HTML`?

When `LaTeX2HTML` finds a LaTeX construct that it does not know how to translate into HTML, it simply turns it into a bitmap. This approach was considered inappropriate by Leroy et al., since

- information transformed into a bitmap is not searchable;
- bitmaps cannot easily be integrated into Macintosh or Windows online documentation systems;
- bitmaps are generally hard to read, since their resolution usually does not match that of the HTML viewer;
- as bitmaps can be quite large, transmission times increase and network bandwidth suffers.

[12] More information can be found at the URL `http://pauillac.inria.fr/~xleroy/w4g.html`.

In order to minimize the generation of bitmaps and to allow the production of a better quality HTML source, the information in the LaTeX source was tagged by LaTeX macros to explicitly show its semantics meaning. Special care was taken to avoid inline mathematics constructs, since they result in bitmap images, for example, `\var{x}` was preferred to its typographic equivalent `$x$` (denoting a meta-variable), and `\nth{v}{n}` was used to mean the n-th component of `v`, rather than writing `$v_n$`. The same technique was also used to eliminate "low-level" typesetting constructs and environments such as `center` and `tabular`.

When the document is typeset with LaTeX the new commands are simply translated into the appropriate typographic representation, but during the translation into HTML they are explicitly recognized and individually translated into a form corresponding to the possibilities of HTML; e.g.,, `\nth{v}{n}` would become something like `&lt;i&gt;v(n)&lt;/i&gt;`, showing `<i>v(n)</i>`.

Programs that automatically generated BNF or program fractions for inclusion in the LaTeX source were adapted so that the contents could now also be included without problems in the HTML source by "hiding" the generated material inside a `rawhtml` environment.

In the few places where more complex mathematical constructs were needed they were translated into a form acceptable to HTML by hand and stored inside a `rawhtml` environment, leaving the original mathematics expressions inside a `latexonly` environment. Thus both the LaTeX and HTML views of the document were optimized. Although in principle such an approach can lead to synchronization problems between the LaTeX and HTML parts of the information, it was found that, due to the care that was taken in using the generic markup approach outlined above, only about 0.2% of the source had to be manually translated.

Although Leroy and his collaborators originally planned to use `LaTeX2HTML` for translating their document into HTML, they found that some commands, especially those using verbatim-like constructs (notably the `alltt` environment), cannot be defined in `perl` in an easy way using the interface of `init` files described earlier. Therefore modifications have to be made inside the body of the `LaTeX2HTML` program itself, and this is very complicated since the inner workings of LaTeX2HTML are undocumented and scarcely commented, so that the perl code is not always clear to follow. Also, the memory requirements of LaTeX2HTML (especially the pre-1995 versions, when the tests were done) can be huge, exhausting all the memory available on the machine and causing the program to crash (this should no longer be a problem with the current version of LaTeX2HTML if the LaTeX source in divided into a set of smaller files). They therefore decided to write their own LaTeX-to-HTML translator for the extended subset of LaTeX commands they used.

This translator works in two main stages:

- The translator first reads the whole LaTeX document and outputs one large HTML document. It is written in Caml Light and uses the lexical analyzer generator camllex (the Caml equivalent of lex for C) heavily. Note that Caml is a modern, type-safe high-level programming language with good memory management, so that the translator has negligible memory requirements, runs quickly, is easy to extend, and took little time to develop.
- The output of the translator is then split into smaller parts (for instance at the <H1> or <H2> heading levels), and these parts are linked together using "Next" and "Previous" buttons. This linking is performed by two simple perl scripts.

In order to get a feeling of the result of the translation, one can look at a randomly chosen page from the manual that was converted. Figure 13 shows the result of LaTeX (viewed with xdvi) and Figure 14 is the result of the HTML conversion, as shown by Mosaic.[13] Appendix E takes a closer look at how the Caml system translates LaTeX commands into HTML.

5.2 Discussion

Based on their experience with writing and using their translator Leroy and collaborators draw the conclusions summarized in the next sections.

5.2.1 About the HTML Language

Despite its apparent simplicity, the HTML language is almost rich enough to format TeX-intensive technical documentation. The features that were absent were tables, sub- and superscripts. This is much less true today, since HTML3 already contains an interesting table model, and allows for super and subscripts. Moreover, the latest versions of Mosaic and Netscape support these functions.

[13] The complete manual—HTML and DVI files—are at the URLs http://pauillac.inria.fr/~xleroy/man-caml/ and ftp://ftp.inria.fr/lang/caml-light/Release7beta/cl7refman.dvi.gz, respectively.

5.2.2 About HTML Viewers

The quality of the typesetting performed by popular HTML viewers (e.g.,, Mosaic and Netscape) is very often insufficient. It seems especially difficult to ensure font consistency throughout a document.

The difficulty in finding good translators and adequate viewers probably has to do with the immaturity of the field. Leroy et al., are convinced that the widespread use of perl for programming translation tools is partly responsible for this situation. They state that perl is inherently not suited to the parsing and transformation of structured languages, such as LaTeX and HTML, and go on to say that languages with high-level parsing capabilities, real data structures and clean semantics are much more suited for these tasks.

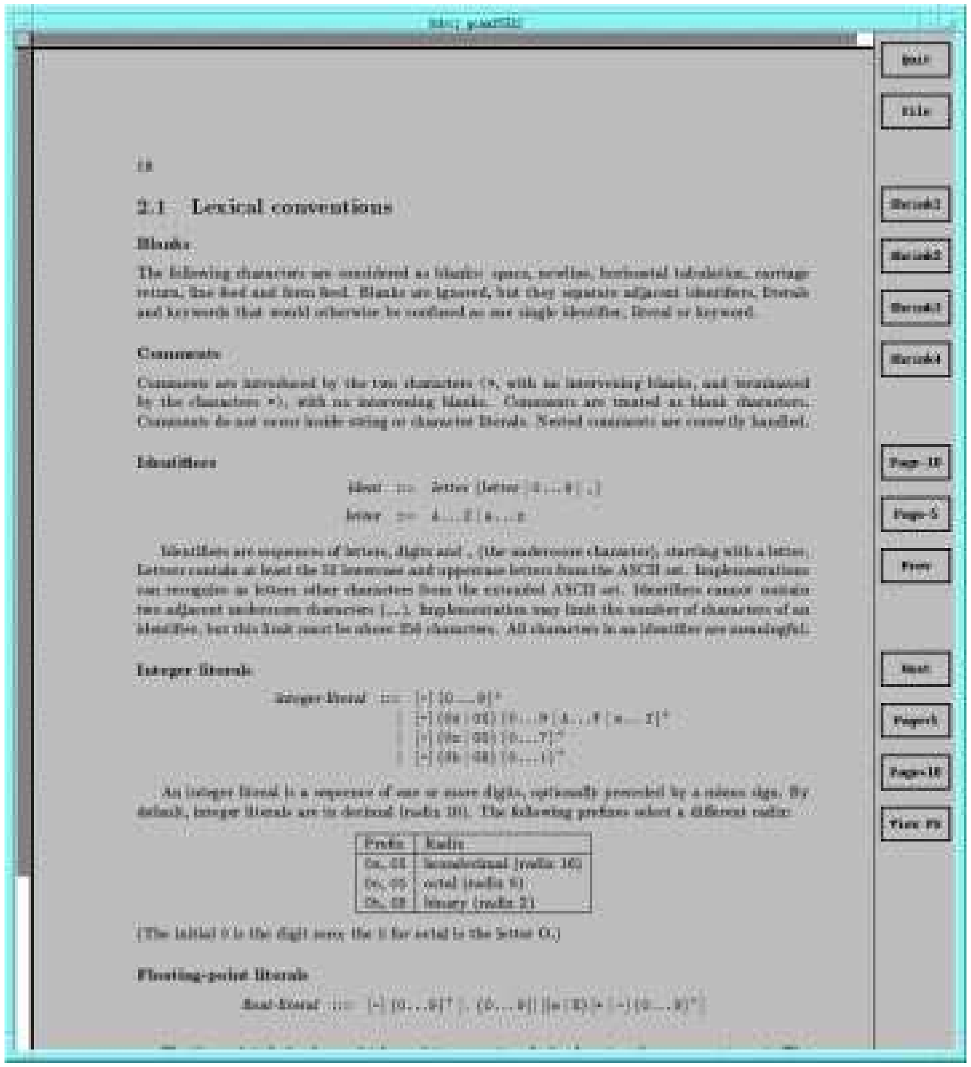
They also ask the question: what is the best markup language for preparing documentation so that it can be nicely printed but also easily transformed into HTML for publishing on the Web? They accept that, LaTeX presently being the *de facto* standard markup language used in computing and other science fields, it will be difficult in the short term to propose a solution other than to invest more effort in developing cleverer and more comprehensive LaTeX-to-HTML translators.

6 Converting HTML to LaTeX

Although utilities for obtaining PostScript representations from HTML files are readily available, either using HTML browsers, such as Mosaic, that offer PostScript as one of their output formats, or directly (for example, htps[14]) the visual layout of these documents is often appalling, and the structuring of the information has been made almost completely invisible. Often one would like to obtain a nicely typeset document that presents the information marked up in HTML in a structured way, with all document elements clearly identifiable. A translation into LaTeX allows one to combine the power of the TeX typesetting engine while at the same time exploiting the structural similarities between HTML and LaTeX as explained in Section 1 and Table 1.

A first program HTML2LaTeX translates a large fraction of the HTML commands into LaTeX, while SGML2TeX takes a more general approach and allows the transformation of an arbitrary SGML source into LaTeX.

[14] More information is at the URL http://info.cern.ch/hypertext/WWW/Tools/htps.html.



18

2.1 Lexical conventions

Blanks

The following characters are considered as blanks: space, newline, horizontal tabulation, carriage return, line feed and form feed. Blanks are ignored, but they separate adjacent identifiers, literals and keywords that would otherwise be confused as one single identifier, literal or keyword.

Comments

Comments are introduced by the two characters (*, with no intervening blanks, and terminated by the characters *), with no intervening blanks. Comments are treated as blank characters. Comments do not occur inside string or character literals. Nested comments are correctly handled.

Identifiers

ident ::= letter {letter | 0...9 | _}
letter ::= A...Z | a...z

Identifiers are sequences of letters, digits and _ (the underscore character), starting with a letter. Letters contain at least the 52 lowercase and uppercase letters from the ASCII set. Implementations can recognize as letters other characters from the extended ASCII set. Identifiers cannot contain two adjacent underscore characters (__). Implementation may limit the number of characters of an identifier, but this limit must be above 256 characters. All characters in an identifier are meaningful.

Integer literals

```
integer-literal ::= [-] {0...9}+
                  | [-] (0x | 0X) {0...9 | A...F | a...f}+
                  | [-] (0o | 0O) {0...7}+
                  | [-] (0b | 0B) {0...1}+
```

An integer literal is a sequence of one or more digits, optionally preceded by a minus sign. By default, integer literals are in decimal (radix 10). The following prefixes select a different radix:

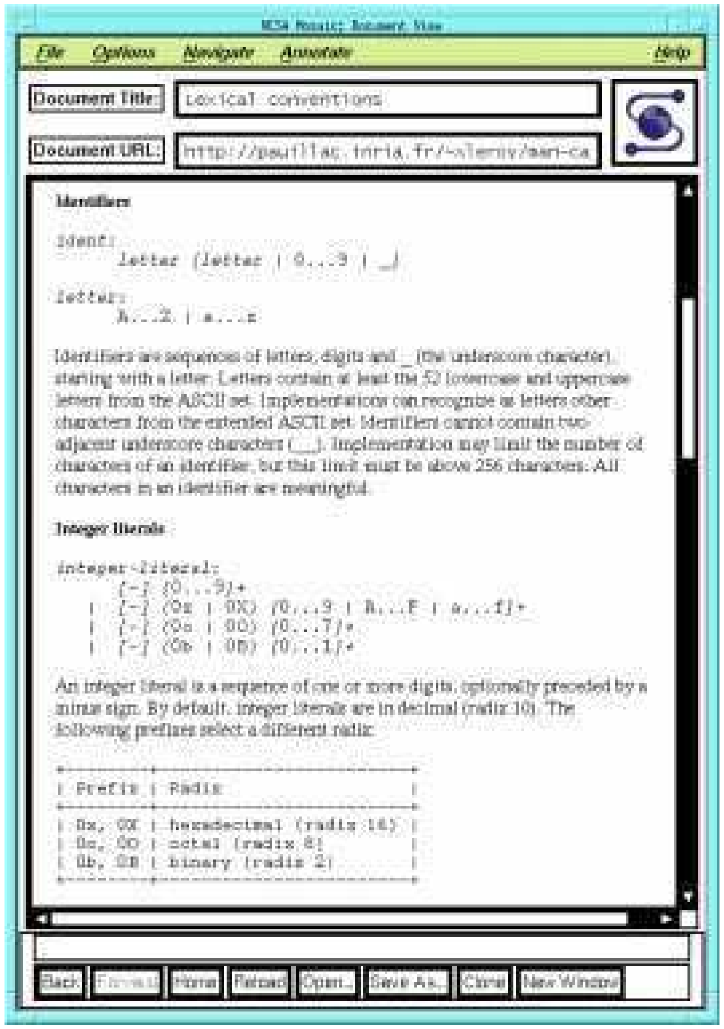
Prefix	Radix
0x, 0X	hexadecimal (radix 16)
0o, 0O	octal (radix 8)
0b, 0B	binary (radix 2)

(The initial 0 is the digit zero; the O for octal is the letter O.)

Floating-point literals

float-literal ::= [-] {0...9}+ [. {0...9}] [(e | E) [+ | -] {0...9}+]

Figure 13: Example page of `Caml` manual (LaTeX viewed with `xdvi`)



Document Title: Lexical conventions

Document URL: http://pauillac.inria.fr/~xleroy/man-ca

Identifiers

```
ident:
      letter {letter | 0...9 | _}

letter:
      A...Z | a...z
```

Identifiers are sequences of letters, digits and _ (the underscore character), starting with a letter. Letters contain at least the 52 lowercase and uppercase letters from the ASCII set. Implementations can recognize as letters other characters from the extended ASCII set. Identifiers cannot contain two adjacent underscore characters (__). Implementation may limit the number of characters of an identifier, but this limit must be above 256 characters. All characters in an identifier are meaningful.

Integer literals

```
integer-literal:
      [-] {0...9}+
   |  [-] (0x | 0X) {0...9 | A...F | a...f}+
   |  [-] (0o | 0O) {0...7}+
   |  [-] (0b | 0B) {0...1}+
```

An integer literal is a sequence of one or more digits, optionally preceded by a minus sign. By default, integer literals are in decimal (radix 10). The following prefixes select a different radix:

```
+--------+-----------------------+
| Prefix | Radix                 |
+--------+-----------------------+
| 0x, 0X | hexadecimal (radix 16) |
| 0o, 0O | octal (radix 8)       |
| 0b, 0B | binary (radix 2)      |
+--------+-----------------------+
```

Figure 14: Example page of `Caml` manual (HTML converted with `Caml` based translator viewed with Mosaic)

6.1 HTML2LaTeX, an HTML-to-LaTeX Converter

HTML2LaTeX is a C-program written by Nathan Torkington (New Zealand). Basically, the HTML parser of the NCSA Mosaic HTML browser is used for the translation. The calling sequence of the program is:

```
html2latex [options] [filenames]
```

For each input file specified, HTML2LaTeX transforms the HTML markup in the source into the equivalent LaTeX markup. When no filenames are specified, HTML2LaTeX will display a short description of how to use the program. If *filenames* is equal to -, then the input text is read on standard input stdin. For each input file an output file with the same name, but with the extension .tex instead of .html is generated.

6.1.1 Options

HTML2LaTeX has a number of options that modify its way of operation. The more important are:

-n	number the sections;
-p	start a new page after the titlepage (if present) or the table of contents (if present);
-c	generate a table of contents;
-s	write output information on stdout;
-t *Title*	generate a titlepage containing the title *Title*;
-a *Author*	generate a titlepage containing the author(s) *Author*;
-h *Start-Text*	introduce the text *Start-Text* immediately following the command \begin{document};
-h *End-Text*	introduce the text *End-Text* immediately preceding the command \end{document};
-o *options*	specify the options *options* on the \documentclass command.

6.1.2 Examples

Let us consider the following command:

```
html2latex -n - < file.html | more
```

In this case the file file.html is transformed into LaTeX and the result is shown on the screen. The option -n makes sure no section numbers are generated.

A more complex example is shown below:

```
html2latex -t 'HTML for Pedestrians' \
           -a 'First Last' -p         \
           -c -o'[12pt,twoside]{article}'\
                  my-article.html
```

In this case file my-article.html will be read, and the output written to the file my-article.tex. A titlepage (using the text "HTML for Pedestrians" as title and "First Last" as author) will be output on a separate page (option -p). A Table of contents (option -c) followed by a new page (option -p again) will also be generated. Sections will be numbered (default behavior). The LaTeX document will be typeset at 12 point using the document option twoside, to allow *two-sided* printing.

6.1.3 Limitations

The present version of HTML2LaTeX recognizes the following HTML tags: <TITLE>, <H1> to <H6>, for lists <OL>,<UL>, <DT>, <DD> and <LI>, plus the presentation tags <B>, <I>, <U>, <EM>, <CODE>, <SAMP>, <STRONG>, <KBD>, <VAR>, <DFN>, <LISTING>, and <CITE>. Of the entities only &amp;, &lt; and &gt; are handled correctly. The content fields of the tags <ADDRESS>, <DIR> and <MENU> are not handled correctly. Moreover, the COMPACT attribute of the <DL> tag is not honored and the text of the <TITLE> tag is ignored. Even worse, the body of the <PRE> elements are completely ignored.

Note that the complete HTML file is read into memory; this can lead to problems when handling large files on machines with limited memory capabilities.

6.2 SGML2TeX, a General-Purpose SGML to LaTeX Converter

SGML2TeX[15] is a program written by Peter Flynn (Flynn, 1995) that translates SGML tags into TeX instructions. At present the system is only implemented in PCL[16] running under MSDOS on a PC but the author has plans to rewrite it in a more portable programming language.

SGML2TeX does not verify the SGML source for correctness but accepts all SGML documents marked up using the reference concrete syntax. It is up to the user to define a LaTeX equivalent for each of the SGML document elements, their attributes, and the entities used in the source. A configuration file that contains a set of such predefined correspondences for certain elements, attributes, or entities, can be read by SGML2TeX, thus substantially alleviating the task of the user, who will only have to provide the

[15] For more information see the URL http://info.cern.ch/hypertext/WWW/Tools/SGML2TeX.html.

[16] PCL stands for Personal Computer Language, an interpreted language for DOS on the *86 chips. It is a very fast prototyping tool, not a production language since it cannot link executable images.

missing definitions. By default, i.e.,, in the absence of an explicit translation, SGML elements are translated in a form acceptable to LaTeX by adopting the following conventions:

- start tags get the prefix `\start` and end tags the prefix `\finish` followed by the tagname in upper case, followed by a pair of braces (`{}`). This pair of braces corresponds to a *do-nothing* definition for each of the tags thus handled;
- SGML entities of the form `&ent;` are translated into `\ent{}` and written into the output file;
- attributes are handled in the same way, but their value is specified between curly braces like a LaTeX argument.

Acknowledgments

We sincerely thank Nelson Beebe (Utah University, `beebe@math.utah.edu`) for email discussions and for his detailed comments of the compuscript. His many suggestions improvements have without doubt substantially increased the readability and quality of the article. We also want to acknowledge Steven Kennedy (CERN) for proofreading the article.

References

Flynn, Peter. HTML and TeX: Making them sweat. *TUGboat*, 16(3), 1995.

Goossens, Michel and Eric van Herwijnen. The elementary Particle Entity Notation (PEN) scheme. *TUGboat*, 13(1), pages 201–207, July 1992.

Goossens, Michel, Frank Mittelbach and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, Reading, 1994.

Lamport, Leslie. *LaTeX, User's Guide and Reference Manual (2nd Edition).* Addison-Wesley, Reading, 1994.

Ousterhout, John K. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, 1994.

Rumbaugh et al., *Object-Oriented Modeling and Design.* Prentice Hall, Inc., Englewood Cliffs, N.J., 1991.

Schwarzkopf, Otfried. The Hyperlatex story. *TUGboat*, 16(3), 1995.

Schwartz, Randal L. *Learning Perl.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1993.

Schrod, Joachim. Towards interactivity for TeX. *TUGboat*, 15(3), pages 309-317, September 1994.

Till, David. *Teach yourself Perl in 21 days.* Sams Publishing, Indianapolis, 1995.

Wall, Larry and Randal L. Schwartz. *Programming Perl.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1991.

A complete and up-to-date list of titles of books on HTML and `perl` is maintained by Nelson Beebe (Utah University, `beebe@math.utah.edu`) and can be found in his BibTeX databases `sgml.bib` and `unix.bib`, respectively, in the directory with URL address `ftp://ftp.math.utah.edu/pub/tex/bib/` .

◇ Michel Goossens and Janne Saarela
CERN, CN Division, CH-1211 Geneva 23, Switzerland
Email: `saarela@cern.ch`

Appendices

Appendices A and B present a few practical details that we found particularly relevant when installing or troobleshooting `LaTeX2HTML`.[17] Appendix C then provides some more information about the internal workings of the `LaTeX2HTML` program and how it can be extended by writing `perl` procedures. Finally, Appendix D contains technical information about the `math2html` extension to `LaTeX2HTML`, while Appendix E takes a closer look at Leroy's `Caml`-based LaTeX-to-HTML translator.

A `LaTeX2HTML`—Installation

A.1 Requirements to run `LaTeX2HTML`

`LaTeX2HTML` uses several publicly available tools that can be readily found on most computer platforms, namely:

- LaTeX (of course).
- `perl` (version 4 from patch level 36 onward, or, even better, version 5).
- `DBM` or `NDBM`, the Unix DataBase Management system.
- `dvips` (version 5.516 or later) or `dvipsk`.
- `gs` (Ghostscript version 2.6.1 or later).
- The `pbmplus` or better still the `netpbm` libraries; some of the filters in those libraries are used during the postscript to `GIF` conversion.
- For making transparent inlined images one needs `giftrans.c`[18] by A. Ley together with `pbmplus`. Alternatively, `netpbm` will do the trick.

To reduce the memory requirements of the translation, `LaTeX2HTML` spawns off separate Unix processes to deal with each of the `input`'ed or `include`'d files. As each process terminates, all the space that it used is reclaimed. Asynchronous communication between processes takes place using the Unix DataBase Management system (`DBM` or `NDBM`) which should be present. To take advantage of these changes it is necessary to split the source text into multiple files that can be assembled using LaTeX's `\input` or `\include` commands.

When `gs` or the `pbmplus` (`netpbm`) library are not available, one can still generate HTML output, but without images (using the `-no_images` option). Also, do not forget to include the html package with the `\usepackage` command if you want to include any of the hypertext extension commands described in Section 3.6.

A.2 Installing `LaTeX2HTML`

Those intending to install `LaTeX2HTML` on their system should read the manual in detail. Below we describe only the main steps.

- *Specify where `perl` is on the system.*
  In the files `latex2html`, `texexpand`, `pstogif`, and `install-test` modify the first line saying where `perl` is on your system.
- *Specify where the external utilities are on the system.*
  In the file `latex2html.config` give the correct pathnames for some directories (the `latex2html` directory and the `pbmplus` or `netpbm` library) and some executables (`latex`, `dvips`, `gs`).
  Note that `LaTeX2HTML` can be run even if one does not have some of these utilities.

One can also include the following supplementary customization:

- *Setting up different initialization files.*
  One can customize on a "per user" basis the initialization file. To this effect one should copy the file `dot.latex2html-init` into the home directory of any user who wants it, modify it according to the user's preferences and rename it to `.latex2html-init`.

  At runtime both `latex2html.config` and `$HOME/.latex2html-init` files will be loaded, but the latter will take precedence. Moreover, one can also set up a "per directory" initialization file by copying

[17] These sections are adapted from the `LaTeX2HTML` manual that is available at the URL `http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html`.

[18] `ftp://ftp.rz.uni-karlsruhe.de/pub/net/www/tools/giftrans.c`.

a version of the initialization file `.latex2html-init` into each directory where it should be effective. In this case an initialization file `/X/Y/Z/.latex2html-init` takes precedence over all other initialization files if `/X/Y/Z` is the "current directory" when `LaTeX2HTML` is invoked.

- *Make local copies of the `LaTeX2HTML` icons.*
  The `icons` subdirectory should be copied to a place in the local WWW tree where it can be served by the local server. Therefore, in the file `latex2html.config` file the value of the variable `$ICONSERVER` should be changed accordingly.

B `LaTeX2HTML`—Troubleshooting

This section gives a few hints about how to solve problems with `LaTeX2HTML`. As a general rule, if one gets really lost, one can obtain a lot of information from the `perl` system by setting the environment variable `DEBUG` to 1. In particular it will point out missing files or utilities. Below we present some often occurring problems and propose a way how to deal with them.

`LaTeX2HTML` just stops without further warnings.

Check the package files that are included, since they might contain raw TEX commands, which cannot be handled. In this case start `LaTeX2HTML` with the option `-dont_include` followed by the name of the package file in question. Alternatively, one can add the name of the package file to the variable `DONT_INCLUDE` in the `HOME/.latex2html-init` file, or create one in the current directory containing the following lines:

```
$DONT_INCLUDE = "$DONT_INCLUDE:<name-of-package-file>";
1;  # This must be the last line
```

Similarly, when the LATEX source file itself contains raw TEX command (`\let` is a common example!) `LaTeX2HTML` might also stop. Such commands should therefore be introduced inside a `latexonly` environment.

`LaTeX2HTML` gives an `Out of memory` message and crashes.

Divide the LATEX source file into several files that can be input using `\include` commands. One can also try the `-no_images` option.

The "tilde" (˜) does not show.

The easiest solution is to use the command `\~{}`. Alternatively it is possible to write something like:

```
\htmladdnormallink{mylink}
\begin{rawhtml}
  {http://host/~me/path/file.html}
\end{rawhtml}
```

Macro definitions do not work correctly.

As already mentioned, plain TEX definitions are be converted. But there can be problems even when using LATEX definitions (with the `\newcommand` and `\newenvironment` commands) if such definitions make use of *sectioning* or *verbatim* commands, since these are handled in a special way by `LaTeX2HTML` and cannot be used in macro definitions.

`LaTeX2HTML` behaves differently when running on the same file.

When noticing strange side-effects due to files remaining from previous runs of `LaTeX2HTML` one can use the option `-no_reuse` and choose `(d)` when prompted. This deletes intermediate files generated during previous runs. One can also delete those files oneself by removing the complete subdirectory created by `LaTeX2HTML` for storing the translated files. Note that in this case the image reuse mechanism is disabled.

```
> latex2html -no_reuse myfile.tex
This is LaTeX2HTML Version 95.1 (Fri Jan 20 1995) by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.
OPENING /afs/cern.ch/user/goossens/myfile.tex
Cannot create directory ./myfile: File exists
(r) Reuse the images in the old directory OR
(d) *** DELETE *** ./myfile AND ITS CONTENTS OR
(q) Quit ?
:d
```

Cannot convert PostScript images included in the LaTeX file.

It is likely that the macros used for including PostScript files (for example, `\epsffile` or `\includegraphics`) are not understood by `LaTeX2HTML`. To avoid this problem enclose them in an environment which will be passed to LaTeX anyway, for instance:

```
\begin{figure}
  \epsffile{<PostScript file name>}
\end{figure}
```

Another reason why this might happen is that the shell environment variable `TEXINPUTS` is undefined. This is not always fatal but if you have problems you can use full pathnames for included postscript files (even when the PostScript files are in the same directory as the LaTeX source file). Therefore it is important to check the setting of the `TEXINPUTS` environment variable and make sure that it ends in a colon "`:`", for example, "`.:/somedir:`".

Some of the inlined images are in the wrong places.

This occurs when any one of the inlined images is more than a (paper) page long. This is sometimes the case with very large tables or large PostScript images. In this case, one should specify a larger paper size (such as "`a3`", "`a2`", or even "`a0`") instead of the default ("`a4`") using the `LaTeX2HTML` variable `PAPERSIZE` in the file `latex2html.config`.

The labels of figures, tables or equations are wrong.

This can happen if inside figures, tables, equations or counters are used inside conditional text, i.e., inside a `latexonly` or a `htmlonly` environment.

With `Ghostscript 3.X` inline images are no longer generated for equations, etc.

One can run the installation script `install-test` again, or else change the way `gs` is invoked in the file `pstogif`, using something like:

```
open (GS, "|$GS -q -sDEVICE=ppmraw -sOutputFile=$base.ppm $base.ps");
```

Cannot get it to generate inlined images.

Try a small test file for example,

```
% image-test.tex
\documentclass{article}
\begin{document}
Some text followed by \fbox{some more text in a box}.
\end{document}
```

One should get something like the following:

```
> latex2html image-test.tex
This is LaTeX2HTML Version 95.1
        (Fri Jan 20 1995) by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/usr/goossens/image-test.tex

Reading ...
Processing macros ...
Translating ...0/1.....1/1.....
Writing image file ...
This is TeX, Version 3.1415 (C version 6.1)
(images.tex
LaTeX2e <1994/12/01>
Generating postscript images using dvips ...
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.05.08:1958' -> 6666_image
(-> 6666_image001) <tex.pro>[1]
Writing 6666_image001.ppm
Writing img1.gif
Doing section links .....
Done.
```

Problems encountered during the conversion from PostScript to GIF can be located by doing the translation manually, as shown below for a generation using gs 3.33.

```
> latex image-test
This is TeX, Version 3.1415 (C version 6.1)
(image-test.tex
LaTeX2e <1994/12/01>
(/usr/local/lib/texmf/tex/latex/base/article.cls
Document Class: article 1994/12/09 v1.2x Standard LaTeX document class
(/usr/local/lib/texmf/tex/latex/base/size10.clo))
No file image-test.aux.
[1] (image-test.aux) )
Output written on image-test.dvi (1 page, 348 bytes).
Transcript written on image-test.log.
> dvips -o image-test.ps image-test.dvi
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.05.08:2006' -> image-test.ps
<tex.pro>. [1]
cblelca% gs -dNODISPLAY pstoppm.ps
> gs -dNODISPLAY pstoppm.ps
Aladdin Ghostscript 3.33 (4/10/1995)
Copyright (C) 1995 Aladdin Enterprises, Menlo Park, CA.  All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
Usage: (file) ppmNrun
   converts file.ps to file.ppm (single page),
     or file.1ppm, file.2ppm, ... (multi page).
   N is # of bits per pixel (1, 8, or 24).
Examples: (golfer) ppm1run   ..or..   (escher) ppm8run
Optional commands you can give first:
   horiz_DPI vert_DPI ppmsetdensity
   horiz_inches vert_inches ppmsetpagesize
   (dirname/) ppmsetprefix
   page_num ppmsetfirstpagenumber
GS>(image-test) ppm1run
Writing image-test.ppm
GS>quit
> pnmcrop image-test.ppm >image-test.crop.ppm
pnmcrop: cropping 74 rows off the top
pnmcrop: cropping 139 rows off the bottom
pnmcrop: cropping 149 cols off the left
pnmcrop: cropping 249 cols off the right
> ppmtogif image-test.crop.ppm >image-test.gif
ppmtogif: computing colormap...
ppmtogif: 2 colors found
```

Still no inlined images are obtained.

When there have been no problems with the above file image-test.tex but some images have still not been successfully converted in some of the files then one should look in the directory with the generated HTML files for the two files images.tex and images.log. In particular, one should check whether there is something unusual in these files. One can copy the source images.tex into the directory of the original LaTeX file, run LaTeX on images.tex and check for any errors in the log file images.log. If errors are found then one should fix images.tex, put it back into the subdirectory with the HTML files, and run LaTeX2HTML on the original document using the option -images_only.

If one gets into trouble, then one should rerun LaTeX2HTML with the options -no_reuse and -no_images, for example,

```
> latex2html -no_reuse -no_images image-test.tex
This is LaTeX2HTML Version 95.1 (Fri Jan 20 1995) by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/user/goossens/image-test.tex
Cannot create directory ./image-test: File exists
(r) Reuse the images in the old directory OR
```

```
(d) *** DELETE *** ./image-test AND ITS CONTENTS OR
(q) Quit ?
:d

Reading ...
Processing macros ...
Translating ...0/1.....1/1.....
Writing image file ...

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

Doing section links .....

*********** WARNINGS ***********

If you are having problems displaying the correct images with Mosaic,
try selecting "Flush Image Cache" from "Options" in the menu-bar and
then reload the HTML file.

Done.
```

Now one should look into the file `images.tex` (as described above) and correct possible problems. Once everything seems alright, `LaTeX2HTML` should be run again with the option `-images_only`.

Some problems in displaying the correct inlined images may be due to the image-caching mechanisms of the browser. With some browsers, a simple "Reload Current Document" will be enough to refresh the images, but with others (including Mosaic) one may need to refresh the cache explicitly. With Mosaic one should select "`Flush Image Cache`" in the `Options` menu, then reload the HTML file.

C For perl Hackers Only—Inside LaTeX2HTML

The basic principle of `LaTeX2HTML` is that it reads a LaTeX source code document, converts the parts it recognizes into HTML and passes unknown parts to LaTeX, which, in turn, creates pictures out of them. These pictures are then placed inside the final hypertext document.

As discussed in Section 3.3, the program is started by specifying the LaTeX source code document together with a set of parameters. The result is a number of HTML documents and images as `GIF` or PostScript files. An overall flow-diagram is shown in Figure 15.

Unknown environments, tables, or pictures are also passed on to LaTeX and transformed into `GIF` or PostScript images, and kept inline or outside the hypertext documents.

C.1 The Translation Process

Below are shown the various phases that a document goes through when translated from LaTeX into HTML. Let us first consider the original LaTeX source document:

```
\documentclass{article}
\begin{document}
\section{test}

This is a list of two items:
\begin{itemize}
  \item{First item}
  \item{Second item}
\end{itemize}

\begin{verbatim}
This section includes some special characters such as $, <, >, _.
\end{verbatim}

\end{document}
```

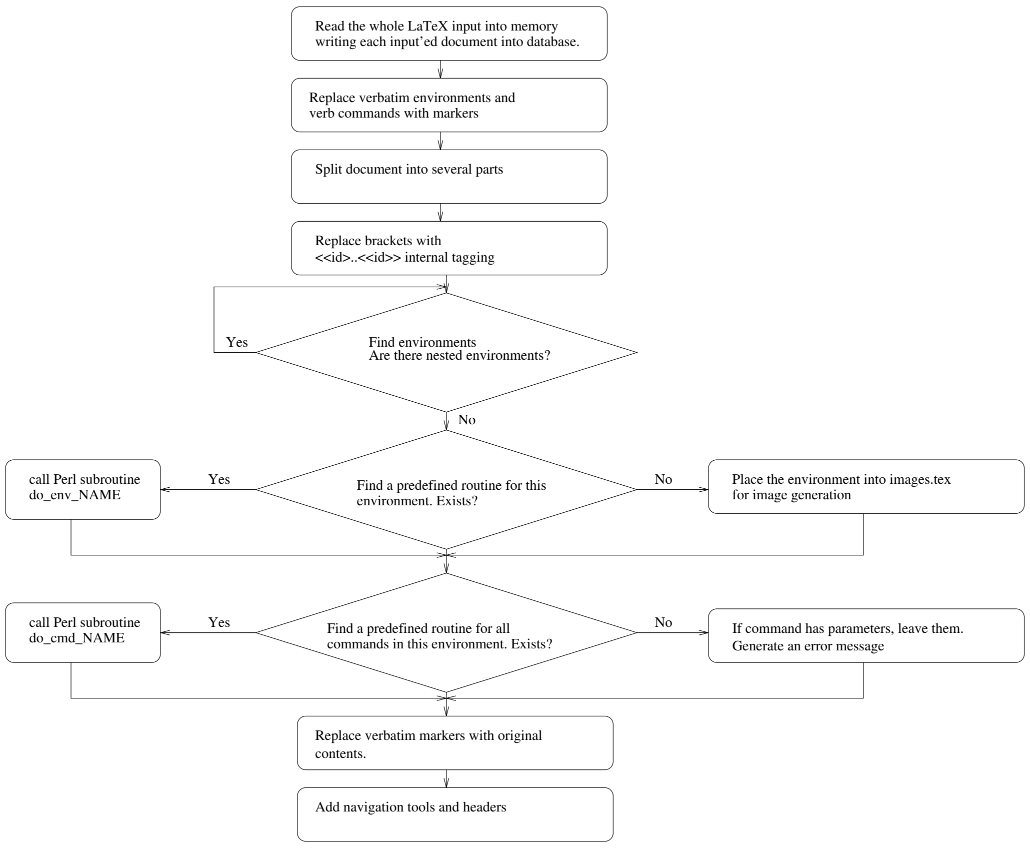

Read the whole LaTeX input into memory writing each input'ed document into database.
Replace verbatim environments and verb commands with markers
Split document into several parts
Replace brackets with <<id>>..<<id>> internal tagging
Yes
Find environments Are there nested environments?
No
call Perl subroutine do_env_NAME
Yes
Find a predefined routine for this environment. Exists?
No
Place the environment into images.tex for image generation
call Perl subroutine do_cmd_NAME
Yes
Find a predefined routine for all commands in this environment. Exists?
No
If command has parameters, leave them. Generate an error message
Replace verbatim markers with original contents.
Add navigation tools and headers

Figure 15: Flow diagram of the `LaTeX2HTML` system

This LaTeX source is first preprocessed by removing parts which have a special meaning in LaTeX, such as the `verbatim` and `\verb` constructs. In this example the verbatim part is stored in a separate file for later reference and a marker is placed inside the document together with a unique identification number "`<id>`" that will later be used to find the original text.

```
\documentclass{article}
\begin{document}
\section{test}

This is a list of two items:

\begin{itemize}
  \item{First item}
  \item{Second item}
\end{itemize}

<tex2html_verbatim_mark>verbatim1

\end{document}
```

At the end of preprocessing in the `mark_string` procedure, all the bracketed areas are replaced by `<<id>><<id>>` tags where "`id`" is identical at both ends of the originally bracketed text.

```
\documentclass<<1>>article<<1>>
\begin<<2>>document<<2>>
\section<<3>>test<<3>>

This is a list of two items:
```

```
\begin<<4>>itemize<<4>>
  \item<<5>>First item<<5>>
  \item<<6>>Second item<<6>>
\end<<7>>itemize<<7>>

<tex2html_verbatim_mark>verbatim1

\end<<8>>document<<8>>
```

Next, the document is split into sections. The LaTeX commands `\chapter`, `\section`, `\subsection`, etc. work as search-patterns used to split the document into items in an `perl` array. In our example, the conversion is configured to create a single document (i.e., no splitting).

For each section, the conversion rules are applied. These rules are implemented as procedures that have names like `do_env_X` or `do_cmd_X`, depending on whether one is dealing with a LaTeX environment or command, where `X` stands for either the environment or command name. For instance, our example document includes an `itemize` environment, and `LaTeX2HTML` will thus call the `perl` procedure `do_env_itemize`, that will receive as its parameter the contents of the environment, and will then parse that information.

Similarly a procedure `do_cmd_chapter` exists for converting a chapter command, and so on for the other sectioning commands. The resulting document after applying these conversion rules looks as follows.

```
<H1><A NAME=SECTION00100000000000000000> test</A></H1>
This is a list of two items:
<UL>
  <LI><#5#>First item<#5#>
  <LI><#6#>Second item<#6#>
</UL>
<tex2html_verbatim_mark>verbatim1
```

After this each document is enhanced with headers and navigation tools.

```
<!DOCTYPE HTML PUBLIC "-//W3O//DTD W3 HTML 2.0//EN">
<!Converted with LaTeX2HTML 95.1 (Fri Jan 20 1995) by Nikos
Drakos (nikos@cbl.leeds.ac.uk), CBLU, University of Leeds >
<HEAD>
<TITLE> test</TITLE>
</HEAD>
<BODY>
<meta name="description" value=" test">
<meta name="keywords" value="example">
<meta name="resource-type" value="document">
<meta name="distribution" value="global">
<BR>
<HR>
<A NAME=tex2html13 HREF="node2.html">
   <tex2html_next_page_visible_mark></A>
<A NAME=tex2html11 HREF="example.html">
   <tex2html_up_visible_mark></A>
<A NAME=tex2html5 HREF="example.html">
   <tex2html_previous_page_visible_mark></A>
<BR>
<B> Next:</B> <A NAME=tex2html14 HREF="node2.html">About this document</A>
<B>Up:</B>    <A NAME=tex2html12 HREF="example.html">No Title</A>
<B> Previous:</B><A NAME=tex2html6 HREF="example.html">No Title</A>
<BR>
<HR>
<P>
<H1><A NAME=SECTION00100000000000000000> test</A></H1>
This is a list of two items:
<UL><LI><#5#>First item<#5#>
    <LI><#6#>Second item<#6#>
```

```
</UL>
<tex2html_verbatim_mark>verbatim1
<BR>
<HR>
```

Finally, the markers are replaced with the contents to which they point. Extraneous tags are removed and the address of the author is appended to the file.

```
<!DOCTYPE HTML PUBLIC "-//W3O//DTD W3 HTML 2.0//EN">
<!Converted with LaTeX2HTML 95.1 (Fri Jan 20 1995) by Nikos
Drakos (nikos@cbl.leeds.ac.uk), CBLU, University of Leeds >
<HEAD>
<TITLE> test</TITLE>
</HEAD>
<BODY>
<meta name="description" value=" test">
<meta name="keywords" value="example">
<meta name="resource-type" value="document">
<meta name="distribution" value="global">
<P>
<BR>
<HR>
<A NAME=tex2html13 HREF="node2.html">
  <IMG ALIGN=BOTTOM ALT="next"
    SRC="http://asdwww.cern.ch/icons/next_motif.gif"></A>
<A NAME=tex2html11 HREF="example.html">
  <IMG ALIGN=BOTTOM ALT="up"
    SRC="http://asdwww.cern.ch/icons/up_motif.gif"></A>
<A NAME=tex2html5 HREF="example.html">
  <IMG ALIGN=BOTTOM ALT="previous"
    SRC="http://asdwww.cern.ch/icons/previous_motif.gif"></A>
<BR>
<B> Next:</B> <A NAME=tex2html14 HREF="node2.html">About this document</A>
<B>Up:</B>    <A NAME=tex2html12 HREF="example.html">No Title</A>
<B> Previous:</B> <A NAME=tex2html6 HREF="example.html">No Title</A>
<BR>
<HR>
<P>
<H1><A NAME=SECTION00100000000000000000> test</A></H1>
<P>
This is a list of two items:
<UL><LI>First item
  <LI>Second item
</UL>
<P>
<PRE>This section includes some special
characters such as $, &lt;, &gt;, _.
</PRE>
<P>
<BR> <HR>
```

C.2 Enhancing the Translator

From the previous section it is evident that the way to handle user commands and environments is to add `perl` code into the system or personal configuration files, as also discussed in Section 3.5. One can include as well a file with new definitions on the command line using the `-init_file` option.

To give a taste of how commands and environments are handled by `LaTeX2HTML`, we provide a few simple examples that nevertheless clearly show the powerful techniques used to generate HTML documents that preserve the information present in the original LaTeX document.

Let us first consider a LaTeX command (`\Ucom`) used to tag commands that have to be typed by the user on the keyboard. A possible definition using the HTML tag `<KBD>` for keyboard input is:

```
sub do_cmd_Ucom {
    local($_) = @_;
    s/$next_pair_pr_rx//o;
    join('',qq+<KBD>$&</KBD>+,$_);
}
```

The `perl` variable `$next_pair_pr_rx` contains the substitution pattern that extracts the string of characters surrounded by the following pair of delimiters. The string of characters and the delimiters are eliminated and the string is then copied between the HTML `<KBD>` and `</KBD>` appended to the output stream.

Similarly, one can translate the argument of a `\URL` command (containing a Universal Resource Locator) into an HTML anchor, as shown below:

```
sub do_cmd_URL {
    local($_) = @_;
    s/$next_pair_pr_rx//o;
    join('',"<a href=\"$&\">$&</a>",$_);
}
```

This procedure creates a link to the specified URL by returning an anchor with the URL as its target and an anchor description along with the rest of the as yet unprocessed document.

Our next example shows an enumerated list `EnumZW` of a special type whose "numbers" are icons available on a WWW server. The name of the icon depends on the value of the `perl` variable `count`, which is incremented for each `\item` command used inside the `EnumZW` environment. Everything takes place inside an HTML description list `<DL>`.

```
sub do_env_EnumZW {
    local($_) = @_;
    local($count) = 0;
    s|\\item|do {++$count; qq!<DT><IMG ALIGN=TOP ALT=""
    SRC="http://somewhere/icons/circled$count.xbm"><DD>!}|eog;
  "<DL COMPACT>$_</DL>";
    }
```

Two or more arguments can also be handled graciously, as shown by the following two commands, which have two and three arguments, respectively, and are typeset by LaTeX as follows:

$\boxed{\texttt{\textbackslash Command\{}\textit{arg1}\texttt{\}}}$

$\boxed{\texttt{\textbackslash Command[}\textit{arg1}\texttt{]\{}\textit{arg2}\texttt{\}}}$

The translation in `perl` is straighforward, since one must merely extract the relevant arguments from the input stream, one after the other.

```
sub do_cmd_BDefCm { # \BDefCm{Command}{arg1}
    local($_) = @_;
    s/$next_pair_pr_rx//o; $command = $&;
    s/$next_pair_pr_rx//o; $mandatory1 = $&;
    join('',"<strong>\\$command\{$mandatory1\}<\/strong>", $_);
}

sub do_cmd_BDefCom { # \BDefCom{Command}{arg1}{arg2}
    local($_) = @_;
    s/$next_pair_pr_rx//o; $command = $&;
    s/$next_pair_pr_rx//o; $optional1 = $&;
    s/$next_pair_pr_rx//o; $mandatory1 = $&;
    join('',"<strong>\\$command\[$optional1\]\{$mandatory1\}<\/strong>", $_);
}
```

Explaining all this perl code would lead us a little too far, but it should be fairly clear by now that before trying to develop new code for LaTeX2HTML it is a good idea to study in detail the way Nikos Drakos coded his program, not only in order to write perl code compatible with his conventions, but also as a source of inspiration for one's own extensions. Below we show definitions for frequently occurring regular expressions in the LaTeX2HTML perl code.

```
$delimiters = '\'\\s[\\]\\\\<>(=).,#;:~\/!-';
$delimiter_rx = "([$delimiters])";

# $1 : br_id
# $2 : <environment>
$begin_env_rx = "[\\\\]begin\\s*$O(\\d+)$C\\s*([^$delimiters]+)\\s*$O\\1$C\\s*";

$match_br_rx = "\\s*$O\\d+$C\\s*";

$optional_arg_rx = "^\\s*\\[([^]]+)\\]";        # Cannot handle nested []s!

# Matches a pair of matching brackets
# $1 : br_id
# $2 : contents
$next_pair_rx = "^[\\s%]*$O(\\d+)$C([\\s\\S]*)$O\\1$C";
$any_next_pair_rx = "$O(\\d+)$C([\\s\\S]*)$O\\1$C";
$any_next_pair_rx4 = "$O(\\d+)$C([\\s\\S]*)$O\\4$C";
$any_next_pair_rx5 = "$O(\\d+)$C([\\s\\S]*)$O\\5$C";

# $1 : br_id
$begin_cmd_rx = "$O(\\d+)$C";

# $1 : largest argument number
$tex_def_arg_rx = "^[#0-9]*#([0-9])$O";

# $1 : declaration or command or newline (\\)
$cmd_delims = q|-#,.~/\'`^"=|;        # Commands which are also delimiters!
# The tex2html_dummy is an awful hack ....
$single_cmd_rx = "\\\\([$cmd_delims]|[^$delimiters]+|\\\\|(tex2html_dummy))";

# $1 : description in a list environment
$item_description_rx =
        "\\\\item\\s*[[]\\s*((($any_next_pair_rx4)|([[][^]]*[]])|[^]])*)[]]";

$fontchange_rx = 'rm|em|bf|it|sl|sf|tt';

# Matches the \caption command
# $1 : br_id
# $2 : contents
$caption_rx = "\\\\caption\\s*([[]\\s*((($any_next_pair_rx5)|([[][^]]*[]])|[^]])*)[]])?$O(\\d+)$C([\\s\\S]*)$O\\8$C";

# Matches the \htmlimage command
# $1 : br_id
# $2 : contents
$htmlimage_rx = "\\\\htmlimage\\s*$O(\\d+)$C([\\s\\S]*)$O\\1$C";

# Matches a pair of matching brackets
# USING PROCESSED DELIMITERS;
# (the delimiters are processed during command translation)
# $1 : br_id
# $2 : contents
$next_pair_pr_rx = "^[\\s%]*$OP(\\d+)$CP([\\s\\S]*)$OP\\1$CP";
$any_next_pair_pr_rx = "$OP(\\d+)$CP([\\s\\S]*)$OP\\1$CP";

# This will be used to recognise escaped special characters as such
# and not as commands
$latex_specials_rx = '[\$]|&|%|#|{|}|_';

# This is used in sub revert_to_raw_tex before handing text to be processed by latex.
$html_specials_inv_rx = join("|", keys %html_specials_inv);

# This is also used in sub revert_to_raw_tex
$iso_latin1_character_rx = '(&#\d+;)';

# Matches a \begin or \end {tex2html_wrap}. Also used be revert_to_raw_tex
$tex2html_wrap_rx = '[\\\\](begin|end)\s*{\s*tex2html_wrap[_a-z]*\s*}';

$meta_cmd_rx = '[\\\\](renewcommand|renewenvironment|newcommand|newenvironment|newtheorem|def)';

# Matches counter commands - these are caught ealry and are appended to the
# file that is passed to latex.
$counters_rx ="[\\\\](newcounter|addtocounter|setcounter|refstepcounter|stepcounter|".
              "arabic|roman|Roman|alph|Alph|fnsymbol)$delimiter_rx";

# Matches a label command and its argument
$labels_rx = "[\\\\]label\\s*$O(\\d+)$C([\\s\\S]*)$O\\1$C";

# Matches environments that should not be touched during the translation
$verbatim_env_rx = "\\s*{(verbatim|rawhtml|LVerbatim)[*]?}";

# Matches icon markers
$icon_mark_rx = "<tex2html_(" . join("|", keys %icons) . ")>";

# Frequently used regular expressions with arguments
sub make_end_env_rx {
    local($env) = @_;
    $env = &escape_rx_chars($env);
    "[\\\\]end\\s*$O(\\d+)$C\\s*$env\\s*$O\\1$C";
}
sub make_begin_end_env_rx {
    local($env) = @_;
    $env = &escape_rx_chars($env);
    "[\\\\](begin|end)\\s*$O(\\d+)$C\\s*$env\\s*$O\\2$C(\\s*\$)?";
}
sub make_end_cmd_rx {
    local($br_id) = @_;
    "$O$br_id$C";
}
sub make_new_cmd_rx {
    "[\\\\](". join("|", keys %new_command) . ")"
        if each %new_command;
}
sub make_new_env_rx {
    local($where) = @_;
    $where = &escape_rx_chars($where);
   "[\\\\]$where\\s*$O(\\d+)$C\\s*(".
       join("|", keys %new_environment) .
           ")\\s*$O\\1$C\\s*"
               if each %new_environment;
}
sub make_sections_rx {
    local($section_alts) = &get_current_sections;
    # $section_alts includes the *-forms of sectioning commands
    $sections_no_delim_rx = "\\\\($section_alts)";
    $sections_rx = "\\\\($section_alts)$delimiter_rx"
}
sub make_order_sensitive_rx {
    local(@theorem_alts, $theorem_alts);
    @theorem_alts = ($preamble =~ /\\newtheorem\s*{([^\s}]+)}/og);
    $theorem_alts = join('|',@theorem_alts);
    $order_sensitive_rx =
        "(equation|eqnarray|caption|ref|counter|\\\\the|\\\\stepcounter" .
        "|\\\\arabic|\\\\roman|\\\\Roman|\\\\alph|\\\\Alph|\\\\fnsymbol)";
    $order_sensitive_rx =~ s/\)/|$theorem_alts|/ if $theorem_alts;
}
sub make_language_rx {
    local($language_alts) = join("|", keys %language_translations);
    $setlanguage_rx = "\\\\setlanguage{\\\\($language_alts)}";
    $language_rx = "\\\\($language_alts)TeX";
}
sub make_raw_arg_cmd_rx {
    # $1 : commands to be processed in latex (with arguments untouched)
    $raw_arg_cmd_rx = "\\\\(" . &get_raw_arg_cmds . ")([$delimiters]+|\\\\|#|\$)";
}
# Creates an anchor for its argument and saves the information in the array %index;
# In the index the word will use the beginning of the title of
# the current section (instead of the usual pagenumber).
# The argument to the \index command is IGNORED (as in latex)
sub make_index_entry {
    local($br_id,$str) = @_;
    # If TITLE is not yet available (i.e the \index command is in the title of the
    # current section), use $ref_before.
    $TITLE = $ref_before unless $TITLE;
    # Save the reference
    $str = "$str###" . ++$global{'max_id'}; # Make unique
    $index{$str} .= &make_half_href("$CURRENT_FILE#$br_id");
    "<A NAME=$br_id>$anchor_invisible_mark<\/A>";
}
```

D Technical Details of the `math2html` Program

D.1 Different Approaches

Various people have approached the problem of translating LaTeX into SGML or HTML using different programming paradigms. Joachim Schrod of the Technical University of Darmstadt, Germany has written a lisp parser for TeX code which can also be used for conversions (Schrod, 1994).[19] As already discussed in Section 5, Xavier Leroy used `Caml` to achieve the same goal, while `LaTeX2HTML` uses `perl` (other approaches based on `sgmls` also use that language).

Common to all approaches, whether using a procedural or a functional language, is the basic implementation. A lexer is used to recognize tokens from the input, a parser to create an internal representation and the conversion process produces the wanted output.

The major difference between functional and procedural languages is the way a language such as TeX can be parsed. Since the TeX language can at any point in the input define new rules for delimiters and symbols, the program parsing this input should also be able to cope with these dynamic features. Functional programming languages can do this by their nature, easily introducing new rules to the parser at runtime. This is what the parser written by Joachim Schrod can do. In comparison this cannot easily be done with a fixed grammar inside a parser.

Xavier Leroy's translator resembles a `bison`[20] input file. It sees groups of tokens and reduces the stacked input by given BNF-like rules. When it reduces the tokens it produces HTML output for LaTeX counterparts.

D.2 Implementation of the Translator

The `math2html` program, written in C++, takes LaTeX mathematics input, parses it and converts it into HTML3 mathematics (if possible). The program consists of the following components:

- `flex`, a fast lexical analyzer generator;
- `bison`, a parser generator;
- C++ code.

The parsing of LaTeX source code is, however, non-trivial, since its grammar has been developed step-by-step to cope with all LaTeX syntactical notations. The basic mathematical notation is presented here in detail.

`\[...\]`	Display mathematics.
`txt1 $...$ txt2`	Inline mathematics.
`{abc}`	Characters `a`, `b` and `c` are grouped into one.
`\abc`	Characters `a`, `b` and `c` are a control sequence.
`a^b`	Superscripts (`b` can be a group of characters).
`a_b`	Subscripts (`b` can be a group of characters). Superscripts and subscripts can be nested.

The lexical analyser recognizes LaTeX primitives by generating tokens for the parser. A control sequence, plain text, superscript, subscript, begingroup, endgroup, fraction, array, column separators and end of row are examples of typical tokens. These tokens correspond to classes. These classes are depicted in Figure 16 with the object modeling technique (OMT) (Rumbaugh et al., 1991).

The class library presents the supported structures of LaTeX mathematics as sums, integrals, fractions, plain input, sequences and groups. These are currently the only primitives which can be reasonably converted into HTML3 mathematics. A few examples of basic primitives that can be treated by `math2html` are shown below:

```
Sum:      \sum_{i=1}^{n}i          Integral: \int_0^1f(x)dx
Fraction: \fraction{1}{n}          Sequence: \infty
Group:    {|x+1|}^2
Table:    \begin{table}{lr}        Eqntable: \begin{eqnarray}
            a & b \\ c & d                     y&=&x^2\\z&<=& x^3
          \end{table}                        \end{eqnarray}
```

[19] The system is available at URL `ftp://ftp.th-darmstadt.de/pub/tex/src/etls/`.
[20] Bison is a parser generator in the style of `yacc`.

The parser analyzes the tokens using an *ad hoc* BNF grammar generated specifically to parse LaTeX code. When reducing the input according to the grammar rules, the parser generates instances of C++ classes (see Figure 16), which correspond to these LaTeX primitives. Once the whole input has been parsed, the internal representation is linked together so that all these instances can be reached from one top-level list.

The conversion is implemented by calling a conversion method to each instance in the list. Each primitive knows how to convert itself and also propagates the conversion to all its children nodes.

An instance of the runtime organization of the parsing tree corresponding to the example of Figure 10 is shown in Figure 17 on the next page.

D.3 Mapping of Control Sequences

Since the wide variety of different control sequences is quite impossible to hardcode into the program, an external configuration file is read every time the program starts. The mapping between control sequences and HTML3 counterparts is read into a hash table and in this way the user can configure the program to cope with special control sequences not natively supported by the converter. An example of this is the Particle Entity Notation scheme (Goossens and van Herwijnen, 1992), a set of standard control sequences for representing elementary particles. This naming scheme consists of about 240 control sequences and their presentation counterparts. The configuration file maps each control sequence into its HTML3 counterpart using the following format:

```
\Pgppm          &pi;<sup>&plusmn;</sup>        \Pgpz           &pi;<sup>0</sup>
\Pgh            &eta;                          \Pgr            &rho;(770)
\Pgo            &omega;(783)                   \Pghpr          &eta;'(958)
\Pfz            <t>f</t><sub>0</sub>(975)
```

D.4 Program Heuristics

The program uses a few heuristics in order to be able to parse LaTeX code successfully. If these coding rules are not used, parsing may fail.

Optional parameters specified between square brackets (`[]`) after a control sequence are not parsed with respect to the control sequence. Therefore, there should be no space left between the control sequence and the opening bracketwhere optional parameters are used. Space should be left if the brackets are used as delimiters. An example is the difference between the following two control sequences:

```
\root[3]{\pi}                          \left [ \pi+2 ]
```

It is also worth noticing that all control sequences not supported primitively in `math2html`, apart from integrals, fractions, roots, sums and a few others, are dropped out during the conversion, for example, no text is produced in the HTML3 version. The only way to convert them is to create specific code or map it in the configuration file.

D.5 Interfacing with other Programs

This application was built to make it easy for other applications to call it. The program can either be compiled into a single executable program with a command line interface or into a library that can be linked with any other applications.

The modular approach has the advantage of being both simple and straightforward. The object-oriented implementation makes the linearisation of the internal representation almost effortless and eases the future addition of new HTML3 primitives by the user. The program is quite flexible and, as pointed out above, can be used in different contexts: embedded or stand-alone.

D.6 Drawbacks of the presented Solution

The end-user may find extending the program too difficult, especially if one has no experience with `flex`, `bison`, or C++. The configuration file that comes with the program provides an easy way to do simple mappings, but if one wants to add more functionality, one must understand the organization of the program.

As trickier tables and equations need to be converted, the program will need extension for analyzing the internal tree structure and to add, modify or delete specific nodes.

If the LaTeX input code uses low-level TeX commands the program will not be able to handle the input.

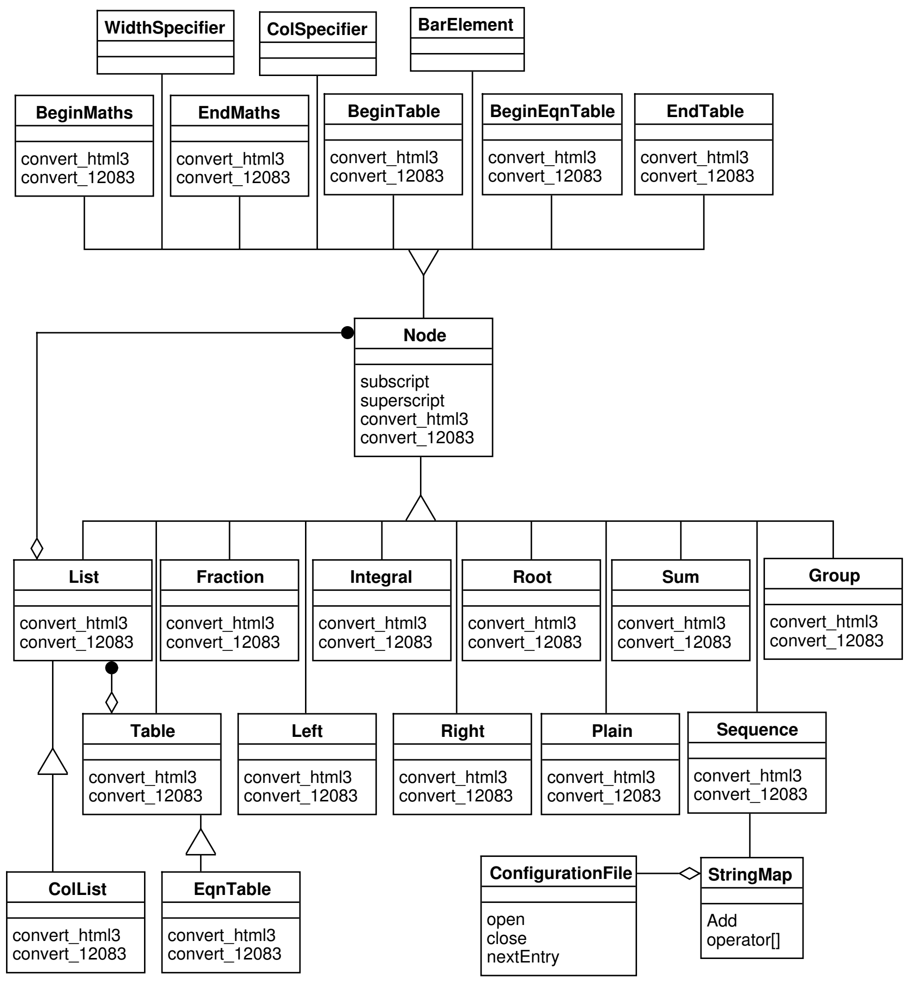


Figure 16: OMT model of the mathematics conversion program

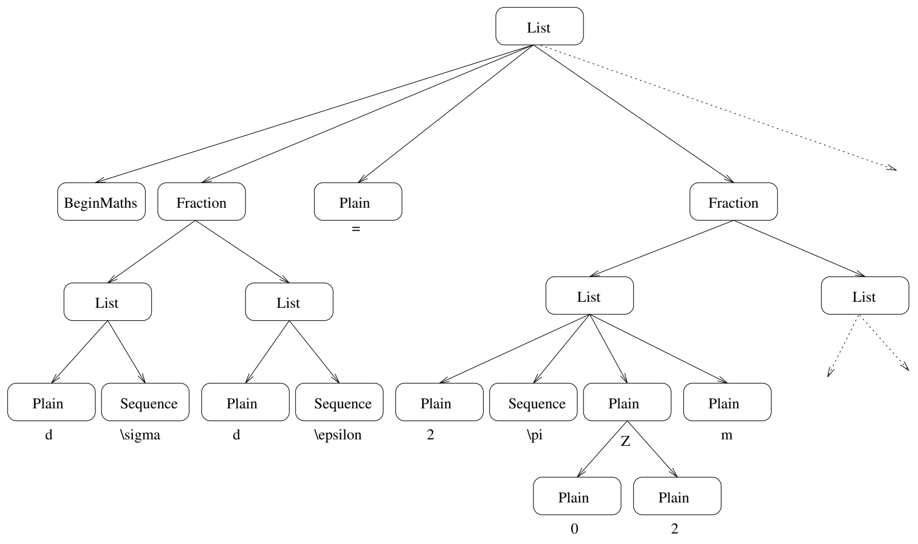


Figure 17: Example of a runtime parsing tree

E Using the Caml System for Translating LaTeX to HTML

The program works by expressing the LaTeX grammar in a yacc-like format and parsing the LaTeX input lines rule by rule, converting all recognized patterns into HTML. An example of Caml Light grammar rules for LaTeX to HTML conversion is given below.

```
(* Font changes *)

  | "{\\it" | "{\\em"
              { print_string "<i>"; upto `}` main lexbuf;
                print_string "</i>"; main lexbuf }

  | "{\\bf"     { print_string "<b>"; upto `}` main lexbuf;
                print_string "</b>"; main lexbuf }

  | "{\\tt"     { print_string "<tt>"; upto `}` main lexbuf;
                print_string "</tt>"; main lexbuf }

  | `"`         { print_string "<tt>"; indoublequote lexbuf;
                print_string "</tt>"; main lexbuf }

(* Verb, verbatim *)

  | "\\verb" _  { verb_delim := get_lexeme_char lexbuf 5;
                print_string "<tt>"; inverb lexbuf;
                print_string "</tt>"; main lexbuf }

  | "\\begin{verbatim}"
              { print_string "<pre>"; inverbatim lexbuf;
                print_string "</pre>"; main lexbuf }
```

Unlike LaTeX2HTML the program does not pass mathematics on to the TeX engine in order to create bitmap images for unparsable input, but produces plain text only. As the LaTeX control sequences recognized by the program are read from a separate file, the addition of new commands and their HTML counterparts is relatively easy. An example of such mappings is the following:

```
def "\\chapter"           [Print "<H1>"; Print_arg; Print "</H1>\n"];
def "\\chapter*"          [Print "<H1>"; Print_arg; Print "</H1>\n"];

def "\\begin{itemize}"    [Print "<p><ul>"];
def "\\end{itemize}"      [Print "</ul>"];

def "\\begin{enumerate}"  [Print "<p><ol>"];
def "\\end{enumerate}"    [Print "</ol>"];

def "\\begin{description}" [Print "<p><dl>"];
def "\\end{description}"  [Print "</dl>"];

def "\\begin{center}"     [Print "<blockquote>"];
def "\\end{center}"       [Print "</blockquote>"];
```

The use of this program requires the compilation of the Caml Light distribution, available for a variety of platforms. The language is compiled with an intermediate step in the C language. The executable program suffers from some overhead, mainly affecting execution time.

Because the program does not deal with mathematics and tables, it can only be used for a restricted set of documents. To be useful for the general user it will have to be extended to convert mathematics and tables either into bitmaps or into HTML3.

Calendar

1995

Jul	6	DANTE TeX–Stammtisch at the Universität Bremen, Germany. For information, contact Martin Schröder (`l15d@zfn.uni-bremen.de`; telephone 0421/628813). 18:30, Universität Bremen MZH, 4th floor, across from the elevator.
Jul	24 – 28	**TUG 16th Annual Meeting**: Real World TeX, St. Petersburg Beach, Florida. For information, send e-mail to `tug95c@scri.fsu.edu`. (For a preliminary announcement, see *TUGboat* 15, no. 2, p. 160.)
Jul	27	TeX and Semitic Languages, Technion, Haifa, Israel. (For information, contact one of the organizers: Dan Berry `dberry@cs.technion.ac.il` or Yannis Haralambous `Yannis.Haralambous@univ-lille1.fr`.)
Aug	10	DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Jul 6.)
Sep	4 – 8	EuroTeX '95, Papendal, Arnhem, The Netherlands. For information, contact `eurotex@cs.ruu.nl`.
Sep	7	DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Jul 6.)
Sep	14 – 15	DANTE e.V., 13th general meeting, Humboldt-Universität, Berlin, Germany. (For information, contact Christiane Schöbel `dante-mv13@rz.hu-berlin.de`.)
Sep	14	DANTE TeX–Stammtisch, Wuppertal, Germany. For information, contact Andreas Schrell (`Andreas_Schrell@FernUni-Hagen.DE`, telephone 0202/502354). Second Thursday, 19:30, Gaststätte Yol, Ernststraße 43, (near Robert-Daum-Platz), 42117 Wuppertal.
Oct	2 – 5	CyrTUG'95 Annual Meeting, Protvino (Moscow region), Russia. (For information, contact `cyrtug@mir.msk.su`.)

TUG Courses, San Francisco, California
(For information, contact `tug@tug.org`.)

Oct	9 – 13	Beginning/Intermediate TeX
Oct	16 – 20	Intensive LaTeX
Oct	23 – 27	Modifying LaTeX Document Classes
Oct	30 – Nov 3	Advanced TeX and Macro Writing

Oct	10	DANTE TeX–Stammtisch at the Universität Bremen, Germany. For information, contact Martin Schröder (`MS@Dream.HB.North.de`; telephone 0421/628813). First Tuesday (if not a holiday), 18:00, Universität Bremen MZH, 28359 Bremen, 4th floor, across from the elevator.
Oct	12	DANTE TeX–Stammtisch, Wuppertal, Germany. (For details, see Sep 14.)
Nov	7	DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)
Nov	9	DANTE TeX–Stammtisch, Wuppertal, Germany. (For details, see Sep 14.)
Dec	5	DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)
Dec	14	DANTE TeX–Stammtisch, Wuppertal, Germany. (For details, see Sep 14.)

1996

Jan	9	DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)
Feb	6	DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)

Status as of 31 May 1995

Mar	5	DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)
Mar	27 – 29	DANTE '96 and 14th general meeting of DANTE e.V., Universität Augsburg, Germany. For information, contact Gerhard Wilhelms (`dante96@Uni-Augsburg.de`).
Apr	2	DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)
Apr	7 – 10	EP96, the International Conference on Electronic Documents, Document Manipulation and Document Dissemination, Xerox Palo Alto Research Center, Palo Alto, California. *Deadline for submission of papers: 4 December 1995.* For information, contact `ep96@xsoft.xerox.com`.
May	7	DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)
Jul	18 – 21	SHARP 1996: Society for the History of Authorship, Reading and Publishing, Fourth Annual Conference, Worcester, Massachusetts. *Deadline for proposals: 20 November 1995.* For information, contact the American Antiquarian Society, `cfs@mark.mwa.org`.

For additional information on the events listed above, contact the TUG office (415-982-8449, fax: 415-982-8559, e-mail: `tug@tug.org`) unless otherwise noted.

Late-Breaking News

Production notes

Mimi Burbank

I oversaw the production of this issue of *TUGboat* and I had to manage a production team working in three different time zones and spanning two continents. But with e-mail and ftp and so forth, time and distance were not a problem. It soon was obvious that some of the production team were at work for practically all of the 24-hour day. Each member was involved in the successful completion of this issue, as well as maintaining and upgrading the system used at SCRI. Bandwidth was often a problem for those "across the pond" and Mimi's main activity was running and previewing files and reporting back to those across the ocean about layout.

Electronic input for articles in this issue was received by e-mail as well as retrieved from remote sites by anonymous ftp. In addition to text, the input to this issue includes METAFONT source code, 38 `.fd` files, and 11 `.vf` files. There were a considerable number and variety of PostScript files. One article contained 39 figures, and required 81 files to produce final output. Over 200 files were used (as input files) to generate final copy; over 300 files represent fonts (`.tfm` and rasters), device-specific translations, earlier versions of files, auxiliary information, and records of correspondence with authors and referees. The Y&Y advertisement was received via anonymous ftp as a PostScript file.

All articles were received as fully tagged for *TUGboat*, using either `plain`-based or LaTeX conventions described in the Authors' Guide (see *TUGboat* 10, no. 3, pages 378–385). 80% of the articles received were in LaTeX 2_ε. Several authors requested copies of the current version of LaTeX 2_ε macros for *TUGboat*, and we were happy to provide these.

Font work was required on all of the articles in the "Font Forum" section. The article by Jeffrey (page 79) used metrics for Adobe Times which were generated in 1994. Unfortunately, a major change to the fontinst macros took place in mid-1995, resulting in different stretch and shrink values in all the PostScript font metrics distributed as PSNFSS. Since Alan's article deals explicitly with the effects of changing TeX's parameters relating to setting text, using the current Adobe Times PSNFSS metrics caused disastrous results, so we had to maintain a copy of the old metrics for this paper.

The production team has been experimenting with a pre-release of changes to dvips that allow automatic partial-downloading of Type1 fonts. The much smaller PostScript files produced are very convenient when they have to be transferred across a slow transatlantic ftp link. The changes to dvips were made by Sergey Lesenko, and are described in a paper which will appear in the 1995 proceedings issue. We hope that they will appear in standard dvips soon. Type1 versions of the CM fonts are now used as standard to avoid printing complications on different devices.

Output

Though individual articles were worked on by members of the production team on their local computer systems, the final output was prepared at SCRI on an IBM RS6000 running AIX, using the *Web2C* implementation of TeX. Output was printed on a QMS 680 print system at 600 dpi.

Future Issues

The next issue will be a theme issue and will be guest-edited by Malcolm Clark. 16(3) will be the TUG'95 proceedings issue, and we plan for 16(4) to be a bilingual issue featuring articles in both Russian and English.

Topics for future theme issues will be announced as plans become firm. Suggestions are welcome for prospective topics and guest editors. Send them to the Editor, Barbara Beeton (see address on page 3), or via electronic mail to `TUGboat@ams.org`.

Coming Next Issue

Guest-edited issue

The next issue of *TUGboat*, guest-edited by Malcolm Clark, focuses on 'portable' electronic documents. It contains articles on the Standard Generalized Markup Language, bringing in its relationship to HTML (Hypertext Markup Language) and the World Wide Web. The other strands are Adobe's Portable Document Format (a hypertext-capable version of PostScript, and more), which can be generated from existing TeX and LaTeX documents, and packages which may be included with LaTeX to produce hypertexts suitable for reading at a screen, rather than paper. The brave new world it heralds is one where the tyranny of paper is broken, and all 'documents' are truly virtual. Xanadu looms through the mists!

- *A Practical Introduction to SGML*
  Michel Goossens and Janne Saarela
- *From LaTeX to HTML and Back*
  Michel Goossens and Janne Saarela
- *The Inside Story of Life at Wiley with SGML, LaTeX and Acrobat*
  Geeti Granger
- *LaTeX, HTML and PDF,* or *the entry of TeX into the world of hypertext*
  Yannis Haralambous and Sebastian Rahtz
- *HTML & TeX: Making them sweat*
  Peter Flynn
- *The Hyperlatex Story*
  Otfried Schwarzkopf
- *The Los Alamos E-print Archives: HyperTeX in Action*
  Mark D. Doyle

Institutional Members

The Aerospace Corporation,
El Segundo, *California*

∗ Air Force Institute of Technology,
Wright-Patterson AFB, *Ohio*

American Mathematical Society,
Providence, *Rhode Island*

∗ ArborText, Inc.,
Ann Arbor, *Michigan*

∗ Brookhaven National Laboratory,
Upton, *New York*
Pasadena, California¡

CNRS - IDRIS,
Orsay, *France*

CERN, *Geneva*, *Switzerland*

∗ College Militaire Royal de Saint
Jean, *St. Jean*, *Quebec*, *Canada*

College of William & Mary,
Department of Computer Science,
Williamsburg, *Virginia*

Communications
Security Establishment,
Department of National Defence,
Ottawa, *Ontario*, *Canada*

CSTUG, *Praha*, *Czech Republic*

Elsevier Science Publishers B.V.,
Amsterdam, *The Netherlands*

∗ Fermi National Accelerator
Laboratory, *Batavia*, *Illinois*

Florida State University,
Supercomputer Computations
Research, *Tallahassee*, *Florida*

Grinnell College,
Noyce Computer Center,
Grinnell, *Iowa*

Hong Kong University of
Science and Technology,
Department of Computer Science,
Hong Kong

Institute for Advanced Study,
Princeton, *New Jersey*

Institute for Defense Analyses,
Communications Research
Division, *Princeton*, *New Jersey*

Iowa State University,
Ames, *Iowa*

Los Alamos National Laboratory,
University of California,
Los Alamos, *New Mexico*

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, *Wisconsin*

Mathematical Reviews,
American Mathematical Society,
Ann Arbor, *Michigan*

∗ Max Planck Institut
für Mathematik,
Bonn, *Germany*

New York University,
Academic Computing Facility,
New York, *New York*

Nippon Telegraph &
Telephone Corporation,
Basic Research Laboratories,
Tokyo, *Japan*

∗ Personal TEX, Incorporated,
Mill Valley, *California*

Princeton University,
Princeton, *New Jersey*

Smithsonian Astrophysical
Observatory, *Cambridge*,
Massachusetts

Space Telescope Science Institute,
Baltimore, *Maryland*

Springer-Verlag,
Heidelberg, *Germany*

∗ Stanford Linear Accelerator
Center (SLAC),
Stanford, *California*

Stanford University,
Computer Science Department,
Stanford, *California*

Texas A & M University,
Department of Computer Science,
College Station, *Texas*

∗ United States Naval
Postgraduate School,
Monterey, *California*

United States Naval Observatory,
Washington DC

University of California, Berkeley,
Center for EUV Astrophysics,
Berkeley, *California*

University of California, Irvine,
Information & Computer Science,
Irvine, *California*

University of Canterbury,
Christchurch, *New Zealand*

University College,
Cork, *Ireland*

University of Delaware,
Newark, *Delaware*

University of Groningen,
Groningen, *The Netherlands*

Universität Koblenz–Landau,
Koblenz, *Germany*

University of Manitoba,
Winnipeg, *Manitoba*

University of Oslo,
Institute of Informatics,
Blindern, *Oslo*, *Norway*

∗ University of Southern California,
Information Sciences Institute,
Marina del Rey, *California*

University of Stockholm,
Department of Mathematics,
Stockholm, *Sweden*

University of Texas at Austin,
Austin, *Texas*

Università degli Studi di Trieste,
Trieste, *Italy*

Uppsala University,
Uppsala, *Sweden*

Vrije Universiteit,
Amsterdam, *The Netherlands*

Wolters Kluwer,
Dordrecht, *The Netherlands*

Yale University,
Department of Computer Science,
New Haven, *Connecticut*

(52 institutions listed)

TEX Consulting & Production Services

Information about these services can be obtained from:

TEX Users Group
1850 Union Street, #1637
San Francisco, CA 94123, U.S.A.
Phone: +1 415 982-8449
Fax: +1 415 982-8559
Email: `tug@tug.org`

North America

Anagnostopoulos, Paul C.
Windfall Software,
433 Rutland Street, Carlisle, MA 01741;
(508) 371-2316; `greek@windfall.com`
We have been typesetting and composing high-quality books and technical Publications since 1989. Most of the books are produced with our own public-domain macro package, ZzTEX, but we consult on all aspects of TEX and book production. We can convert almost any electronic manuscript to TEX. We also develop book and electronic publishing software for DOS and Windows. I am a computer analyst with a Computer Science degree.

Cowan, Dr. Ray F.
141 Del Medio Ave. #134, Mountain View, CA 94040;
(415) 949-4911; `rfc@netcom.com`
Twelve Years of TEX and Related Software Consulting: Books, Documentation, Journals, and Newsletters
TEX & LATEX macropackages, graphics; PostScript language applications; device drivers; fonts; systems.

Hoenig, Alan
17 Bay Avenue, Huntington, NY 11743; (516) 385-0736
TEX typesetting services including complete book production; macro writing; individual and group TEX instruction.

NAR Associates
817 Holly Drive E. Rt. 10, Annapolis, MD 21401;
(410) 757-5724
Extensive long term experience in TEX book publishing with major publishers, working with authors or publishers to turn electronic copy into attractive books. We offer complete free lance production services, including design, copy editing, art sizing and layout, typesetting and repro production. We specialize in engineering, science, computers, computer graphics, aviation and medicine.

Ogawa, Arthur
40453 Cherokee Oaks Drive,
Three Rivers, CA 93271-9743;
(209) 561-4585
Experienced in book production, macro packages, programming, and consultation. Complete book production from computer-readable copy to camera-ready copy.

Quixote Digital Typography, Don Hosek
555 Guilford, Claremont, CA 91711;
(909) 621-1291; Fax: (909) 625-1342;
`dhosek@quixote.com`
Complete line of TEX, LATEX, and METAFONT services including custom LATEX style files, complete book production from manuscript to camera-ready copy; custom font and logo design; installation of customized TEX environments; phone consulting service; database applications and more. Call for a free estimate.

Richert, Norman
1614 Loch Lake Drive, El Lago, TX 77586;
(713) 326-2583
TEX macro consulting.

Type 2000
16 Madrona Avenue, Mill Valley, CA 94941;
(415) 388-8873; Fax: (415) 388-8865
`pti@crl.com`
$2.50 per page for 2000 DPI TEX and PostScript camera ready output! We provide high quality and fast turnaround to dozens of publishers, journals, authors and consultants who use TEX. Computer Modern, PostScript and METAFONT fonts available. We accept DVI and PostScript files only and output on RC paper. $2.25 per page for 100+ pages, $2.00 per page for 500+ pages; add $.50 per page for PostScript.

Outside North America

TypoTEX Ltd.
Electronical Publishing, Battyány u. 14. Budapest,
Hungary H-1015; (036) 11152 337
Editing and typesetting technical journals and books with TEX from manuscript to camera ready copy. Macro writing, font designing, TEX consulting and teaching.