What is (tagged) text?

Frank Wm. Tompa

Centre for the New OED
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

ABSTRACT

In working on the *New OED* project, we, like many other researchers, have wrestled with large, intricate bodies of text. Based on this exposure, we have begun to investigate the similarities and differences between managing conventional business data and managing reference text data.

The paper begins with the claim that text can support complex models of the real world that cannot be captured more formally. Thus important information resources must be held as text, but the very absence of a formal model makes it difficult to identify the structures present in a text.

A common text structuring technique is descriptive markup, which introduces tags into a text stream. We present three views of tagged text: one based on tags *as* text, one on arbitrarily interleaved tags with text, and one on constrained tag placement in the text. Throughout the discussion, examples are drawn from our experience with the *OED*.

1. Text as a model

The role of a database is to model an enterprise, so that when queries are posed against the database, information can be obtained about the enterprise. Similarly a reference text is consulted to obtain information about aspects of our collective knowledge as modelled by its contents. A reference text database must capture the information of the reference materials, so that it can provide answers to queries for information about the same collective knowledge.

Unfortunately working with a reference text database is not as simple as working with a conventional database, because the content is not formally constrained: modelling with text does not distinguish which aspects of perceived reality are captured in the database and which are omitted [Kent78]. Whereas conventional database design begins with a business analysis to determine the users' requirements, followed by a synthesis of a model to capture all the relevant features in a highly structured form, text demands more editorial freedom.

Consider the following two definitions from the *OED2*:

> **conference** *sb.*
> **6.** A formal meeting for consultation or discussion; *e.g.* between the representatives of different sovereign states, the two Houses of Parliament or of Congress, the representatives of societies, parties, etc.
>
> **congress** *sb.*
> **6. a.** A formal meeting or assembly of delegates or representatives for the discussion or settlement of some question; *spec.* (in politics) of envoys, deputies, or plenipotentiaries representing sovereign states, or of sovereigns themselves, for the settlement of international affairs. Also an annual or periodical meeting or series of meetings of some association or society, or of persons engaged in special studies, as *Church Congress,* the name of annual meetings of the Church of England for discussion; *Social Science Congress, Congress of Orientalists,* etc.

To what extent can these definitions be modelled sufficiently well by other means that the text itself can be discarded? Can one hope to create a stand-alone "lexical knowledge base" that is derived from and equivalent to this data, yet distinct from it?

Working on the *New OED* project has convinced us that we must maintain the database in the form of text. Certainly a *legal* text database must preserve original wording if it is to serve as a definitive resource. Similarly, the decision to represent information by running text is equally valid for many other reference text databases.

Our approach is to represent a text as a single character string; in fact, the 540 Mbytes of data representing the *OED2* are stored in one file without line or record delimiters. The PAT text search engine then interprets such a character string as if it were a set of overlapping semi-infinite strings, each starting at a specified point in the text and encompassing the remainder of the text (to the end-of-file) [Gonnet88]. By providing operators that deal with arbitrarily long "prefixes" of these semi-infinite strings, all occurrences of truncated or complete words and truncated or complete phrases can be retrieved with equal efficiency, and all continuations of text after such a prefix can be readily examined [Fawcett89].

All users pose queries in terms of this text model, even though some queries against the database seek information about the text, and others seek information about the world modelled by that text. For example, consider a search for the string "proprietary name" in the *OED2*. An editor might pose such a query to find all entries that might need to be updated as names alter their proprietary status, whereas another user might pose such a query to retrieve information about earliest recorded usages. In the first instance, the editor is interested in the text itself; however, the second user accesses the dictionary to learn about the reality being modelled.

In summary, within the context of reference text databases, *text* models some domain of knowledge via sequences of words. We maintain the model by editing the text. Queries can be posed regarding the text itself or the reality it models. Of course, responses to queries about the knowledge domain can be no more accurate than is the model. For example, when the query "How many words did Shakespeare introduce into English?" is posed against the *OED* database as

```
docs EQ including "<A>Shaks</A>"
```

the answer depends not only on Shakespeare, but on the lexicographical skills of the dictionary's editors and on the consistency of the encoding.

2. Identifying the structure of a text

> "The logical structure of a document is defined exclusively by the sequence of its constituent elements and their relationships to each other. The elements comprise the intellectual content of the document. The author originally grouped the document into a sequence of elements which were intended to have some logical significance to the eventual reader. Examples of logical segments are chapters, headings, paragraphs, and figures." [AAP87]

For a reference text database, as for text databases more generally, we are faced with the problem that we must represent a complex text. Users of the text database must be able to access not only precise sequences of words, but also the structural units in which the editor placed those words and the relationships among those structural units (particularly their ordering and nesting). On the other hand, users of a text database typically wish to ignore arbitrary conventions, such as end-of-line-hyphens and choice of typography, that were imposed on the text to make it suitable for printing as a book.

If we start from an image of the printed page, the task of separating content from presentation is to distinguish an author's or editor's choice of words and their arrangement from the compositor's choice of format. In texts that evolve over time, we also wish to distinguish between components that appear in distinct versions (whether corrections, augmentations, or other variants) and to distinguish materials that serve as commentary on the original text.

Consider then the problem of identifying text structure, starting from the machine-readable form from which a document is typeset (i.e., starting from a typesetting or word processor file or from text data that is optically scanned or keyboarded). The typography and layout that the compositor chose to present the text must be analysed to discern the elements that underlie the logical structure.

When the document's author is no longer available for consultation, the task relies on postulating hypotheses for the author's intent. For structuring the *OED*, the editors of the second edition supplied descriptions of typical entries, from which a grammar was developed to define the apparent nature of the logical elements as well as their order and nesting [Kazman86]. In spite of the remarkable consistency in style, however, those involved with transduction had to decide repeatedly whether an anomalous construct captured a particular nuance intended by the authors, or whether it arose from arbitrary constraints on "page real estate" or perhaps as an error.

As a starting point, it was decided that the literary concept of a pure, original text need not be applied to such a reference text. The purpose of data capture was not to preserve an original for intensive analysis, but rather to form the foundation for a living text. Thus improvements could be made to the content itself, such as introducing IPA-based pronunciations and rewording some definitions to reflect political and social changes that have occurred during the last century. In fact, the text of the original *OED* is no longer separable from that of the *Supplement* and from the more recently added materials.

Having decided that the database need not be an exact image of the text that appeared in the original *OED*, the question of which aspects to preserve must be addressed. For example, is a sense number a part of the content or is it a printing convention for easy reference and cross-reference? Are the abbreviations used for parts of speech important aspects of content? A conservative approach was taken almost uniformly: the contents of the database were to reflect the

OED2 directly. Thus, it was decided that after sense numbers were updated to reflect the addition of new materials, they would be retained explicitly in the database, and, at least for the present, all abbreviations are maintained in their original form.

In modelling a reference text, a particularly difficult question to address is to what extent the ordering of text elements carries information. For example, if an entry carries several headwords, is the first in the list to be preferred, in which case the order is certainly part of the content, or is the ordering arbitrarily assigned, so that it is merely an artefact of presentation? Is it important that the name of the author of a cited quotation appear between the quotation date and the title of the work? To what extent is it merely convention that etymologies appear near the start of entries in the *OED*?

In most of the text, the ordering in the *OED* is critical to preserving content, in that it reflects historical development. This does not apply to the ordering of entries, which could be said to be arbitrary (based on alphabetical ordering by convention), but the use of relative cross-references such as "[f. prec.]" implies that re-ordering of entries requires alteration of some content. Because it was difficult or impossible to determine for which text components the ordering did, in fact, convey information, it was decided that *all* text was to be treated as sequential, preserving the order in which the text appears in the printed book.

Another difficult question arises from an examination of cross-references in the text. Whereas most cross-references are important signals to readers of the *OED* to examine related entries, others are merely space-saving devices that are therefore presentation conventions. Consider, for example, the following two cross-references:

lith, *sb.*^{2}
  [OE. ...(see LEAN *v.*^{1}, LADDER)...]
parameter. 2. a.
  1965 [see *keyboarded* ppl. adj. s.v. KEYBOARD *v.*]

In the first of these, more related information can be found in the etymologies for the referenced entries; for the second, the quotation referenced merely happens to use both words in a single sentence. Is the cross-reference itself, then, part of the content or presentation?

When a cross-reference directs a reader to additional information, it should be interpreted as a link to be resolved by the software retrieval system at a user's discretion [Raymond88]. However, when a cross-reference is merely a convenient shorthand for saving space, it could be replaced by the target text; this would make it more convenient for users and more efficient for the retrieval software. At least for the present, all cross-references in the *OED2* have been explicitly preserved.

The examination of cross-references raises a general question: to what extent can the text be altered to provide more convenient access? That is, should a database designer have some flexibility in how to represent text much as a compositor did when print was the storage medium?

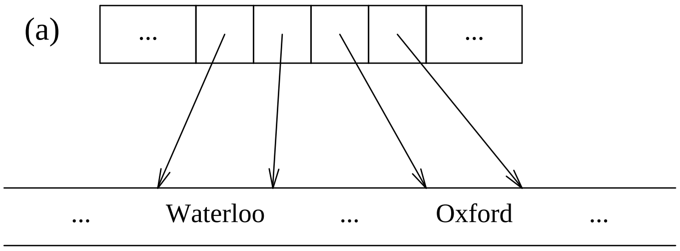
Consider, for example, research concerned with the development of English as evidenced by the introduction of new words into the language (see, for example, [Anshen86, Logan88]). This work relies extensively on identifying earliest cited quotations, for which simple algorithms can be applied at the time of a query. However, if the "earliest quotation" components are identified beforehand and marked explicitly in the text, the retrieval software operates more efficiently and display specifications can be written to present the earliest quotation differently from the rest (e.g., to suppress the printing of all quotations but the earliest one).

A database designer must account for expected uses of the data, and therefore text structuring is like other forms of classification: a structure's appropriateness is not a property of the data alone, but also of its intended use [Raymond89c]. Thus, deliberations in choosing how to structure the *OED* database have led us to conclude that text structuring is an ongoing activity, rather than something that can be done once and for all.

3. Descriptive markup

Having chosen the structuring to be applied to a text, the extent of each element in the text must be recorded. Consider, for example, a text structure that includes identifying occurrences of city names. The system can maintain a separate file of pointers to the start and end positions of all occurrences, as in Figure 1(a). Alternatively, the solution adopted by most text markup systems is to interleave "tags" with the text of the original document, still leaving the original words in proper sequence (Figure 1(b)). In its simplest interpretation, by suppressing the tags we are left with the original text, and by converting the tags to formatting instructions we can produce a suitable presentation of the text.

(a)



(b) `... <City>Waterloo</City> ... <City>Oxford</City> ...`

Figure 1: Element demarcation via external pointers and via tags

It would seem that an external representation might be advantageous since its use separates the "pure text" from the description of its structure, and thus simplifies modifying the descriptive component. In fact, this approach was attempted for representing the *OED*: the structure of the dictionary was recorded in a collection of external files, one per element type — taken as a whole, the files captured the hierarchical structure needed for managing the document. We quickly realized, however, that the pure text alone is inadequate for almost any processing needs, which often rely on interpreting the text as conveyed through its structure. To be practical, the identification of the structure associated with a fragment of text must be efficiently computable. For human readers, this implies that recognizing the extent and type of structure that includes a segment of text should require scanning that segment only; for most computer processing, it implies at least the ability to recognize the extent and type with a bounded amount of additional search. Thus relying on the external files only to determine the structure of an arbitrary segment of text was found to be overly cumbersome, and we have chosen to interleave tags with the text.

SGML defines one approach to descriptive markup, in which the ordering among tags is controlled by a "document type definition" [ISO86]. The chief role of SGML's document type definition is to serve as a *prescriptive* grammar to control the creation of new materials, much as a database schema is used to control the insertion of new data into a conventional database.

Such a grammar can also be used *descriptively* to inform new users about the data organization and to direct search and display tools to particular sections of the text.

For a text such as the *OED*, however, neither function is well-served by a grammar: there are too many variants for each rule. A prescriptive grammar that is small enough to be comprehensible by a human reader will not be sufficiently constraining to disallow incorrect text segments to be entered. Furthermore, the computer system *should* not constrain the dictionary's editors, if they decide that a certain segment of an entry is best expressed in an unconventional manner.

One use for the grammar is to define the structure of a single particular instance, such as the *OED2* at the time of its publication, by generating a syntax tree. Under the assumption that start and end tags appear in the text as pairs that conform to a proper nesting and without the aid of a grammar, a process can extract an equivalent syntax tree, in which each internal node with label L subtends a segment of text delimited by tags such as <L> and </L>. For example, Figure 1 shows a syntax tree for the text

```
<Q><D>1634</D> <A>Milton</A> <W>Comus</W> 782
       <T>The Sun-clad power of Chastity.</T></Q>
```

Thus, for many applications, the weaker constraint that tags be properly nested (e.g., disallowing <A>...<B>...</A>...</B>) can replace the constraint that the text conform to a grammar.

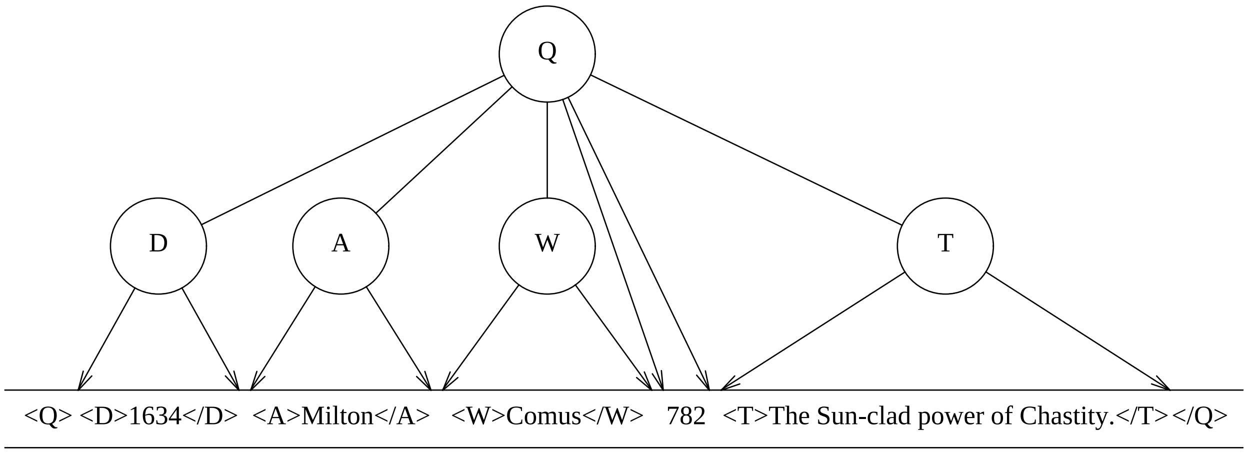


Figure 2: A tag-induced syntax tree

Although "full markup of an element consists of a start-tag to identify the beginning of an element and an end-tag at the end" [AAP87], the SGML standard permits tags to be omitted if their presence is implied by the document type definition and the positioning of other tags. Following arguments similar to those supporting our choice to adopt tags rather than merely external pointers to elements, we have chosen to avoid such a feature. At every location in the text it must be not only possible to determine all applicable tags for that location, but such determination must be efficient. Although tag omission does not necessarily violate the principle of bounded search, checking for the presence of any of several possible strings that would indicate the limits of an element is a complexity that would need to be added to each application that reads the text. Thus, the potential gains resulting from the use of tag omission are offset by the inconvenience or inefficiency necessitated to determine the location of the implied tags.

Another feature of SGML is that attributes can be associated with elements by specifying them as part of an opening tag. For example, according to the proposed Dictionary Encoding Standard [Amsler88], a specific dictionary sense could be tagged as

```
<S sn="1 a" time=obs><deftext>A half-pint of ale.</deftext></S>
```

indicating that the first subsense of sense 1 is obsolete and means “A half-pint of ale.”

In choosing to define attributes, a document designer must distinguish which components of a document represent “a characteristic quality, other than type or content” [ISO86]. As an alternative criterion, the use of attributes provides a method of specifying unordered data, so this feature could be chosen to indicate that order is deemed to be arbitrary. Using either criterion, the discussion in Section 2 indicates that the decision of whether or not to encode some structure as an attribute-value pair as opposed to a tagged element is difficult to make in general.

As a further deterrent against adopting attributes, SGML places severe restrictions on the form, worst of which is that no attribute’s value may be hierarchically structured. This could be overcome by augmenting SGML so that a list of attribute-value pairs is represented by a tagged text stream embedded in the opening tag. Thus, we could represent the above example as

```
<S <sn>1 a</sn><time>obs</time>><deftext>A half-pint of ale.</deftext></S>
```

The occurrence of tagged elements *inside the opening tag* distinguishes these as attributes rather than nested elements ithin the tagged element corresponding to `<S>...</S>`. Because each attribute is now itself represented as a tagged element, it can have its own attributes and be assigned any value, including one with subelements.

Consider again the Milton quotation from Figure 2. “1634,” “Milton,” “Comus,” and “782” were all interpreted to be part of the dictionary’s pure text, stored as separate elements hierarchically related to the quotation text. Because the quotation’s presence in the entry for **sun** is to indicate the use of the form “sun-clad”, it may be argued that the *content* of the element is the quotation text itself. The date, author, and work in a cited quotation could therefore have been indicated by attributes rather than as nested tags:*

```
<Q D="1634" A="Milton" W="Comus" loc="782">The Sun-clad power of Chastity.</Q>
```

Note that a syntax tree, such as that shown in Figure 3, can be built to incorporate the presence of attributes: non-terminal nodes indicated by rectangles in the figure are interpreted as denoting attributes.

In light of our difficulty in distinguishing those characteristics in the *OED* that should be represented as attributes from those that should be content, we have omitted the concept of attributes from our model of tagged text. In order that our software be compatible with data prepared for SGML, documents that are defined with attributes must be converted to alternative encodings that can be used in their place. Two styles have been suggested:

---

* Perhaps the alternative with attributes was not chosen because of a confusion between the text and the reality being modelled: attributes are viewed as being less important (and more transient) than content, and we are uncomfortable implying that Milton is somehow subservient to a text fragment he wrote.

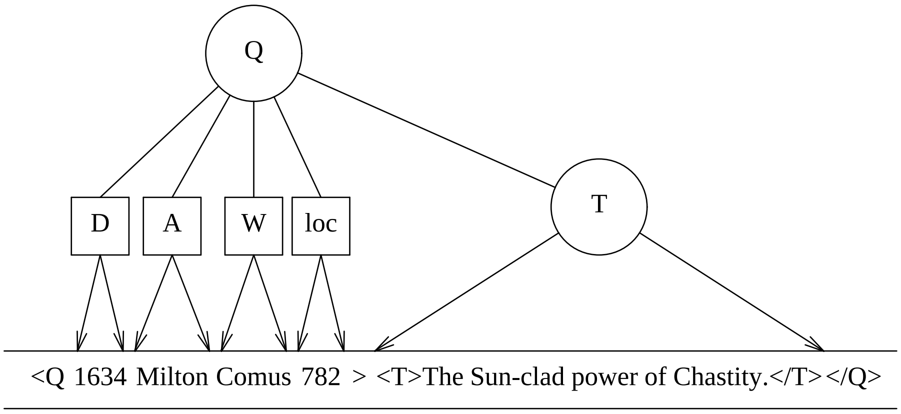


Figure 3: Syntax tree incorporating attributes

nested element:

In place of each attribute-value pair, include a nested element having the attribute name as its tag identifier and the value as its content. For the above example of a sense, this yields

```
<S><sn>1 a</sn><time>obs</time><deftext>A half-pint of ale.</deftext></S>
```

enclosing element:

In place of an attribute-value pair, surround the element by tags that reflect the presence of that pair. This option seems particularly applicable when the appearance of a value in a given tag uniquely determines the attribute involved (as is required in SGML when "NMTOKEN" is specified) or when the value is boolean and specified only to override the default (e.g., `<xref illus=t>` to identify cross-references to illustrations). Applying this to the "time" attribute in the above example yields

```
<obs><S sn="1 a"><deftext>A half-pint of ale.</deftext></S></obs>
```

In light of the similarity of the syntax trees in Figures 2 and 3 and the generality of the first option, we have implemented transducers to replace attribute-value pairs by nested tags. It is our intention to provide facilities to re-convert such nestings into attributes when a text is exported, choosing which tags to interpret as attributes at the discretion of the user.

By converting attributes to nested tags, we are apparently promoting some markup into a form often reserved for content, and thereby violating a fundamental premise of text representation. However, there is sufficient tagging preserved to distinguish the pure text written by the original author from the text that represents the values of attributes. This is therefore no more an intrusion on the author's words than is an editorial note.

4. Models of tagged text

When markup is represented by strings of characters embedded in the text, it can be subjected to the same operations as the document's content. From this arises PAT's model of tagged text in which tags are themselves just embedded strings of text. For example, PAT can be used to find all occurrences of the label music by a single search for the string "<LB>music</LB>" rather than intersecting all occurrences of the string music with all occurrences of the element type LB. Similarly, PAT can find the most frequently used label by using the command signif "<LB>" in the same way that the most frequently occurring word following "the " can be found by means of the command signif "the ".

On the other hand, to format a document or to extract complete elements from it, we need a model in which tags are distinguished from content. In practice, a tag is distinguished from the document's text by beginning with a character (or short sequence of characters) that is disallowed in the pure text (typically < for SGML applications and the character pair consisting of newline and . for -ms applications [Lesk78]).

LECTOR, a program written to filter a text and perform screen-based formatting, is built on a model of text as a linear sequence of interleaved, distinguishable tags and content [Raymond89a]. A user creates a "tag specification file" containing a set of format specifications, each of which defines for each tag the actions to be taken by the formatter when it encounters that tag. For example, Figure 4 contains most of the specification for formatting entries from the *OED*'s bibliography when only the author fields (tagged with <A>...</A>) are to be printed. Using LECTOR, tags can cause text to be output and can set the state of the transducer so that the content that follows is suppressed or output in a particular font and conforming to a particular layout; the content itself cannot be further interpreted or altered in any way. For example,

```
<ET><XR><XL>bi-</XL><HO>2</HO> <SN>1</SN> and <SN>2</SN></XR></ET>
```

could cause the string [**BI-**^{2} **1** and **2**] to appear on its own line of output, and <ed>check this part carefully</ed> could be completely suppressed.

Because LECTOR is designed to run interactively on arbitrarily large bodies of text, it must identify all relevant tags efficiently and perform the indicated formatting immediately. Thus LECTOR assumes that all tags begin and end with delimiting characters that can be recognized with no lookahead in the text (as, for example, SGML's "start-tag open," "tag close," "entity reference open," and "entity reference close" delimiters). LECTOR determines these symbols by examining the names of tags listed in the specification file. For example, the appearance of tags named <A> and &ae. in the specification file in Figure 4 indicate that all strings delimited by < and > or by & and . are treated as tags. As LECTOR reads characters from the text file, it compares them to possible opening characters for tags until a match is found, after which it searches for the matching closing delimiter; the character string thus isolated is treated as a tag.

More sophisticated software can be constructed if the ordering among tags is controlled by a grammar such as SGML's document type definition; this, then, is a third model of tagged text. As an example, the GOEDEL text management system provides various selection and transformation operators defined in terms of *p-strings*, which incorporate the concept of a text conforming to a grammar [Gonnet87]. In fact a syntax tree such as that shown in Figure 2 provides a sufficient framework to define GOEDEL's operators, and therefore an explicit grammar is not required. Furthermore, we propose that the text management system use the syntax tree, rather than a prescriptive grammar, to determine whether a newly created or modified segment of text

```
<Spec>
        <Name>Authors Only</Name>

        <Family>Times</Family>
        <Type>Roman</Type>
        <Size>10</Size>
        <PrText>off</PrText>             # By default, no printing
        <PrTag>off</PrTag>
        <PrUndef>off</PrUndef>
        <Tabs>3 6 9 12 15 18 21 24 27 30 33 36 39 42 45</Tabs>

        # First author in an entry
        #
        <Tag><Name><E><A></Name>
                <Indent>0</Indent>
                <PrText>on</PrText>
                <Type>Bold</Type>
                </Tag>

        # Second and subsequent authors
        #
        <Tag><Name><A></Name>
                <Indent>1</Indent>
                <PrText>on</PrText>
                <Type>Bold</Type></Tag>

        # Auxiliary author information
        #
        <Tag><Name><AA></Name>
                <Type>Roman</Type></Tag>

        # End of author's name
        #
        <Tag><Name></A></Name>
                <LineBreak>on</LineBreak>
                <PrText>off</PrText>
                </Tag>
                . . .

        # Handle entity references by substituting characters
        #
        <Tag><Name>&ae.</Name><String>ae</String></Tag>
        <Tag><Name>&Ae.</Name><String>Ae</String></Tag>
        <Tag><Name>&AE.</Name><String>AE</String></Tag>
                . . .

</Spec>
```

Figure 4: Format specification segment

appears only rarely (or never) in the remainder of the text, and to caution the editor whenever a structure thus appears to be incorrect.

In contrasting these last two models for tagged text (i.e., arbitrarily interleaving tags and content *vs.* constrained tag placement), it is important to recognize one major benefit of the simpler model. LECTOR is intended to be used to browse documents that are under construction as well as finished documents. By design, LECTOR is not dependent on a particular arrangement of tags with respect to the text; each tag occurrence is handled completely independently of every other one. Thus even documents not accompanied by a grammar, or containing tags that violate an accompanying grammar, can be processed according to the user's specifications. In fact by formatting that document with LECTOR, a proof-reader can highlight erroneous data.

The fourth major software component in the current version of our system is TRUC, the editor for tagged text [Raymond89b]. Because TRUC is used to create new segments as well as to retrieve and display sections of text that already exist, its model of tagged text shifts through all those already described. In searching for particular character strings within a text, tags are treated as character strings upon which any of the search tools can be used; TRUC also allows tags to be edited as any other character strings. In browsing a particular segment, the tags are assumed to conform to a syntax tree: the cursor can be quickly moved from start to end tag and back, as well as to the start tag of the enclosing text element or sequentially through the text elements nested within a given structure. In formatting text for display, TRUC uses the unconstrained tags-in-text model, sharing tag specification files with LECTOR.

Acknowledgements

The ideas in this paper have arisen from many discussions with members of the Data Structuring Group and of the Centre for the New OED; I am particularly indebted to Darrell Raymond, Gaston Gonnet, Tim Bray, and Derick Wood, all of whom have repeatedly forced me to reconsider and reformulate the ideas. The contents of this paper, including statements of fact or opinion, are solely my responsibility and do not reflect official positions of the Oxford University Press nor of the University of Waterloo. Financial support from the University of Waterloo and from the Natural Sciences and Engineering Research Council of Canada (through grant 0039063 sponsored under the University-Industry Program) is gratefully acknowledged.

References

[AAP87] Association of American Publishers, *Standard for Electronic Manuscript Preparation and Markup,* Version 2.0 edition, AAP, Washington, D.C., 1987.

[Amsler88] Amsler, Robert A. and Frank Wm. Tompa, “An SGML-based standard for English monolingual dictionaries,” *Proc. 4th Ann. New OED Conf. — Information in Text,* Univ. of Waterloo (Oct. 26-28, 1988), pp.61-79.

[Anshen86] Anshen, Frank, Mark Aronoff, Roy J. Byrd, and Judith L. Klavens, “The role of etymology and word length in English word formation,” *Proc. 2nd Ann. New OED Conf. — Advances in Lexicology,* Univ. of Waterloo (Nov. 9-11, 1986), pp. 17-28.

[Fawcett89] Fawcett, Heather, *Pat 3.3 User's Guide,* Centre New OED, Univ. Of Waterloo, 1989, 102 pp.

[Gonnet87] Gonnet, Gaston H. and Frank Wm. Tompa, "Mind your grammar: a new approach to modelling text," *Proc. 13th Int. Conf. on Very Large Data Bases* (VLDB87), Brighton, England (Sept. 1-4, 1987), pp. 339-346.

[Gonnet88] Gonnet, Gaston H., "Efficient searching of text and pictures," Centre New OED Tech. Rept. OED-88-02, Univ. of Waterloo (June 1988), 11 pp.

[ISO86] International Organization for Standardization, *International Standard 8879: Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML),* first edition — 1986-10-15 (Ref. No. ISO 8879-1986 (E)), 155 pp..

[Kazman86] Kazman, Rick, *Structuring the text of the Oxford English Dictionary through finite state transduction,* Centre New OED Tech. Rept. OED-86-20, Univ. of Waterloo (June 1986), 117 pp.

[Kent78] Kent, William, *Data and Reality: Basic Assumptions in Data Processing Reconsidered,* North-Holland, 1978.

[Lesk78] Lesk, Michael E., "Typing documents on the UNIX system: using the -ms macros with troff and nroff," in *UNIX User's Manual, Supplementary Documents*, 4.2 Berkeley Software Distribution, Virtual VAX-11 Version, March 1984.

[Logan88] Logan, Harry M. and Grace Logan, "An inquiry into inquiry systems: a discussion of some applications of data retrieval to the New OED database," *Proc. 4th Ann. New OED Conf. — Information in Text,* Univ. of Waterloo (Oct. 26-28, 1988), pp.81-95.

[Raymond88] Raymond, Darrell R. and Frank Wm. Tompa, "Hypertext and the *Oxford English Dictionary*," *Comm. ACM 31*,7 (July 1988), 871-879.

[Raymond89a] Raymond, Darrell R., "Lector: a formatter for tagged text," Centre New OED tech. rept., Univ. of Waterloo, in preparation.

[Raymond89b] Raymond, Darrell R., ed., "A potpourri of prototypes," Centre New OED tech. rept., Univ. of Waterloo, in preparation.

[Raymond89c] Raymond, Darrell R., "Reading between the tags: an appraisal of descriptive markup," Centre New OED tech. rept., Univ. of Waterloo, in preparation.