A Practical Introduction to SGML

Michel Goossens and Janne Saarelal/CD-ASD

Presented at TUG95, Saint-Petersburg, Florida, July 23-28, 1995 and
EuroTeX95, Papendaal, Netherlands, September 4-8, 1995

To be published in TUGboat, 16-3 (1995)
A Practical Introduction to SGML
Michel Goossens and Janne Saarela, CERN, CN Division, CH-1211 Geneva 23, Switzerland

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

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

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

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

To recapitulate, the strong points of a generic markup (in SGML) are the following:
- the quality of the source document is improved;
- the document can be used more rationally, resulting in an improved life-cycle;
- the publishing costs are reduced;
- the information can be easily reused, yielding an added value to the document (printed, hypertext, data base).

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

A big step forward was the publication by ISO (the International Standards Organization, with headquarters in Geneva, Switzerland) in October 1986 of SGML as Standard ISO8879 (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 who 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 12083.

Another application actively developed during the last few years is the CALS (Computer-aided Acquisition and Logistic Support) initiative of the American Department of Defense (DoD). This initiative aims at the replacement of paper documents by electronic media for the documentation of all arms systems. The DoD decided that all documentation
must be marked up in SGML, thus also making (the frequent) revisions a lot easier.

A few other examples of the use of SGML are:

- 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 and the British Patents Office (HMSO);
- Oxford University Press and Virginia Polytechnic (PhD, USA);
- the Text Encoding Initiative (classic texts and comments);
- the technical documentation of many major computer manufacturers or scientific publishers, for instance the DocBook or other dedicated DTDs used by IBM, HP, OSF, O'Reilly, etc.
- many text processing and data base applications have SGML input/output modules (filters), for example, Frame, Interleaf, Microsoft, Oracle, Wordperfect;
- McGraw-Hill (Encyclopedia of Science and Technology);
- the electronics industry (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, Exoterica, Grif, Softquad;
- and, of course, HTML and www!

2 SGML Basic Principles

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

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

2.1 Different Types of Markup

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

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

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

2.2 Generalized Logical Markup

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

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

---

1 See also the "SGML Web Page" at the URL http://www.si.1.org/sgml/sgml.html for more information on who uses SGML and why.
Specific markup

\TeX

\lbrack \texttt{\vfil} \texttt{\eject}\rbrack
\par \noindent (\bf Chapter 2: Title of Chapter)
\par \vskip \baselineskip

\textbf{Script}

\texttt{.pa}
\texttt{.bd Chapter 2: Title of Chapter}
\texttt{.sp}
\chapter{Title of Chapter}\par
\HTML (SGML)

\texttt{<H1>Title of Chapter</H1>}
\texttt{<P>}

\textbf{Figure 1: Different kinds of markup}

<table>
<thead>
<tr>
<th>Article A</th>
<th>Article B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Title</td>
</tr>
<tr>
<td>Section 1</td>
<td>Section 1</td>
</tr>
<tr>
<td>Subsection 1.1</td>
<td>Subsection 1.1</td>
</tr>
<tr>
<td>Subsection 1.2</td>
<td>Subsection 1.2</td>
</tr>
<tr>
<td>Section 2</td>
<td>Section 2</td>
</tr>
<tr>
<td>Subsection 3.1</td>
<td>Subsection 2.1</td>
</tr>
<tr>
<td>Subsection 3.2</td>
<td>Subsection 2.2</td>
</tr>
<tr>
<td>Subsection 3.3</td>
<td>Subsection 3.4</td>
</tr>
<tr>
<td>Bibliography</td>
<td>Bibliography</td>
</tr>
</tbody>
</table>

\textbf{Figure 2: Two instances of the same document class "article"}

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

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

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

\texttt{<article>}
\texttt{<tit>An introduction to SGML</tit>}
\texttt{<sec>SGML: the basic principles</sec>}
\texttt{<P>...}
\texttt{<sec>Generalized logical markup</sec>}
\texttt{<P>...}

\textbf{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 29.)

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. For instance when "sec" identifies the element "Section", then in the document source <sec> is the tag marking the beginning of a Section, with the delimiters "<" and ">", indicating, respectively, the tag start and end. Similarly, the formal structure of the document (described by the DTD) has its own language defined by the SGML Standard.

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

4 The Structure of a DTD

To better understand how SGML works we propose to examine a real example of a modern SGML application, namely HTML level 2, which corresponds to the functionality offered by popular HTML viewing programs, such as Mosaic, Netscape or Lynx. The complete DTD of HTML2 is shown in Appendix B starting on page 22. 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 29. 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 ", see Lines 37-39, and 303-305).

4.1 Comments

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

An SGML comment has the form:

<!-- text of the comment -->

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

4.2 The Elements

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

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

3 The form used in the DTD at line 616 uses a parameter entity, see Section 4.4.
Section 4.2.3.

The parameter entity 'Lf1o¤ can be found on Line 313, see also Section 4.2.3, see especially Table 1; 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.).

4.2.3 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 (|), 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:

```
<ENTITY Z head_extzm>
```

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>

```
<ENTITY HTML 0 0 (HEAD, BODY)>
```

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

4.2.2 Omitting Tags

It is possible that under certain circumstances one can infer automatically from the context that an omitted tag is present. This possibility must be declared for each element between the element’s name and its content model in the form of two blank separated characters, corresponding, respectively, to the omitted character’s 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.)

```
<ENTITY Z head_extr>
```

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.

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 (|), 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:

```
<ENTITY DL U %flow>
```

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?).

4.2.2 Omitting Tags

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>.

```
<ENTITY HTML 0 0 (HEAD, BODY)>
```

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:

```
<ENTITY TITLE - - (#PCDATA)>
```

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

```
<ENTITY CDATA replaceable 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:

```
<ENTITY TITLE - - (#PCDATA)>
```

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:

```
<ENTITY TEL CDATA>
```

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

<table>
<thead>
<tr>
<th>symbol</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>all must appear and in the order indicated (ordered &quot;and&quot;)</td>
</tr>
<tr>
<td>&amp;</td>
<td>all must appear but any order is allowed (unordered &quot;and&quot;)</td>
</tr>
<tr>
<td></td>
<td>one and only one can appear (exclusive &quot;or&quot;)</td>
</tr>
<tr>
<td>+</td>
<td>element must appear once or more</td>
</tr>
<tr>
<td>?</td>
<td>optional element (0 or one)</td>
</tr>
<tr>
<td>*</td>
<td>element can appear once or more</td>
</tr>
</tbody>
</table>

Table 1: Order and choice operators
Entities can be used for the following purposes:

- The definitions of abbreviated notations to ease repetitive text strings (general entities); for example,
  \[\text{<!ENTITY TUG "\TeX\{} Users Group"} \]

- 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;
be preceded by the keyword "SYSTEM", for exam-

ple, for the UNIX operating system one might have a declaration of the form:

```
<!ENTITY article SYSTEM "/usr/goossens/tug/sgmlart.sgm1">
```

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

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

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

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

### 4.5 Other DTDs

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

#### 4.5.1 HTML3

As its name indicates, HTML3 is a successor to the present HTML Standard (also known as HTML2, and discussed in detail in the previous sections). HTML3 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.

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

Mathematics support for equations and formulae in HTML3 mainly uses T\TeX's box paradigm. The implementation uses a simple markup scheme, that is still powerful enough to cope with over 90% of the most common cases. Filters from T\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 &em-space;_, horizontal tabs, and alignment of headers and paragraph text.

---

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

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

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

To illustrate the use of this DTD one can look at the table and mathematics parts of the HTML3 DTD (see Appendix F on page 32) and at the markup examples and the 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.

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

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 12083

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

The ISO 12083 Table Model

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

\[
\begin{align*}
<table> & \text{the table element;} \\
<np> & \text{number;} \\
<title> & \text{title;} \\
<body> & \text{table body;} \\
<head> & \text{head;} \\
<subhead> & \text{table subhead;} \\
<row> & \text{row;} \\
<stub> & \text{table stub;} \\
<cell> & \text{cell.}
\end{align*}
\]

This table model does not support spanning rows or columns. It does, however, support subhead elements that can be used to give more granularity.

\(^7\) They can be found at the URL http://www.sil.org/sgml/gen-apps.html\#is12083DTDs.
Figure 3: HTML3 example of tables (source and result with the Mosaic browser)

Figure 4: HTML3 example of simple mathematics (source and result with the arena browser)
The mathematics model in ISO 12083 consists of the following element categories:

- **Character transformations**: \(<\text{bold}>, \langle\text{italic}>, \langle\text{sansser}>, \langle\text{typewrit}>, \langle\text{smallcap}>, \langle\text{roman}>;\)
- **Fractions**: \(<\text{fraction}>, \langle\text{num}>, \langle\text{den}>;\)
- **Superiors, inferiors**: \(<\text{sup}>, \langle\text{inf}>;\)
- **Embellishments**: \(<\text{top}>, \langle\text{middle}>, \langle\text{bottom}>;\)
- **Fences, boxes, overlines and underlines**: \(<\text{mark}>, \langle\text{fence}>, \langle\text{post}>, \langle\box>, \langle\overline{\text{line}}>, \langle\underline{\text{line}}>;\)
- **Roots**: \(<\text{radical}>, \langle\text{radix}>, \langle\text{radicand}>;\)
- **Arrays**: \(<\text{array}>, \langle\text{arrayrow}>, \langle\text{arraycol}>, \langle\text{arraycell};\)
- **Spacing**: \(<\text{hspace}>, \langle\vspace>, \langle\break>, \langle\markref>;\)
- **Formulas**: \(<\text{formula}>, \langle\text{dformula}>, \langle\text{dformgrp};\)

The ISO 12083 Mathematics Model

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

- **Character transformations**: \(<\text{bold}>, \langle\text{italic}>, \langle\text{sansser}>, \langle\text{typewrit}>, \langle\text{smallcap}>, \langle\text{roman}>;\)
- **Fractions**: \(<\text{fraction}>, \langle\text{num}>, \langle\text{den}>;\)
- **Superiors, inferiors**: \(<\text{sup}>, \langle\text{inf}>;\)
- **Embellishments**: \(<\text{top}>, \langle\text{middle}>, \langle\text{bottom}>;\)
- **Fences, boxes, overlines and underlines**: \(<\text{mark}>, \langle\text{fence}>, \langle\text{post}>, \langle\box>, \langle\overline{\text{line}}>, \langle\underline{\text{line}}>;\)
- **Roots**: \(<\text{radical}>, \langle\text{radix}>, \langle\text{radicand}>;\)
- **Arrays**: \(<\text{array}>, \langle\text{arrayrow}>, \langle\text{arraycol}>, \langle\text{arraycell};\)
- **Spacing**: \(<\text{hspace}>, \langle\vspace>, \langle\break>, \langle\markref>;\)
- **Formulas**: \(<\text{formula}>, \langle\text{dformula}>, \langle\text{dformgrp};\)

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 \(<\text{left}>\) and \(<\text{right}>\) elements. A simple example is:

\[
S = \sum_{n=1}^{10} \text{sup}^1 \text{num}^1 \text{den}^1
\]

The complete DTD is shown in Appendix G on page 37, which shows the file math.dtd that is part of the ISO 12083 DTD set.

5 | SGML Editors

Several solutions exist to enter SGML or HTML markup into a document, but an editor that is SGML-aware is probably the best solution. Several (mostly commercial) products exist (see 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.

\(^8\) The psgml home page is at the URL http://www.lrysator.liu.se/projects/about_psgml.html.
5.2 Symposia

At the Third International World Wide Web Conference "Technology, Tools and Applications"\textsuperscript{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\textsuperscript{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.

6 SGML Utilities

As SGML is now actively used in many applications in the field of document production (see Section 1.2

\textsuperscript{9} See the documentation coming with psgml for more details.

\textsuperscript{10} An overview of the papers is at the URL http://www.igd.fhg.de/www95/papers/.

\textsuperscript{11} Their paper is available at the URL http://www.igd.fhg.de/www95/papers/04/EditHTML.html.
Figure 8: Inserting text in an SGML document with Symposia

Figure 9: SGML source of the document shown in Figure 8
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 sp13, a system developed by James Clark (jjc@clark.com), and a successor to his older sgmls and arcsgml, written by Charles Goldfarb, who is considered by many as the father of SGML, and who is also the author of “The SGML Handbook” (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 [ -deglsrusvux ] [ -alinktype ]
[ -ffile ] [ -iname ] [ -mfile ]
[ -tfile ] [ -warning_type ]
[ filename... ]
```

nsgmls needs at least four files to run:

- the catalog file, which describes how the SGML file's `<!DOCTYPE>` declaration is mapped to a filename (see below);
- the SGML declaration, defining the character set used by subsequent files, and the sizes of various internal limits, such as the permitted length of identifiers, as well as what features of SGML are used, such as tag minimization (see the start of Section 4 on page 4 and Appendix C on page 29);
- the DTD for the document type;
- an SGML or HTML document instance.

13 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 public SGML software list at the URL http://www.sil.org/sgml/publicS.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.

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"
/usr/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.

```xml
<!DOCTYPE html PUBLIC
"-//IETF//DTD HTML 2.0//EN">
<html>
<title>This is document test1.html</title>
<head>
<title>Document test1.html</title>
</head>
<body>

-<dt>term 1</dt>
<dd data 1</dd>
<dt>term 2</dt>
<dd data 2</dd>
<dt>term 3</dt>
<dd data 4</dd>
<dt>term 4</dt>
<dd data 4 bis</dd>

</body>
</html>
```

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- `n` a record end;
- `\|` \n brackets end;
- `\#{}` 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` references 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:

```xml
<!DOCTYPE html PUBLIC
"-//IETF//DTD HTML//EN">
<html>
<body>
<p>text inside a paragraph</p>
</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:

```bash
> nsgmls -m catalog sgml.decl test2.html
```

as follows.

```xml
<!DOCTYPE html PUBLIC
"-//IETF//DTD HTML 2.0//EN">
<html>
<body>
```

When reading the output, the command output is not included.
6.3 Prettyprinting

Nelson Beebe (beebe@math.utah.edu) has developed a program htmlpt
ty\footnote{It is at URL ftp://ftp.math.utah.edu/pub/misc/htmlpty-x.y.y.tar.gz (choose the latest version x.y.z offered).}, 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 -i 2 -n test1.html
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML//EN">

<!-- This is document doc1.sgm -->
<head>
  <title>title</title>
</head>
<body>
  <!-- Beginning of body of document -->
  
</body>
</html>
```

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

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

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 `<UL>`, `<OL>`, `<MENU>`, and `<DIR>`).

```html
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head>
<title></title>
</head>
<body>
</body>
</html>
```
6.4 SGML document analysis tools

Earl Hook (ehood@convex.com) has developed a set of tools per1SGML\(^\text{15}\), 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;
- **dtdiff** compares two DTDs and shows possible differences;
- **dttree** 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 dttree 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 (I) and (Ia)—the content model of the element. Figure 10 shows in four columns the (condensed) output generated by the dttree program when treating the HTML2 DTD. For more clarity most of the repeated blocks have been eliminated and replaced by the string "*\*\*" at the beginning of a line and a few lines have been cut to make them fit (marked with \*\*\* at the end of the line).

### 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*.

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

### 6.5 Searching and index entries

A search engine for regular expressions for use with the HTML2 DTD is available\(^\text{16}\) (Figure 12), as well as an index with more than 1100 entries and phrases\(^\text{17}\) (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 htmlcheck\(^\text{18}\), including heuristic checkers for common style and grammar violations. The programs are available in both perl and awk versions and syntactically check HTML2 and HTML3 files for a number of possible errors; they can perform local link cross-reference verification, and generate a rudimentary reference-dependency map.

**htmlcheck** checks an HTML file for errors, and giving warnings about possible problems;

---

\(^{15}\) This system can be found at the url ftp://ftp.uci.edu/pub/dtd2html.

\(^{16}\) http://hopf.math.nwu.edu/html2.0/dosearch.html.

\(^{17}\) http://hopf.math.nwu.edu/html2.0/docindex.html.

Figure 10: Output of the dtdtree program for the HTML2 DTD
Figure 13: Index entries for the HTML2 DTD

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

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

dehmtl 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 typically:

```
> perl htmlcheck.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 1 data 1
term 2 data 2
term 3 data 4 data 4 bis

> perl -f htmlcheck.pl test2.html
Diagnostic for file "test2.html":
<body> without preceding <head>...
Warning! at line 4 of file "test2.html"
Bo <H1> in <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 IDDOCTYPE occurred

> perl htmlcheck.pl test3.html
Diagnostic for file "test3.html":
<CLOSED> outside of list
ERROR! at line 8 of file "test3.html"
Bo <H1> in <body>...
Warning! at line 9 of file "test3.html"
Tag IDDDOCTYPE occurred
Tag BODY occurred
Tag HEAD occurred
Tag HTML occurred
Tag Li occurred
Tag TITLE occurred
```

7 DTD Transformations

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

7.1 sgm1s.pl
A translator skeleton, sgm1s.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 sgm1s.pl
David Megginson (University of Ottawa, Canada, dme@uwaterloo.ca) has developed a more object-oriented approach for the translations, also based on the ESIS output of nsgmls and calling event-routines for each element found in the input stream. This package includes a default configuration for translating documents marked up according to the DocBook DTD into HTML or \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 DocBook to HTML3
The translation program generates events for each primitive in the source document and these events are handled by calling a corresponding routine. These routines then produce the corresponding HTML/\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.

```perl
# Program listings appear in verbatim
sgml('<PROGRAMLISTING>', "\begin{verbatim}
")
sgml('</PROGRAMLISTING>', "\end{verbatim}\n")

# Class names appear in typewriter.
sgml('<CLASSNAME>', "{\ttfamily }");
sgml('</CLASSNAME>', "\}"
);
```

A conversion example of an extract from the DocBook DTD manual is given in Appendix H on page 40. It shows part of the original DocBook document markup, how it is presented in the ESIS format, finally its translation in 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 few.

8 Other Standards in the Area of Electronic Documents
SGML is part of a vast project conceived by the International Standards Organisation (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 10179 (ISO 1994), 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.\(^{(10)}\)

**SPDL (Standard Page Description Language)**

Draft International Standard ISO DIS 10180 (ISO 1991) defines a formalism for the description of documents in their final, completely typeset, unreviewable 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 9541 (ISO 1991a) 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 manuscript. His suggestions and hints have without doubt substantially improved the quality of the text. We also want to acknowledge the help of Steven Kennedy (CERN) who proofread the article.

**References**


---


\(^{(20)}\) More on SPDL can be found at the URL [http://www.strim.or.jp/uda/spdl/spdl.html](http://www.strim.or.jp/uda/spdl/spdl.html).


Appendices

B The DTD of the HTML2 Language

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN" >

<!-- Typical usage: -->
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<!ENTITY X HTML.Version "HTML"/>
<!-- 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 these features. -->
<!ENTITY X HTML.Deprecated "IGNORE"/>
<!-- Certain features of the language are necessary for compatibility with earlier versions of the specification, but they tend to be used implemented inconsistently, and their use is deprecated. This feature test entity enables a document type definition that eliminates these features. -->
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no highlighting tags, which may be ignored on minimal implementations. -->
<!ENTITY X HTML.Forms "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no forms, which may not be supported in minimal implementations. -->
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no forms, which may not be supported in minimal implementations. -->
<!ENTITY X HTML.Forms "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no forms, which may not be supported in minimal implementations. -->
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!ENTITY X HTML.Forms "INCLUDE"/>
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!ENTITY X HTML.Deprecated "IGNORE"/>
<!-- Certain features of the language are necessary for compatibility with earlier versions of the specification, but they tend to be used implemented inconsistently, and their use is deprecated. This feature test entity enables a document type definition that eliminates these features. -->
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!-- Certain features of the language are necessary for compatibility with earlier versions of the specification, but they tend to be used implemented inconsistently, and their use is deprecated. This feature test entity enables a document type definition that eliminates these features. -->
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no highlighting tags, which may be ignored on minimal implementations. -->
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!-- Certain features of the language are necessary for compatibility with earlier versions of the specification, but they tend to be used implemented inconsistently, and their use is deprecated. This feature test entity enables a document type definition that eliminates these features. -->
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no highlighting tags, which may be ignored on minimal implementations. -->
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!-- Certain features of the language are necessary for compatibility with earlier versions of the specification, but they tend to be used implemented inconsistently, and their use is deprecated. This feature test entity enables a document type definition that eliminates these features. -->
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no highlighting tags, which may be ignored on minimal implementations. -->
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!-- Certain features of the language are necessary for compatibility with earlier versions of the specification, but they tend to be used implemented inconsistently, and their use is deprecated. This feature test entity enables a document type definition that eliminates these features. -->
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no highlighting tags, which may be ignored on minimal implementations. -->
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!-- Certain features of the language are necessary for compatibility with earlier versions of the specification, but they tend to be used implemented inconsistently, and their use is deprecated. This feature test entity enables a document type definition that eliminates these features. -->
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no highlighting tags, which may be ignored on minimal implementations. -->
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!-- Certain features of the language are necessary for compatibility with earlier versions of the specification, but they tend to be used implemented inconsistently, and their use is deprecated. This feature test entity enables a document type definition that eliminates these features. -->
<!ENTITY X HTML.Highlighting "INCLUDE"/>
<!-- Use this feature test entity to validate that a document contains no highlighting tags, which may be ignored on minimal implementations. -->
<!ENTITY X HTML.Deprecated "INCLUDE"/>
<!-- Certain features of the language are necessary for compatibility with earlier versions of the specification, but they tend to be used implemented inconsistently, and their use is deprecated. This feature test entity enables a document type definition that eliminates these features. -->
<!ENTITY X HTML.Highlighting "INCLUDE"/>
Note that CDATA attributes are limited by the LITERAL capacity (1024 in the current version of html.decl), so that URIs in HTML have a bounded length.

Encoding for the Blind, +1 800 221 4792
Usenet news group bit.listserv.easi
ICADD ListServ
ICADD applications are designed to support accessible access to structured information by print-impaired individuals through Braille, large print and voice synthesis. For more information on the ICADD DTD in support of GREY transformation to the International Committee for Accessible Design (ICADD) DTD

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

Character encoding entities

SGML Document Access (SDA) Parameter Entities

---

ISO 12083:1993, Annex A.8, Facilities for Braille, large print and computer voice
ICADD ListServ<br>
<ICADDLISTSERV.BITNEWS@ARIZONAVCCIT.ARIZONA.EDU>

Recording for the Blind, +1 800 221 4792

---

Text Markup Entities
To be published in TUGboat, Volume 16 (1995), No. 3

188 <TT> Typesetter text
189 </TT>
190 <!ENTITY EM "Emphasized phrase"
191 <!ENTITY STRONG "Strong emphasis"
192 <!ENTITY CODE "Source code phrase"
193 <!ENTITY SAMPLE "Sample text or characters"
194 <!ENTITY PDK "Keyboard phrase, e.g. user input"
195 <!ENTITY VAR "Variable phrase or substitutable"
196 <!ENTITY CIT "Name or title of cited work"
197 <!ENTITY X pro.content "&PCDATA A | AB | &BK | %font | Xphrase">
198 </ENTITY>
199 <!ENTITY X text "&PCDATA A | AB | &BK | %text">
200 <!ELEMENT BR - O EMPTY>
201 <!ATTLIST BR XIDREF "#ID"/>
202 <!-- (A NAME="xxx">Heading</A></H1> -->
203 <!-- (A NAME="xxx">Heading</A></H1> -->
204 <!ENTITY X linkType "NAME">
205 <!-- a list of these will be specified at a later date -->
206 <!-- XIDREF "#ID" -->
207 <!-- (A NAME="xxx">Heading</A></H1> -->
208 <!-- (A NAME="xxx">Heading</A></H1> -->
209 <!-- (A NAME="xxx">Heading</A></H1> -->
210 </ELEMENT>
211 </ENTITY>
212 <!-- (A NAME="xxx">Heading</A></H1> -->
213 <!-- <CITE> Name or title of cited work -->
214 <!-- <VAR> Variable phrase or substitutable -->
215 <!-- <KBD> Keyboard phrase, e.g. user input -->
216 <!-- <SA> Sample text or characters -->
217 <!-- <CODE> Source code phrase -->
218 <!-- <EMPH> Emphasized phrase -->
219 <!-- <B>

To be published in TUGboat, Volume 16 (1995), No. 3

188 <TT> Typesetter text
189 </TT>
190 <!ENTITY EM "Emphasized phrase"
191 <!ENTITY STRONG "Strong emphasis"
192 <!ENTITY CODE "Source code phrase"
193 <!ENTITY SAMPLE "Sample text or characters"
194 <!ENTITY PDK "Keyboard phrase, e.g. user input"
195 <!ENTITY VAR "Variable phrase or substitutable"
196 <!ENTITY CIT "Name or title of cited work"
197 <!ENTITY X pro.content "&PCDATA A | AB | &BK | %font | Xphrase">
198 </ENTITY>
199 <!ENTITY X text "&PCDATA A | AB | &BK | %text">
200 <!ELEMENT BR - O EMPTY>
201 <!ATTLIST BR XIDREF "#ID"/>
202 <!-- (A NAME="xxx">Heading</A></H1> -->
203 <!-- (A NAME="xxx">Heading</A></H1> -->
204 <!ENTITY X linkType "NAME">
205 <!-- a list of these will be specified at a later date -->
206 <!-- XIDREF "#ID" -->
207 <!-- (A NAME="xxx">Heading</A></H1> -->
208 <!-- (A NAME="xxx">Heading</A></H1> -->
209 <!-- (A NAME="xxx">Heading</A></H1> -->
210 </ELEMENT>
211 </ENTITY>
212 <!-- (A NAME="xxx">Heading</A></H1> -->
213 <!-- <CITE> Name or title of cited work -->
214 <!-- <VAR> Variable phrase or substitutable -->
215 <!-- <KBD> Keyboard phrase, e.g. user input -->
216 <!-- <SA> Sample text or characters -->
217 <!-- <CODE> Source code phrase -->
218 <!-- <EMPH> Emphasized phrase -->
219 <!-- <B>

To be published in TUGboat, Volume 16 (1995), No. 3

188 <TT> Typesetter text
189 </TT>
190 <!ENTITY EM "Emphasized phrase"
191 <!ENTITY STRONG "Strong emphasis"
192 <!ENTITY CODE "Source code phrase"
193 <!ENTITY SAMPLE "Sample text or characters"
194 <!ENTITY PDK "Keyboard phrase, e.g. user input"
195 <!ENTITY VAR "Variable phrase or substitutable"
196 <!ENTITY CIT "Name or title of cited work"
197 <!ENTITY X pro.content "&PCDATA A | AB | &BK | %font | Xphrase">
198 </ENTITY>
199 <!ENTITY X text "&PCDATA A | AB | &BK | %text">
200 <!ELEMENT BR - O EMPTY>
201 <!ATTLIST BR XIDREF "#ID"/>
202 <!-- (A NAME="xxx">Heading</A></H1> -->
203 <!-- (A NAME="xxx">Heading</A></H1> -->
204 <!ENTITY X linkType "NAME">
205 <!-- a list of these will be specified at a later date -->
206 <!-- XIDREF "#ID" -->
207 <!-- (A NAME="xxx">Heading</A></H1> -->
208 <!-- (A NAME="xxx">Heading</A></H1> -->
209 <!-- (A NAME="xxx">Heading</A></H1> -->
210 </ELEMENT>
211 </ENTITY>
212 <!-- (A NAME="xxx">Heading</A></H1> -->
213 <!-- <CITE> Name or title of cited work -->
214 <!-- <VAR> Variable phrase or substitutable -->
215 <!-- <KBD> Keyboard phrase, e.g. user input -->
216 <!-- <SA> Sample text or characters -->
217 <!-- <CODE> Source code phrase -->
218 <!-- <EMPH> Emphasized phrase -->
219 <!-- <B>
<!-- <!ENTITY % literal "CDATA" -->
<!-- historical, non-conforming parsing mode where -->
<!-- the only markup signal is the end tag -->
in full

<!-- <!-- <PRE> Preformatted text -->
<!-- <PRE WIDTH=...> Maximum characters per line -->
<!ENTITY X "literal">
<ATTLIST MENU
  COMPACT (COMPACT) #IMPLIED
  XSDAFORM: "List">
<ATTLIST LI
  XSDAFORM: "List">
<ATTLIST BLOCKQUOTE
  XSDAFORM: "BQ">
<ATTLIST ADDRESS
  XSDAFORM: "Address"
  XSDAPREF: "Address">
<ATTLIST BLOCKQUOTE
  XSDAPREF: "BQ">
<ATTLIST FORM
  XSDAPREF: "<Head>Directory</Head>">
<ATTLIST MENU
  COMPACT (COMPACT) #IMPLIED
  XSDAFORM: "List">
<ATTLIST LI
  XSDAFORM: "List">
<ATTLIST BLOCKQUOTE
  XSDAFORM: "BQ">
<ATTLIST ADDRESS
  XSDAFORM: "Address"
  XSDAPREF: "Address">
<ATTLIST BLOCKQUOTE
  XSDAPREF: "BQ">
<ATTLIST FORM
  XSDAPREF: "<Head>Directory</Head>">
SGML Declaration for HyperText Markup Language (HTML)

<!DOCTYPE SGML SYSTEM "ISO 8879:1986">

--

SGML Declaration for HyperText Markup Language (HTML).

--

CHARSET

BASESET "ISO 646:1985//CHARSET

International Reference Version

(IN)///ESC 2/5 4/0"
To be published in TUGboat, Volume 16 (1995), No. 3
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.

```xml
<!ENTITY Acute SDATA "[acute]"—=capital A, acute accent—>
<!ENTITY Acute SDATA "[acute]"—=small a, acute accent—>
<!ENTITY Acirc SDATA "[a]"—=capital A, circumflex accent—>
<!ENTITY Acirc SDATA "[a]"—=small a, circumflex accent—>
<!ENTITY Agrave SDATA "[a]"—=capital A, grave accent—>
<!ENTITY Agrave SDATA "[a]"—=small a, grave accent—>
<!ENTITY Atilde SDATA "[~]"—=capital A, tilde—>
<!ENTITY Atilde SDATA "[~]"—=small a, tilde—>
```

The ISO-Latin1 Entity Set

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

```xml
<!-- catalog: SGML Open style entity catalog for HTML -->
<!ENTITY Acute SDATA "[acute]"—=capital A, acute accent—>
<!ENTITY Acute SDATA "[acute]"—=small a, acute accent—>
<!ENTITY Acirc SDATA "[a]"—=capital A, circumflex accent—>
<!ENTITY Acirc SDATA "[a]"—=small a, circumflex accent—>
<!ENTITY Agrave SDATA "[a]"—=capital A, grave accent—>
<!ENTITY Agrave SDATA "[a]"—=small a, grave accent—>
<!ENTITY Atilde SDATA "[~]"—=capital A, tilde—>
<!ENTITY Atilde SDATA "[~]"—=small a, tilde—>
```

To be published in *TUGboat*, Volume 16 (1995), No. 3
Tables and figures can be aligned in several ways:

1. **bleedleft** flush left with the left (window) border
2. **left** flush left with the left text margin
3. **center** centered (text flow is disabled for this mode)
4. **right** flush right with the right text margin
5. **bleedright** flush right with the right (window) border
6. **justify** when applicable the table/figure should stretch to fill space between the text margins
7. 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 these further down the page.

To assist with formatting tables to paged media, authors
can differentiate leading and trailing rows that are to
duplicate when splitting tables across page boundaries. The recommended way is to subclass rows with the CLASS attribute
for example: &lt;TR CLASS=Header&gt;, &lt;TH CLASS=Footer&gt; 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.

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 alone 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: &lt;TR CLASS=Header&gt;, &lt;TH CLASS=Footer&gt; 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.
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.

Note that these only apply within the MATH element and can't be used in normal text!

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

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

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

<!ENTITY ZHTMLmath PUBLIC "-//IETF//ENTITIES Math and Greek for HTML//EN">
<!ENTITY zmath2HTML PUBLIC "-//IETF//ENTITIES Math and Greek for HTML//EN">

Use \hspace{1em} etc for greater control of spacing. --

Subscripts and Superscripts

\(X^{a+b}\) is mapped to \(<box>a + b</box>

\(y_z\) is mapped to \(y<sub>z</sub>

\(x'2\) is mapped to \(x<sup>2</sup>

to save typing when manually editing HTML math, e.g.

\(z^{x^2}\) is mapped to \(z<sup>x</sup><sup>2</sup>

\(y_{x+y}\) is mapped to \(y<sub>x+y</sub>

\(\langle x\rangle\) is mapped to \(<box>x</box>

Note that these only apply within the MATH element and can't be used in normal text!
The inclusion of Zmath and exclusion of Znotmath 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.

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 \( (\text{ and } \) ) as shorthand for BOX and \( \{\text{ } \}\) respectively:

\[
1 + x
\]

\[
\frac{a + b}{c - d}
\]

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

\[
[ a \times b ]
\]

Use \{brace; and \}brace; for \(\{\text{ and }\}\) respectively as these symbols are used as shorthand for BOX, e.g.

\[
\{ a \times b \}_{\text{RIGHT}} \]

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

\[
\int_a^b f(x) \, dx
\]

Note the complex content model for BOX in a work around for the absence of support for infix operators in SGML.
<table>
<thead>
<tr>
<th>Column Position</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>R</td>
</tr>
</tbody>
</table>

Use the COLDEF attribute to specify the column definitions. The COLDEF attribute allows authors to supply an entity name for each column determining how the columns should be aligned, e.g. `COLDEF="CCC"`. The ALIGI attribute alters the vertical position of the array as compared with preceding and following expressions.

Use COL1M1 and CB1M1 attributes for delimiter entities. When the LALIGN 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.

Note that for vertical, horizontal and diagonal ellipsis dots, use \textbullet; to fill an array cell with horizontal dots (e.g. for a full row).

When the LALIGN 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 \textbullet; to fill an array cell with horizontal dots (e.g. for a full row).

Note that the first element of the first row should normally be left blank.

Use the COLDEF attribute to specify the column definitions. The COLDEF attribute allows authors to supply an entity name for each column determining how the columns should be aligned, e.g. `COLDEF="CCC"`. The ALIGI attribute alters the vertical position of the array as compared with preceding and following expressions.

Use COL1M1 and CB1M1 attributes for delimiter entities. When the LALIGN 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.

Note that for vertical, horizontal and diagonal ellipsis dots, use \textbullet; to fill an array cell with horizontal dots (e.g. for a full row).

When the LALIGN 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 \textbullet; to fill an array cell with horizontal dots (e.g. for a full row).

Note that the first element of the first row should normally be left blank.
To be published in TUGboat, Volume 16 (1995), No. 3

G The ISO-12083 Mathematics DTD

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

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

<!-- The subform element is defined later -->
To be published in TUGboat, Volume 16 (1995), No. 3
H Example of a Conversion of the DocBook DTD to HTML3

H.1 The original document marked up in the DocBook DTD

The listing below is part of the manual describing the DocBook DTD and is tagged according to that same DocBook DTD (V2.1).

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

```
$ get docbook.2.2.1.shar
```

Most of these files also exist separately and may be ftp'd individually.

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

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQCT</td>
<td>TITLE</td>
<td>FILENAME</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| «how to get the DocBook DTD Online» | /pub/davenport/docbook | (FILENAME)
| \( <nitpick>at \)</nitpick> in | \( <nitpick>\text{-line ends and leading white space must be preserved in output}\)</nitpick> | \( <nitpick>/p\text{ub/davenport/docbook}\)
| \( <nitpick>\text{FILENAME}\)</nitpick> | \( <nitpick>/ftp.ora.com\)
| \( <nitpick>\text{FILENAME}\)</nitpick> | \( <nitpick>\text{(198.112.208.13)}\).
```

H.2 ESIS representation of the source document

The following is the ESIS representation of the same document produced by nsgmls.
The following presents the final HTML3 output resulting from the translation process.

**H.3 HTML3 output**

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). This sample session shows how to retrieve the DTD and its documentation:

```plaintext
%> cd pub/davenport/docbook
```

To be published in TUGboat, Volume 16 (1995), No. 3
The DocBook DTD and related ASCII files are in a file named `docbook.H.shar`, where `<STRONG>`H</STRONG>` is the current revision number:

```plaintext
ftp&gt;get docbook.2.2.1.shar
```

Most of these files also exist separately and may be ftp'd individually.

The `get` command will put this ASCII shar file on your system.

You must later unpack it on your system:

```plaintext
sh docbook.2.2.1.shar
```

To be published in TUGboat, Volume 16 (1995), No. 3