C **An Introduction to SGML**
W. W. Davis (Internal Revenue Service, Washington, DC)

D **Issues in Digital Typography**
Richard Rubenstein (Digital Equipment Corp., Hudson, MA)

E **Introduction to PostScript – A Graphics Solution**
Yvonne Perry (Adobe Systems, Palo Alto, CA)

F **Document Databases and Technical Publishing**
Geoffrey James (Honeywell Information Systems, Los Angeles, CA) under the auspices and organized by the University of Massachusetts/Boston and University of California/Los Angeles Extension Programs (to be confirmed)

Further Information

To obtain further information, request a form from the Conference Organizer:

Paulene McKeever
Conference Management Services
P. O. Box 5
51 Sandycove Road
Dún Laoghaire
Co. Dublin, Ireland
(+353-1) 452081
Telex: 30547 SHCN EI (Ref. Boole)

Please note that there is an early rate that applies to all fees *received* by the Conference Organizer before 1 September 1987. Reservations for accommodations must be made directly with the hotel before 15 September 1987 to ensure availability; a block of rooms has been reserved at a special conference rate for participants. Details of all fees and other arrangements will be on the form.

6th German TEX Meeting

On October 8th and 9th, 1987, German TEX Users and other people interested in TEX will meet in Münster (Westphalia) at the University Computing Centre.

For the first day we plan — for people who just had their first contacts with TEX — introductory lectures about installing TEX and using LATEX and other macro packages.

The second day we want to give information about further developments related to TEX. Also, we are going to deal with problems specifically concerning the German language.

During both days demonstrations will be offered on several Output Device Drivers and TEX Implementations on workstations and PCs.

For further information, please contact:

W. Kaspar
Univ of Münster
Computing Center
Einsteinstraße 60
D-4400 Münster
Fed Rep Germany

Late-Breaking News

SGML and TEX

Lynne A. Price
Hewlett-Packard, Palo Alto, CA

The Standard Generalized Markup Language, SGML, is defined in International Standard 8879, published in October, 1986. This paper gives an overview of SGML, discussing its relationship with other text processing tools such as TEX, PostScript, and WYSIWYG systems. It gives examples of applications for SGML. It concludes with a description of the way SGML and TEX are used together in one particular environment.

What is SGML?

The Standard Generalized Markup Language (SGML) is a notation for representing documents and making their inherent structure explicit. Various forms of automatic processing can be performed on documents coded in SGML; they can be formatted, loaded into online databases, or analyzed for various linguistic properties. SGML is defined in International Standard 8879.

SGML evolved from macro-based word-processing and text-formatting tools. Like a TeX macro package, it encourages a writer to use *descriptive markup*, identifying structures within a document, rather than *procedural markup*, specifying processing. For example, "this is a section heading" is preferred to "center this line in boldface". As the word "generalized" implies, documents prepared with SGML can be processed in various ways. For example, the same markup tags used to prepare a book's index might also be used by an information retrieval application to locate text relevant to selected terms.

SGML views a document as a hierarchy of *structural elements*. For example, a manual may be composed of front matter, some chapters, optional appendices, and an index. Similarly, a chapter may be a series of sections, while a section is composed of text and optional figures, tables, lists, and so on.

No finite set of structural elements can account for the vast flexibility permitted in written texts. SGML therefore provides features for defining types of documents and then coding particular documents that belong to the defined types. Possible document types include reference manuals, journal articles, term papers, short stories, third-grade book reports, memos, letters, and employee performance evaluations. A document type is formally defined with a *document-type definition* that itemizes the structural elements permitted in documents of that type and defines the contexts in which each element can occur. Most document-type definitions are prepared by a small group of individuals for many more people to use with a large number of documents. Thus, most users of SGML are concerned with creating and maintaining documents rather than document-type definitions.

Document-type definitions frequently distinguish elements that are formatted in similar fashion. For example, newly introduced terms and titles of books may both be typeset in italics. However, logically they are different structures. Markup that distinguishes between them enables software that supports glossary and bibliography maintenance.

Context-Sensitive Interpretation of Markup

The document-type definition can control context-sensitive interpretation of parts of a document. For instance, an asterisk may be interpreted as a code for the multiplication symbol inside a mathematical formula but as a footnote indicator elsewhere. Context-sensitive knowledge of valid document structure also permits various abbreviations of SGML constructs, called *markup minimization*. If it is known, for example, that every chapter begins with a chapter title, the SGML processor can recognize the first words in a new chapter as the title whether or not the writer has explicitly coded them as such.

Most SGML markup consists of identifying the beginning or end of structural elements. The most common convention (which can be overridden) is to mark the beginning of an element with the element name enclosed in angle brackets and to mark the end of an element similarly, but with the element name preceded with a slash. These delimiters are illustrated in the (deliberately verbose) example shown in Figure 1.

This example assumes that the document-type definition specifies rules for creating glossaries. Glossaries in this context are assumed to have titles and to contain multiple entries. Each entry has a term followed by a definition. Definitions may contain cross-references to other terms in the glossary.

The document-type definition may also specify context-sensitive text-entry conventions. For example, glossaries may be defined so that the title and terms never extend past the end of a line and that entries are separated by blank lines. With these definitions, SGML treats the example in Figure 2 exactly like the more complete form in Figure 1.

Since SGML knows which document-type definition is being used, the start-tag `<glossary>` can be omitted. The start-tag `<title>` is optional because all glossaries must start with a title. The end-tag `</title>` is optional because the title cannot extend for more than one line. The blank line after the title is ignored because no text characters have been encountered to start the element expected after the title. At the word "aardvark", SGML recognizes that one or more start-tags have been omitted. The start-tag `<entry>` is implied since `<entry>` is the only element allowed after a title. The start-tag `<term>` is then implied since every entry begins with a term. The term cannot extend past the end of the line, so the material on the next line must be something else. Since every entry consists of a term followed by a definition, this must be a definition and SGML infers a `<definition>` start-tag. The blank line after the definition ends both the entry and the definition contained within it. The end-tag `</glossary>` is implied by the end of the input file.

SGML's knowledge of context can also be used in some forms of error checking. Most TEX users can sympathize with the user who inadvertantly omits the closing brace after an emphasized phrase and generates several pages printed in a boldface font or who neglects to close an indented list and discovers the rest of the document in narrow columns. SGML, referencing the appropriate document-type definition, knows that an emphasized phrase cannot span multiple paragraphs and that an indented list cannot cross a chapter boundary. When such markup occurs, the effect can be limited to a single paragraph or chapter and appropriate error messages issued. This context-checking is an inherent property of SGML rather than something that must be laboriously built into individual macros.

SGML and Other Tools

SGML is used with classes of related documents, rather than one-of-a-kind texts. The language is carefully and deliberately defined independently of any application. The International Standard specifies possible input of SGML source files without the corresponding output. Unlike TEX and various what-you-see-is-what-you-get systems, the purpose

```
<glossary>
<title>Glossary of Animals</title>
<entry>
  {\it Aardvark\/}
  <definition>The first animal listed in a
              dictionary.</definition>
<entry>
  {\it Cat>\/}
  <definition>A carnivorous mammal long domesticated and kept by
              man as a pet or for catching mice (Webster's New
              Collegiate Dictionary, 1973).
  <definition>
<entry>
  {\it Dog\/}
  <definition>A domesticated <xref>canine</xref>.</definition>
</entry>
          . . .
</glossary>
```

Figure 1. Example of glossary data with full markup

```
Glossary of Animals

Aardvark
The first animal listed in a dictionary.

Cat
A carnivorous mammal long domesticated and kept by
man as a pet or for catching mice (Webster's New
Collegiate Dictionary, 1973).

Dog
A domesticated <xref>canine</xref>.

. . .
```

Figure 2. Example of glossary data after markup minimization

of SGML is not to determine how to arrange characters on paper. Nor is SGML a page-description language like PostScript. The purpose of SGML is to describe the logical structure of a document in a way that can be used by different processes.

Of course, individual uses of SGML have a particular goal, which may be document formatting or page layout. Software to support these applications from SGML can be written. The advantages of doing so are the advantages of context-sensitive markup described above and the ability to use the same source file for other applications as well. These advantages can be preserved whether the application code is written specifically for use with SGML or SGML is used as a front-end to independent tools.

Applications of SGML

Although little SGML software is commercially available as yet, there are several ongoing development efforts. A project is under way at the Institute of Computer Sciences and Technology at the National Bureau of Standards (NBS) to develop an SGML validation suite. The validation suite is being built, along with a public-domain SGML parser, under the Computer-Aided Logistic Support (CALS) program of the Department of Defense. While the purpose of the test suite is to validate SGML parsers intended to process documents that conform to Department of Defense SGML requirements, its examples of correct syntax may also be useful to individuals learning the language.

The Association of American Publishers has defined several SGML document-type definitions for use by authors using machine-readable media to submit books and articles for publication. *The Chicago Guide to Preparing Electronic Manuscripts* (University of Chicago Press, 1987) describes similar markup for use by authors submitting material to the University of Chicago Press. Their guidelines also form a template for publishers defining their own requirements for submission of electronic material.

One particular publisher beginning to use SGML is the Internal Revenue Service. Potential SGML applications at the IRS include embedding the text of relevant sections of the tax code in explanatory material. The tax code can then be printed by an application that generates copy for legal review, but suppressed by the application that prints the information for the taxpayer. SGML can also be used to supply text to tax information services that can in turn distribute it to tax preparers and taxpayers with no government expense.

SGML as a Preprocessor for TeX

Over fifty independent writing departments located throughout the world produce user guides and reference manuals for Hewlett-Packard computers, software, and electronic instruments. Electronic interchange of material between departments is often desirable; for example, a manual written in one country may be translated to the local language in another. Interchange is complicated by the assortment of text processing tools used (which includes TeX) and the corresponding differences in markup as well as by the diverse hardware on which the tools are installed. The same problems hamper communication between the staff of different writing groups.

SGML supplies a means of standardizing markup conventions throughout the company, thereby allowing interchange of files without requiring replacement of all existing text processing software and the corresponding hardware. A shared markup technique also provides a vehicle for discussion among writers in different groups.

As an internal tool to aid in the production of user documentation, Hewlett-Packard has therefore developed an SGML parser and application generator called MARKUP. MARKUP allows SGML to be used as a front-end to other text-processing systems. A document-type definition that represents the structure of Hewlett-Packard user documentation has been developed and successfully compared to segments of different published manuals. The first MARKUP application, scheduled for beta testing mid-summer of 1987, uses TeX to print documentation from source files coded in SGML.

A MARKUP application is specified by a table which indicates the processing to be performed for each instance of every element included in the document-type definition. Table entries specify actions to be taken at the beginning of the element, within it, and at its end. When the MARKUP application generates a TeX source file analogous to the original SGML input, the actions usually consist of the TeX markup corresponding to the SGML construct. For example, the string "`{\it`" might be generated at the beginning of a book title, an introduced term, or a variable component in a computer command, while "`}`" is generated at the end of these structures. When a quotation mark occurs within normal text, the TeX open-quote convention "`‘‘`" is generated; when a quotation mark occurs within a quote element, the close-quote sequence "`’’`" is output.

When necessary, actions can also be entered as C code to be executed when the corresponding structure occurs. For example, C code is used to process forward and backward cross-references and to verify that every term introduced in the text is entered in the glossary.

Use of TEX with MARKUP differs from use of TEX by itself. For example, consider the TEX code used to start a chapter. Should a macro be defined for this purpose? Not necessarily. Macros are used to give a convenient label to sequences of instructions that are needed repeatedly. In this case, the code is isolated in the start-chapter cell of MARKUP's definition table. MARKUP invokes it as often as needed and, in effect, it has already been given a logical name (start-chapter).

However, debugging is simplified when macros are used; the TEX source file generated by MARKUP is more readable when it contains macro calls. The macros are not parameterized as they would be if the calls were user-written instead of automatically generated. For example, suppose that the chapter title is normally printed on the inside margin of the page header, but that the user can specify a different header if the chapter title is too long to fit in available space. User-invoked macros should be designed for the usual case. The chapter macro needs one parameter, the chapter title. To override the default page header, the user can call a second macro. When macros are automatically invoked, however, the chapter macro can have two parameters, the chapter title and the header specification, even though the values are usually identical. This repetition is not tedious to the user, since he enters the chapter title only once. Furthermore, there is no risk that two copies intended to be identical will in fact differ.

TUG and the Standards Community

In the United States, standards work on SGML began in ANSI Committee X3J6 and then moved to X3V1. TUG has maintained liaison with these committees since 1982, and TUGBoat regularly publishes liaison reports. Larry Beck is the current representative.

When TUG first sent me to an X3J6 meeting in January, 1982, my goal was to explain TEX concepts such as boxes and glue to committee members. I would like to belatedly thank TUG for my current involvement with SGML and am delighted to take this opportunity to convey information in the other direction.

TUG Business

Treasurer's Report

For the first time, TUG's financial statements have been audited. We invited the firm of Deloitte Haskins & Sells to examine our records. DH&S have stated their conclusions in the report which appears on the following pages. The report shows that cash receipts during 1986 exceeded cash disbursements by $98,000. The auditors' letter reflects their opinion that TUG's accounts might be more fairly stated were they reported on an accrual, rather than a cash basis, a change TUG plans for its 1987 reports (for example, although TUG did have an excellent year in 1986, a large portion of the ending cash balance represents 1987 membership dues, paid in advance during 1986).

It should be noted that one of TUG's fiscal goals is the building of an available reserve equal to one year's operating budget, a policy consistent with the practice of other non-profit societies, including the American Mathematical Society. Cash reserves at the end of 1986 totaled $143,000, as compared with operating expenses of $406,000.

Samuel B. Whidden, Treasurer