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 multinational 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),
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.

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>...</p>
  <ssec>Generalized logical markup</ssec>
  <p>...</p>
</article>
```

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. 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 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.

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

For instance Lines 614 and 616 are equivalent to the declaration:\footnote{The form used in the DTD at line 616 uses a parameter entity, see Section 4.4.4}

```xml
<!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 hyphens, “--”, indicating that the tag must be present (cannot be omitted), and an uppercase letter O “0” 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.):\footnote{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.}

```xml
<!ELEMENT (OL|UL) - - (LI)+>
```

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 “= 0” 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 (\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:

```xml
<!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 NEXCID tags, and zero or more META and LINK:

```xml
<!ELEMENT HEA D O O (%head.content>)
```

and

```xml
<!ENTITY % head.content

"TITLE & ISINDEX? & BASE? & LINK* % (

%head.content)"
```

```xml
<!ENTITY % head.extra

"NEXCID? & META* & LINK* %

%head.extra"
```

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:

<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 “and”)</td>
</tr>
<tr>
<td>k</td>
<td>all must appear but any order is allowed (unordered “and”)</td>
</tr>
<tr>
<td></td>
<td>one and only one can appear (exclusive “or”)</td>
</tr>
<tr>
<td>?</td>
<td>optional element (0 or one)</td>
</tr>
<tr>
<td>*</td>
<td>element can appear zero times or more</td>
</tr>
</tbody>
</table>

Table 1: Order and choice operators

and 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:

```xml
<!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:

```xml
<!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:

```xml
<!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;
- 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:

```xml
<!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.

```xml
<!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:

```xml
<!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:

```xml
<!ATTLIST IMG SRC %URI; #REQUIRED
```

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

Table 2: Keywords for attribute types
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 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,

```xml
<!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;

```xml
<!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:

```xml
&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:

```xml
<!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;

```xml
<!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.

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 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 formulæ 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 `\&emsp;`, horizontal tabs, and alignment of headers and paragraph text.

As well as this, many other often-requested features have been included, most notably a 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.

The primary goal in developing the DTD was to filter existing software documentation into SGML. It describes the structures 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.
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 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>
  <np>1</np><title>Capitals in Europe</title>
  <tbody>
    <row><cell>Helsinki</cell><cell>Finland</cell></row>
    <row><cell>Rome</cell><cell>Italy</cell></row>
    <row><cell>Bern</cell><cell>Switzerland</cell></row>
  </tbody>
</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&sup;10\frac{1}{\radical3\radix n}
</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 Grib-based Symposia editor.

5.1 Emacs and `psgml`

A major mode for editing SGML documents, `psgml`, 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
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.

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 smgls,13 or by arcsxml, 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 [ -deg1prsvux ] [ -alinktype ]
   [ -ffile ] [ -iname ] [ -mfile ]
   [ -tfile ] [ -warning_type ]
   [ filename... ]
```

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.
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 nsgmls 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 nsgmls 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.
Figure 8: Inserting text in an SGML document with Symposia

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

```xml
<!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.

```xml
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.

```html
<!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</dt>
  <dd data value="1">data 1</dd>
  <dt>term 2</dt>
  <dd data value="2">data 2</dd>
  <dt>term 3</dt>
  <dt>term 4</dt>
  <dd data value="4">data 4</dd>
  <dd data value="4 bis">data 4 bis</dd>
</dl>

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.

```
For incorrect documents \texttt{nsgmls} shows an error:

\begin{verbatim}
<!DOCTYPE html PUBLIC
"-//IETF//DTD HTML//EN">
<HTML>
<BODY>
 <P>text inside a paragraph
</BODY>
</HTML>
\end{verbatim}

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

\begin{verbatim}
> 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
\end{verbatim}

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

\begin{verbatim}
<!DOCTYPE html PUBLIC
"-//IETF//DTD HTML//EN">
<HTML>
<head>
<title>Document test HTML</title>
</head>
<body>
<dl>
<dt>term 1
<dd>data 1
<dt>term 2
<dd>data 2
</dd>
</dl>
</body>
</html>
\end{verbatim}

The program was run on file \texttt{test1.html} with the result shown below.

\begin{verbatim}
> html-pretty -i2 -n test1.html
<!DOCTYPE html PUBLIC
"-//IETF//DTD HTML//EN">
<HTML>
<head>
<title>This is document doc1.sgm --></title>
</head>
<body>
<dl>
<dt>term 1
<dd>data 1
<dt>term 2
<dd>data 2
</dd>
</dl>
</body>
</html>
\end{verbatim}

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

\begin{verbatim}
> nsgmls -s -m catalog sgml.decl test3.html
</verbatim}

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

\subsection{Prettyprinting}

Nelson Beebe (beebe@math.utah.edu) has developed a program \texttt{htmlfty},\footnote{It is at URL \url{ftp://ftp.math.utah.edu/pub/misc/htmlfty-x.yy.trz} (choose the latest version x.yz offered).} written in the lex and C languages, to prettyprint HTML files. Its calling sequence is:

\begin{verbatim}
htmlfty [-options] [file(s)]
\end{verbatim}

where the more interesting options are:

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

The program was run on file \texttt{test1.html} with the result shown below.

\begin{verbatim}
> html-pretty -i2 -n test1.html
<!DOCTYPE html PUBLIC
"-//IETF//DTD HTML//EN">
<HTML>
<head>
<title>This is document doc1.sgm --></title>
</head>
<body>
<dl>
<dt>term 1
<dd>data 1
<dt>term 2
<dd>data 2
</dt>
</dl>
</body>
</html>
\end{verbatim}
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,\(^\text{15}\) 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;
- **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 \(*\backslash vbar\backslash vbar\backslash vbar\) 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).

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**.

---

\(^{15}\) This system can be found at the URL ftp://ftp.uci.edu/pub/dtd2html.
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\(^\text{16}\) (Figure 12), as well as an index containing more than 1100 words and phrases\(^\text{17}\) (Figure 13).

6.5.1 Checking an HTML document

For those who do not have\texttt{sgmls} or\texttt{nsgmls} installed there exists a set of programs\texttt{htmlchek},\(^\text{18}\) including heuristic checkers for common style and grammar violations. The programs, available in both\texttt{perl} and\texttt{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.

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

The syntax to use these programs is typically:

\begin{verbatim}
awk -f htmlchek.awk [opts] infile > outfile
perl htmlchek.pl [opts] infile > outfile
\end{verbatim}

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.

\begin{verbatim}
> 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"
\end{verbatim}

\(^{16}\) http://hopf.math.nwu.edu/html2.0/dosearch.html.
\(^{17}\) http://hopf.math.nwu.edu/html2.0/docindex.html.

Figure 12: Searching the HTML2 DTD

Figure 13: Index entries for the HTML2 DTD
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 nsxml distribution. This perl script reads the ESIS output of nsxml and provides a set of routines that can be used for calling user-specified translation routines of each element.

7.2 SGMLS.pm and sgmls.pl

David Megginson (University of Ottawa, Canada, dmeiggs@aix1.uottawa.ca) has developed a more object-oriented approach for the translations, also based on the ESIS output of nsxml 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 \( \text{sp} \) parser provides an application level interface to SGML document handling. The core of \( \text{sp} \) 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.

```perl
<CLASSNAME>', "\{\tfamily \}=");
sgml('<CLASSNAME>', "}");
```

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

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

```perl
sgml('<\text{TITLE}>',
sub {
my $element = shift;
my $data = pop_output;
if ($element->in(ARTHEADER)) {
$title = $data;
} elsif ($element->in(SECT1) || $element->in(IMPORTANT)) {
output "\section{$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";
```
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 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.\footnote{More on DSSSL by James Clark is available at the URL http://www.jclark.com/dsssl/}

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.\footnote{More on SPDL can be found at the URL http://www.st.rim.or.jp/~uda/spdl/spdl.html} The structure of the language and its syntax strongly resemble the PostScript language, which is not surprising since PostScript has become the \textit{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.

\footnotetext[19]{More on DSSSL by James Clark is available at the URL http://www.jclark.com/dsssl/}
\footnotetext[20]{More on SPDL can be found at the URL http://www.st.rim.or.jp/~uda/spdl/spdl.html}
References


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 HTML Language

<!-- html.dtd

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


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:
<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.

-->

<!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
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.
<!DOCTYPE [ %HTML.Deprecated [ ] ]>
<!-- <XMP> Example section -->
<!-- <LISTING> Computer listing -->
<!ELEMENT PLAINTEXT - O %literal>
<!-- <PLAINTEXT> Plain text passage -->
<!ELEMENT (DL|UL) - - (LI)+>
<!ELEMENT (DIR|MENU) - - (LI)+ -(%block)>
<!ELEMENT (OL|UL) - - (LI)+>
<!ELEMENT (XMP|LISTING) - - %literal>
<!ATTLIST XMP %SDAFORM; "Lit"
%SDAPREF; "Example:&#RE;"
>
<!ATTLIST LISTING %SDAFORM; "Lit"
%SDAPREF; "Listing:&#RE;"
>
<!ELEMENT DT - O (%text)*>
<!ATTLIST DT %SDAFORM; "Term"
>
<!ELEMENT DD - O %flow>
<!ATTLIST DD %SDAFORM; "LItem"
>
<!ELEMENT (DT | DD)>
<!ATTLIST DL COMPACT (COMPACT) #IMPLIED
%SDAFORM; "List"
%SDAPREF; "Definition List:"
>
<!ELEMENT (OL | UL)>
<!ATTLIST OL COMPACT (COMPACT) #IMPLIED
%SDAFORM; "List"
>
<!ELEMENT (DIR | MENU)>
<!ATTLIST UL COMPACT (COMPACT) #IMPLIED
%SDAFORM; "List"
>
<!ELEMENT (OL | UL)>
<!ATTLIST OL COMPACT (COMPACT) #IMPLIED
%SDAFORM; "List"
>
<!ELEMENT (DIR | MENU)>
<!ATTLIST DIR COMPACT (COMPACT) #IMPLIED
%SDAFORM; "List"
<ENTITY % body.content "(heading|block|HR|ADDRESS|IMG)*" %HTML.Recommended [ ]>
<h1>Heading</h1>
<p>Text ...</p>
<!-- <FORM METHOD=...> Method of submitting form -->
<!-- <FORM ENCTYPE="..."> Representation of form data -->
<!-- <FORM ACTION="..."> Address for completed form -->
<!-- <FORM> Fill-out or data-entry form -->
<!-- <ADDRESS> Address, signature, or byline -->
</html>
<!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>"
>
<!ELEMENT OPTION - O (#PCDATA)*>
<!ATTLIST OPTION
 SELECTED (SELECTED) #IMPLIED
 VALUE CDATA #IMPLIED
 %SDAFORM; "LItem"
 %SDAPREF;
 "Option: #AttVal(Value) #AttVal(Selected)"
>
<!ELEMENT TEXTAREA - - (#PCDATA)* -(INPUT|SELECT|TEXTAREA)>
<!ATTLIST TEXTAREA
 NAME CDATA #REQUIRED
 ROWS NUMBER #REQUIRED
 COLS NUMBER #REQUIRED
 %SDAFORM; "Para"
 %SDAPREF; "Input Text -- #AttVal(Name):
 >

<!--======= Document Head ======================-->
<! [ %HTML.Recommended [
<!ENTITY % head.extra "META* & LINK*”>
]
]>
The HTML2 SGML Declaration

<!SGML "ISO 8879:1986"
--
SGML Declaration for HyperText Markup Language (HTML).
5 --
6
7 CHARSET
8 BASESET "ISO 646:1983//CHARSET
9 International Reference Version
10 (IRV)//ESC 2/5 4/0"
11 DESCSET 0 9 UNUSED
12 9 2 9
13 11 2 UNUSED
14 13 1 13
15 14 18 UNUSED
16 32 96 32
17 127 1 UNUSED
18 BASESET "ISO Registration Number 100//CHARSET
19 ECMA-94 Right Part of
20 Latin Alphabet Nr. 1/ESC 2/13 4/1"
21 DESCSET 128 32 UNUSED
22 160 96 32
23
24 CAPACITY SGMLREF
25 TOTALCAP 150000
26 GRPCAP 150000
27
28 SCOPE DOCUMENT
29 SYNTAX
30 SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
31 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 127
32 BASESET "ISO 646:1983//CHARSET
33 International Reference Version
34 (IRV)//ESC 2/5 4/0"
35 DESCSET 0 128 0
36 FUNCTION
37 RE 13
38 RS 10
39 SPACE 32
40 TAB SEPCCHAR 9
41
42 NAMING LCNMSTRT "".
43 UCNMSTRT "".
44 LCNMCHAR ".-"
45 UCNMCHAR ".-"
46 NAMECASE GENERAL YES
47 ENTITY NO
48 DELIM GENERAL SGMLREF
49 SHORTREF SGMLREF
50 NAMES SGMLREF
51 QUANTITY SGMLREF
52 ATTSPLEN 2100
53 LITLEN 1024
54 NAMELEN 72 -- somewhat arbitrary; taken from
55 internet line length conventions --
56 PILEN 1024
57 TAGLEN 2100
58 GRPCTCN 150
59 GRPCTN 64
60
61 FEATURES
62 MINIMIZE
63 DATATAG NO
64 OMITTAG YES
65 RANK NO
66 SHORTTAG YES
67 LINK
68 SIMPLE NO
69 IMPLICIT NO
70 EXPPLICIT NO
71 OTHER
72 CONCUR NO
73 SUBDOC NO
74 FORMAL YES
75 APPINFO "SDA" -- conforming SGML Document Access application
76 --
77 >
78 <!--
80 Author: Daniel W. Connolly <connolly@hal.com>
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.

```plaintext
-- catalog: SGML Open style entity catalog for HTML --

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

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

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

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 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
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 "-//IETF/DTD HTML 3.0//EN/" /afs/cern.ch/user/j/jsaarela/sgml/dtds/html3.dtd
PUBLIC "-//HAL and O'Reilly/DTD DocBook//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/docbook.2.2.1.dtd

PUBLIC "-//ISO 8879-1986//ENTITIES Added Latin 1//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/iso-lat1.gml
PUBLIC "-//ISO 8879-1986//ENTITIES Added Latin 1 for HTML//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/iso-lat1.gml
PUBLIC "-//ISO 8879-1986//ENTITIES icons for HTML//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/htmdicent.gml
PUBLIC "-//ISO 8879-1986//ENTITIES Math and Greek for HTML//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/iso-grk1.gml
PUBLIC "-//HAL and O'Reilly//DTD DocBook//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/docbook.2.2.1.dtd

PUBLIC "-//HAL and O'Reilly//DTD DocBook//EN" /afs/cern.ch/user/j/jsaarela/sgml/dtds/docbook.2.2.1.dtd
PUBLIC "-//General--" /afs/cern.ch/user/j/jsaarela/sgml/dtds/html3.decl
```

See also: [http://www.hal.com/~TEdonnally/html-spec](http://www.hal.com/~TEdonnally/html-spec)
See also: [http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html](http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html)
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.

```xml
<!DOCTYPEento ISOS1 PUBLIC "ISO 8879-1986//ENTITIES Added Latin1//EN">
ISOlat1;

<!-- (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 "ISOlat1";>
    "ISO 8879-1986//ENTITIES Added Latin1//EN">

-->

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.
F  The HTML3 DTD — Tables and Mathematics Parts

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

```xml
<!ELEMENT CAPTION ((text;)+ -- table or figure caption -->
<!ATTLIST CAPTION %attrs; align (top|bottom|left|right) #IMPLIED
</caption>
<!ELEMENT TABLE (%block.align)+ -- table -->
<!ATTLIST TABLE %attrs; align (bleedleft|left|center|right|bleedright|justify) center
<!ENTITY % block.align "align (bleedleft|left|center|right|bleedright|justify) center">
```

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.

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 R40". 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 AXES 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 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 --
novrap (novrap) #IMPLIED -- don't wrap words --
>

<!ENTITY % cell "TH | TD">
<!ENTITY % vertical.align "top|middle|bottom|baseline">
<!ELEMENT TR - O (%cell)* -- row container -->
<!ATTLIST TR
attrs;
align (left|center|right|justify) #IMPLIED
valign (%vertical.align) top -- vertical alignment --
novrap (novrap) #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 --
novrap (novrap) #IMPLIED -- don't wrap words --
axis CDATA #IMPLIED -- axis name, defaults to element content --
axes CDATA #IMPLIED -- comma separated list of axis names --
>

<!ENTITY % HTMLmath PUBLIC
"-//IETF//ENTITIES Math and Greek for HTML//EN">
<!-- ISO subset chosen for use with the widely available Adobe math font -->
<!-- Subscripts and Superscripts
<SUB> and <SUP> are used for subscripts and superscripts.
X<SUP>i</SUP><SUB>j</SUB> is X^i \_j
-->
i.e. the space following the \text{X} disambiguates the binding.

The \text{align} attribute can be used for horizontal alignment, e.g. to explicitly place an index above an element:

\[ X^{\text{align=center}i} \] 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 \( x^{sup}2 \)
\[ y_{z} \] is mapped to \( y_{sub}z \)
\[ \{a+b\} \] is mapped to \( <box>a + b</box> \)

Note that these only apply within the \text{MATH} element and can’t be used in normal text!

\[
\begin{aligned}
\text{<!ENTITY REF1 STARTTAG "SUP">}
\text{<!ENTITY REF2 ENDTAG "SUP">}
\text{<!ENTITY REF3 STARTTAG "SUB">}
\text{<!ENTITY REF4 ENDTAG "SUB">}
\text{<!ENTITY REF5 STARTTAG "BOX">}
\text{<!ENTITY REF6 ENDTAG "BOX">}
\end{aligned}
\]

\[
\begin{aligned}
\text{<!USEMAP MAP1 MATH>}
\text{<!USEMAP MAP2 SUP>}
\text{<!USEMAP MAP3 SUB>}
\text{<!USEMAP MAP4 BOX>}
\end{aligned}
\]

\[
\begin{aligned}
\text{<!SHORTREF MAP1 "^" REF1}
\text{"_" REF3}
\text{"{" REF5 >}
\end{aligned}
\]

\[
\begin{aligned}
\text{<!SHORTREF MAP2 "^" REF2}
\text{"_" REF3}
\text{"{" REF5 >}
\end{aligned}
\]

\[
\begin{aligned}
\text{<!SHORTREF MAP3 "_" REF4}
\text{"^" REF1}
\text{"{" REF5 >}
\end{aligned}
\]

\[
\begin{aligned}
\text{<!SHORTREF MAP4 "}" REF6
\text{"^" REF1}
\text{"_" REF3}
\text{"{" REF5 >}
\end{aligned}
\]

\[
\begin{aligned}
\text{<!ENTITY % mathvec "VEC|BAR|DOT|DDOT|HAT|TILDE" -- common accents -->}
\text{<!ENTITY % mathface "B|T|BT" -- control of font face -->}
\text{<!ENTITY % math "BOX|ABOVE|BELOW|%mathvec|ROOT|SQRT|ARRAY|SUB|SUP|%mathface">}
\text{<!ENTITY % formula="#PCDATA|%math">}
\end{aligned}
\]

\[
\begin{aligned}
\text{<!ELEMENT MATH - - (#PCDATA)* -(%notmath) +(%math)>}
\text{<!ATTLIST MATH id ID #IMPLIED}
\text{model CDATA #IMPLIED>}
\end{aligned}
\]

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

\[
\begin{aligned}
\text{\{1 + X<OVER>Y\} is _______}
\text{Y}
\end{aligned}
\]

\[
\begin{aligned}
\text{\{a + b<ATOP>c - d\} is}
\text{c - d}
\end{aligned}
\]

The delimiters are represented using the \text{LEFT} and \text{RIGHT} elements as in:
Use \{ and \} for "{}" respectively as these symbols are used as shorthand for BOX, e.g.

\{\{\langle LEFT\rangle x + y<RIGHT\rangle\}\} is \{ x + y \}

\{\langle LEFT\rangle a<RIGHT\rangle\} is \(a\)

\{||\langle LEFT\rangle a<RIGHT\rangle||\} is \(\|a\|\)

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

\{\int_{\langle SUB\rangle a<RIGHT\rangle}^{\langle SUP\rangle b<RIGHT\rangle}\langle LEFT\rangle \frac{f(x)}{1+x} \langle \over\langle OVER\rangle dx\rangle\}\} is \(\int_{a}^{b} \frac{f(x)}{1+x} \, dx\)

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>...</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>
<!ELEMENT ABOVE - - (%formula)+>
<!ATTLIST ABOVE symbol ENTITY #IMPLIED>
<!ELEMENT BELOW - - (%formula)+>
<!ATTLIST BELOW symbol ENTITY #IMPLIED>
<!ELEMENT ROOT - - ((%formula)+, OF, (%formula)+)>
<!ELEMENT OF - O (%formula)* -- what the root applies to>
<!ELEMENT SQRT - - (%formula)* -- square root convenience tag -->
<!ELEMENT (T|BT) - - (%formula)+>
<!ATTLIST (T|BT) class NAMES #IMPLIED>
<!ELEMENT (\%mathvec) - - (%formula)+>
<!ELEMENT (\langle LEFT\rangle | BT\langle RIGHT\rangle) - - (%formula)+>
<!ELEMENT (\langle LEFT\rangle | BT\langle RIGHT\rangle) - - (%formula)+>
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. -->

---
<!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 -->
<!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" -->
<!ENTITY % a.types "(latin|greek|cyrillic|hebrew|kanji) latin" -->
<!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 -->
<!ELEMENT fraction - - (num, den) -- fraction -->
<!ELEMENT num - - (%p.trans;|%m.math;)* -- numerator -->
<!ELEMENT den - - (%p.trans;|%m.math;)* -- denominator -->
<!ATTLIST fraction shape (built|case) #IMPLIED align (left|center|right) style (single|double|triple|dash|dot|bold|blank|none) -->
<!ELEMENT sup - - (%p.trans;|%m.math;)* -- superior -->
<!ELEMENT inf - - (%p.trans;|%m.math;)* -- inferior -->
<!ATTLIST sup location (pre|post) -->
TUGboat, Volume 16 (1995), No. 2

109 arrange (compact|stagger)
110 compact

111 <!ATTLIST inf location (pre|post) post
113 arrange (compact|stagger) compact

114 <!-- Embellishments -->
116 <!ELEMENT MIN CONTENT EXPLANATIONS -->
118 <!ELEMENT top - - (%p.trans;|%m.math;)*
122 -- top embellishment
124 <!ELEMENT bottom - - (%p.trans;|%m.math;)*
125 -- bottom embellishment

126 <!--- ELEMENT NAME VALUE DEFAULT -->
127 <!ATTLIST top align (left|center|right)
129 center
132 align (left|center|right)
133 center
135 align (left|center|right)
137 center

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

143 <!--- Fences, boxes, overlines and underlines -->
147 <!--- ELEMENT MIN CONTENT EXPLANATIONS -->
151 <!ELEMENT mark - O EMPTY >
153 -- fence
155 -- overline
157 <!ATTLIST mark id ID #REQUIRED >
159 post post CDATA "|
161 style (single|double|triple|dash|dot|bold|blank|none)
165 -- to pick up a height
166 <!ATTLIST post post CDATA "|
167 style (single|double|triple|dash|dot|bold|blank|none)
169 sizeid ID #IMPLIED
171 sizeref IDREF #IMPLIED
173 box style (single|double|triple|dash|dot|bold|blank|none)
175 -- to pick up a height
177 start IDREF #IMPLIED

181 <!ATTLIST undrline type CDATA "_" -- embellishment
184 single
188
<!ELEMENT subform - (p, m)* -- base element -->
<!ATTLIST subform sizeid ID #IMPLIED
sizeref IDREF #IMPLIED
-- to pass on a width, or
-- to pick up a width --

<!ELEMENT radical - (radix?, radicand) -- root or radical -->
<!ELEMENT radix - (p, m)* -- radix -->
<!ELEMENT radicand O O (p, m)* -- radicand -->

<!ELEMENT array - (arrayrow+, arraycol+) -- array -->
<!ELEMENT arrayrow - O (arraycel*) -- array row -->
<!ELEMENT arraycol - O (arraycel*) -- array column -->
<!ELEMENT arraycel O (p, m)* -- array cell -->

<!ATTLIST array rowalign NMTOKENS #IMPLIED
-- row alignment --
colalign NMTOKENS #IMPLIED -- column alignment --
rowsep NMTOKENS #IMPLIED -- row separators --
colsep NMTOKENS #IMPLIED -- column separators --

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

<!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 -->

<!ELEMENT formula - (p, m)* -- in-line formula -->
<!ELEMENT dformula - (p, m)* -- display formula -->
<!ELEMENT dformgrp - (formula|dformula)* -- display-formula group -->

<!ATTLIST formula id ID #IMPLIED
alphabet %a.types;
-- %SDAPREF; "<@SDATRANS>Inline formula"
-- %SDASUSP; "SUSPEND" --

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>

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

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

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

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

</para>
```

The <command>get</command> 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</userinput>
</screen>
```

</para>
H.2 ESIS representation of the source document

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

```xml
<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>-ftp.ora.com</filename>:/pub/davenport/docbook-331 Guest login ok, send your complete e-mail address as password.</para>
  <input>-ftp ftp.ora.com</input>
  <input>-N</input>
  <output>-The DocBook DTD and related ASCII files are in the file named docbook.N.shar</output>
</sect1>
```
H.3 HTML3 output

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

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

```bash
% ftp ftp.ora.com
Connected to amber.ora.com.
Name (ftp.ora.com:terry): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: type e-mail address
ftp> cd pub/davenport/docbook
```

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

```bash
ftp> get docbook.2.2.1.shar
```

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

```bash
% sh docbook.2.2.1.shar
```

The get command will put this ASCII shar file on your system.
You must later unpack it on your system:

```bash
% sh docbook.2.2.1.shar
```