

Cahiers **GUT** *enberg*

☞ TYPESETTING SGML DOCUMENTS USING
T_EX

☞ Andrew E. DOBROWOLSKI

Cahiers GUTenberg, n° 10-11 (1991), p. 185-196.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_1991__10-11_185_0>

© Association GUTenberg, 1991, tous droits réservés.

L'accès aux articles des *Cahiers GUTenberg*

(<http://cahiers.gutenberg.eu.org/>),

implique l'accord avec les conditions générales

d'utilisation (<http://cahiers.gutenberg.eu.org/legal.html>).

Toute utilisation commerciale ou impression systématique

est constitutive d'une infraction pénale. Toute copie ou impression

de ce fichier doit contenir la présente mention de copyright.

Typesetting SGML documents using T_EX*

Andrew E. DOBROWOLSKI

ArborText Inc., 535 West William Street, Suite 300, Ann Arbor, MI 48103
aed@arbortext.com

Abstract. Since its publication as an international standard in 1986, the Standard Generalized Markup Language (SGML) has become a preferred document markup standard within many industries. Many users have developed their own document type definitions (DTDs) which define the elements (tag sets) for their documents. However, if SGML is to become a universally accepted standard of document interchange then a standard way to specify formatted output and a means of producing that output will be needed.

The U.S. government's Computer-aided Acquisition and Logistic Support (CALs) initiative selected SGML as the standard of text interchange. The output specification section of the CALs standards proposed the Formatted Output Specification Instance (FOSI) as the means of formatted output specification interchange.

T_EX can be used as the formatting engine to implement FOSI-based formatting. But without extending T_EX not every FOSI formatting request can be fulfilled. Conversely, certain T_EX capabilities cannot be formulated in terms of FOSI characteristics. However a FOSI/T_EX based formatting system would be a major advance towards fulfilling the document interchange needs of a growing community of SGML users.

Résumé. Depuis sa publication en tant que norme internationale en 1986, SGML a été adopté comme système de marquage de documents dans beaucoup de milieux industriels. Nombre d'utilisateurs développent leurs propres DTD (définition de type de document) qui définissent les différents éléments de leurs documents. Cependant, si SGML est appelé à devenir une norme universelle pour l'échange de documents, il est nécessaire de disposer d'un standard pour spécifier les sorties finales et les façons de les obtenir.

Le programme américain CALs a choisi SGML comme norme d'échange de documents et a proposé la norme FOSI pour spécifier l'aspect physique du document.

T_EX peut être utilisé pour traiter les formats FOSI. Mais toutes les requêtes FOSI ne peuvent être exécutées sans extension de T_EX. De même,

*The first publication of this paper appears in *TUGboat* vol. 12(3), Proceedings of the the TUG 1991 annual meeting. It is published here with authorization of *TUGboat*'s editor.

certaines possibilités de T_EX ne peuvent s'exprimer en termes FOSI. Quoi qu'il en soit, un système basé sur T_EX et FOSI sera un grand pas en avant pour répondre à la demande de la communauté grandissante des utilisateurs de SGML.

Key words: SGML, FOSI, CALS, standards

1. Document Interchange Standards.

In the past ten years T_EX has become a well known and widespread language for typesetting technical documents. From its original base of universities and colleges it has spread so that people in industries with only incidental needs for publishing have heard about it. A large part of T_EX's appeal comes from its portability, since the program is in the public domain and has been ported to quite a number of operating systems. There is no standard for the way a T_EX document is "marked up", it depends on the macro package being used. Given the right macro package and fonts, the formatted output of two different T_EX implementations on two different machines will produce identical results.

By contrast generic markup systems identify document structures without making assumptions about the end application of the document. This makes the same document useful to various programs and for various applications. Generic markup has been around in several flavours for over ten years. These dissimilar flavours were a hindrance to its utility. To remove this hindrance and to promote the portability and acceptance of generic markup, an international standard (IS) specification for generic markup was established in 1986. Since then SGML (Standard Generalized Markup Language) has become extremely important to industry, especially in areas where huge quantities of data have become a document management nightmare. Today a large number of programs can read and write SGML on a variety of platforms.

The U.S. government's Computer-aided Acquisition and Logistic Support (CALS) initiative gave SGML additional clout by selecting SGML as the standard of text interchange between the Department of Defense and its sub-contractors. However SGML contains no information pertaining to the printed representation of a document or the meaning attached to the markup. The companion standard to SGML which addresses standardized formatting specifications, the Document Style Semantics and Specification Language (DSSSL), is still in the design stages. It is not expected to

become an international standard until at least 1993. For this reason the output specification section of the CALS standards proposed the Formatted Output Specification Instance (FOSI) as the means of output specification interchange.

2. SGML and FOSI structure: An Overview.

All SGML documents must conform to certain rules which are defined partially by the standard and partially by a prolog to the document which is called the document type definition (DTD). The DTD defines the “elements” of a document, which in a document instance are marked off by start tags and end tags. For example, a hypothetical section may be marked up like the fragment on the left in Figure 1. There `<head>` and `</head>` (pronounced “head” and “end head”) are start and end tags that delimit the head element. The parent of head is section and its siblings are the two para elements.

<pre> <section> <head>SGML and FOSIs: An Overview.</head> <para>All SGML documents conform to certain rules which are defined partially by the standard and partially by a prolog to the document which is called the document type definition (DTD).</para> <para> In addition to being first off the starting blocks to becoming a national standard, the FOSI is also the most manageable.</para> </section> </pre>	<pre> \section{} \sectionhead{}SGML and FOSIs: An Overview.\endsectionhead{} \firstpara{}All SGML documents conform to certain rules which are defined partially by the standard and partially by a prolog to the document which is called the document type definition (DTD).\endfirstpara{} \nonfirstpara{} In addition to being first off the starting blocks to becoming a national standard, the FOSI is also the most manageable.\endnonfirstpara{} \endsection{} </pre>
--	--

Figure 1. A Document Instance Fragment and its Translation into T_EX.

A DTD also defines what “attributes” are associated with an element. An attribute is an annotation that appears in the document instance and augments the information provided by the markup. Attributes appear within an elements start tag. If the element called “head” had an attribute “id” for

use in cross references, then that attribute could be assigned some value in the document instance, for example `<head id="overview">`.

It is important to note that SGML allows the same element to appear in many contexts within a document structure. The same markup can be used to describe a chapter head, a section head and even a table head. At some point a distinction must be made between these various contexts, at least for the purpose of formatting the document. But since the DTD also restricts the possible contexts in which any element may appear, using content models, the task of defining the style of every element in every possible context can be fairly well defined. Thus a FOSI will not define the formatted output style of a document element but of an element in context (or e-i-c).

Many industries have developed DTDs which define the elements (tag sets) used to mark up their documents. Before SGML becomes a universally accepted standard of document interchange, one of SGML's companion standards for output specification must be fully implemented. T_EX could be the engine in the implementation—the means of producing standardized output for any SGML document. The ultimate goal would be to make this process automatic for the arbitrary DTD document. The only information that would need to pass from one site to another in order to print a document would be the document instance, the DTD, and an output specification.

It appears that FOSI's are the closest of all proposed output specification standards to becoming a recognised standard. In addition the FOSI specification is the easiest to implement. A FOSI is itself an SGML document that conforms to the Output Specification (OS or outspec) DTD. But instead of being made up of parts, chapters, or sections, a FOSI is made up of divisions that describe page models and the output format of each of the document's elements.

There are six major divisions in an output specification instance: the security description (secdesc), the page description (pagedesc), the element style description (styldesc), the table element style description (tabdesc), the graphical element description (grphdesc), and the footnote area description (ftndesc). All but the pagedesc and styldesc are optional. There is still no definition for the output style of mathematical formula elements. Thus either mathematics must be passed through in the native language of the formatting system, translated into the native language by the translator, or the formatting system must have the output specification for the mathematical elements "hard wired".

The style description is the most important division of the outspec for simple text documents. The styl Desc contains a document description (docdesc), zero or more environment descriptions (envdesc) and at least one formatting specification for an e-i-c. It is in these subdivisions that special FOSI elements called categories appear. Each category provides data on a different aspect of the formatted output. There are 24 categories (with names such as font, leading, etc.) and each of these has from one to 13 attributes. These, when fully specified, exactly define the formatting aspect with which their category is concerned. These attributes are called characteristics of which there are 128 in total. Once values for all the characteristics of any given e-i-c have been determined, it should be possible to define that e-i-c's appearance on the printed page.

The categories control the font, leading, hyphenation, word spacing, letter spacing, indents, horizontal justification, highlight, change marks, prespace, postspace, page breaking, vertical justification, text breaking, spanning, page borders, ruling, character fill, enumeration, print suppression, automatic generation of text, automatic generation of graphics, the saving of text for cross reference, and the use of text saved for cross reference.

SGML and FOSI structure: An Overview.

All SGML documents must conform to certain rules which are defined partially by the standard and partially by a prolog to the document which is called the document type definition (DTD).

In addition to being first off the starting blocks to becoming a recognised standard, the FOSI is also the most manageable.

Figure 2. Typeset Document Fragment.

As mentioned above, the elements that may appear in a styl Desc are docdesc, envdesc and e-i-c. The characteristics of the docdesc define the style of the overall document and the default values for characteristics that are needed but not specified in an e-i-c. When used in this way, the docdesc is called the default environment. The envdesc section defines "named" environments that may be used instead of the default environment. The actual style definition for an element in a particular context in the document instance is given by an e-i-c. The SGML terminology for an element's name is generic identifier (gi). An e-i-c specifies an element, its context and

its occurrence within that context by using the `gi`, `context` and `occur` attributes, as is seen in Figure 1.

```

<e-i-c gi="section">                                \def\section{\starteic{section}
  <charlist>                                         \font{\def\family{cm}}
    <font family="cm">                                \presp{\nominal=30pt
    <presp nominal="30pt"                             \minimum=30pt \maximum=30pt}
      minimum="30pt" maximum="30pt">                \eiccontent}
  </charlist>
</e-i-c>                                             \def\endsection{\endeic{section}}

<e-i-c gi="head" context="section">                 \def\sectionhead{\starteic{head}
  <charlist>                                         \font{\inherit=1 \def\style{sans}
    <font inherit=1 style="sans"                       \size=14pt \def\weight{bold}}
      size="14pt" weight="bold">                     \leading{\lead=14pt}
    <leading lead="14pt">                             \quadding{\def\quad{right}}
    <quadding quad="right">                           \keeps{\keep=1 \next=1}
    <keeps keep="1" next="1">                         \postsp{\nominal=24pt
    <postsp nominal="24pt"                             \minimum=20pt \maximum=30pt}
      minimum="20pt" maximum="30pt">                \eiccontent}
  </charlist>
</e-i-c>                                             \def\endsectionhead{\endeic{head}}

<e-i-c gi="para" occur="first">                     \def\firstpara{\starteic{para}
  <charlist>                                         \indent{\firstln=0pt}
    <indent firstln="0pt">                             \eiccontent}
  </charlist>
</e-i-c>                                             \def\endfirstpara{\endeic{para}}

<e-i-c gi="para" occur="nonfirst">                 \def\nonfirstpara{\starteic{para}
  <charlist>                                         \indent{\firstln=15pt}
    <indent firstln="15pt">                           \presp{\nominal=6pt
    <presp nominal="6pt"                             \minimum=4pt \maximum=6pt}
      minimum="4pt" maximum="6pt">                \eiccontent}
  </charlist>
</e-i-c>                                             \def\endnonfirstpara{\endeic{para}}

```

Figure 3. A FOSI fragment and its Translation into T_EX.

Furthermore this FOSI also uses the `occur` attribute of an `e-i-c` to make a distinction between the output format of the first and non-first occurrences of the `para` element. The paragraph indent of the first `para` within a structure is zero, while non-first paragraphs have an indent of 15 points and an additional pre-space of 6 points. Figure 2 shows the formatted output from

the document instance fragment. Characteristics not explicitly listed in the e-i-c definitions default to the values specified in the docdesc (not shown).

3. SGML to T_EX Translation.

As with most SGML documents, the FOSI must first be read by an SGML parser or a dedicated program and translated into a form suitable for the formatting engine. Likewise, the document instance must be translated by some process into a suitable form.

The translation of a FOSI into T_EX looks like a series of T_EX macro definitions. These define the macros that appear in the translation of the document instance into T_EX. Given a suitable starting set of macros it is possible to load the new macro definitions produced automatically from the FOSI translation and to format the document.

Because the output specification for a given document element is context sensitive, either the translation process or T_EX must track and differentiate between differing contexts. To make the work of the macro package easier, the context sensitivity should be built into the translation process. In fact, T_EX's limited lookahead prescribes that the translation be context sensitive. T_EX cannot recognise when an element is the last of its kind within the parent structure. But some occurrence conditions require that this distinction be made. For example the last item in a list may need to inhibit a page break from separating it from the second last item. This occurrence recognition must therefore be done by the translation process.

The easiest way to accomplish this is to give each e-i-c in the FOSI a distinct name and to use that name, when appropriate, in the translation of the document instance. The right sides of Figures 1 and 3 show the T_EX translation of the SGML document instance and the sample FOSI fragment in the respective left sides. Notice how the two sets of `<para>...</para>` tags in Figure 1 are translated according to their occurrence within the `<section>`.

4. Implicit Specification of Characteristics.

Let us examine more closely the specification of the first para e-i-c in the FOSI fragment in Figure 3. It explicitly sets the values for the `firstln` characteristic of the "indent" category and the `startln` and `endln`

characteristics of the “textbrk” category. However, it neglects to explicitly define many other important formatting parameters. Nowhere was the font mentioned, nor the prespace, nor the justification (quadding). Nonetheless, as the formatted output suggests, these characteristics are well defined. In general, one of two implicit methods is used to determine the value of a characteristic not mentioned explicitly in an e-i-c.

One of the methods is inheritance. An unspecified characteristic that is inherited assumes the value it had at the level of its parent. In the example of Figure 1 above, the font family of the head is inherited from its parent (the section). If the font family characteristic for section is changed this will in turn affect the head. This method of determining the value of an unspecified characteristic has to be explicitly requested by setting the `inherit` attribute of the affected category to one, as shown in Figure 3. Explicitly assigned characteristic values override inherited values.

The usual method of determining the value of a characteristic that has not been explicitly assigned in the e-i-c is to look up its value in an environment. Every FOSI contains the document environment which explicitly mentions all 128 formatting characteristics. This is the default or “unnamed” environment normally used when a lookup must be done. For example, the prespace category (`presp`) was entirely left out of the declaration for head in Figure 3. So head was typeset using the default environment’s prespace characteristic values, which were all zero.

Other “named” environments may optionally be defined in the `envdesc` section. For an e-i-c’s characteristic to be looked up from a named environment, the structure in an e-i-c that contains the categories (`charlist`) must set its `envname` attribute to the environment name.

Of the two methods of determining the values of unspecified characteristics (inheritance from parent and defaulting from an environment) the inheritance method is the more problematic. Since the value of an inherited characteristic cannot be decided until the element’s context is known, current characteristic values need to be tracked by `TeX`. Fortunately `TeX`’s grouping already works this way. Characteristic values that need to be looked up from an environment can be either added to the definitions in the FOSI as part of the translation process, or the lookup can be performed by `TeX` as part of the typesetting process.

5. Typesetting the Translated SGML Document.

The processes performed by T_EX that culminate in typesetting the translated document can be separated into two levels. The top level is responsible for the inheritance, lookup, and setting of characteristic values, as discussed above. Macros, such as `\starcic` and `\endcic` used in the right side of Figure 3, group these values to restrict inheritance, while `\font`, `\textbrk` and the like are used to set explicit overrides.

The lowest level is responsible for the setting of T_EX parameters. This layer is invoked at the end of every start tag. In Figure 3 it is the call to `\eiccontent` that triggers this processing.

Various optimizations are possible. For example if the only category changed since the last text fragment was the leading category (which controls line spacing) then there is no reason to change the current font. By keeping track of the categories that have not changed since the last time the lowest layer was called we save the overhead of computing any T_EX parameter that relies entirely on those unchanged categories.

Whatever optimizations are used, it is required that the current font, horizontal and vertical size, margins, indent, interword space, page breaking, line breaking, and baselineskip parameters be properly set. Some non-primitive parameters (e.g., for controlling the number of columns) must also be set. In addition certain T_EX commands must be executed at the appropriate times, commands such as inserts, vertical and horizontal skips, counter increments, macro text expansions for typesetting, and so on. All of these actions must conform to the current settings of the FOSI characteristics.

Sometimes the correspondence between FOSI characteristics and T_EX capabilities is close, and a simple transformation will allow T_EX to produce the results specified by the FOSI. An example is the transformation of the pre-space category (`presp`) which controls vertical spacing. `Presp` contains characteristics called `minimum`, `nominal` and `maximum` which specify the whitespace that precedes an e-i-c. The actions T_EX must take can be defined by means of the transformation:

```
<presp nominal=x minimum=y maximum=z>  ──►
\vskip x plus min(z - x, 0) minus min(x - y, 0)
```

The indent category's characteristics are also easy to transform into T_EX. There are only three indent characteristics, all of which are dimensions:

leftind, rightind, and firstln. It is possible to specify that a dimension is absolute or relative to its current value. So assuming that the conditional `\ifabslind` is set to false if the leftind is specified relatively and to true if it is specified as an absolute value, and likewise assuming `\ifabsrind` and `\ifabsfind` are appropriately set, the transformation becomes:

```
<indent leftind=x rightind=y firstind=z>  →  
\ifabslind\else\advance\fi\leftskip x  
\ifabsrind\else\advance\fi\rightskip y  
\ifabsfind\else\advance\fi\parindent (z - x)
```

Another fairly straightforward transformation between FOSI characteristics and \TeX parameters is the font assignment. The FOSI font category includes characteristics named style, famname, size, posture, weight, width, allcap, smallcap, and offset. A table lookup scheme can be devised that allocates the fonts found on the users system based on the classification given by these characteristics. I would exclude allcap and offset from the classification as these are not really properties of a font.

6. Difficult or Impossible Transformations.

The three transformations listed above are among the easiest. The characteristics affecting one \TeX parameter do not necessarily come from a single category. Sometimes the transformation into \TeX requires a long and complex algorithm. The seemingly simple request `` would cause an element to interrupt the current column mode in a multicolumn document, balance off the existing text on the page, switch into one column mode for the duration of the element contents, and then switch back into the interrupted column mode. These changes would also effect any \TeX parameter whose setting depends on the `\hsize`. Nonetheless, multicolumn algorithms exist and the required side effects of switching column modes can be rigorously determined. So the span characteristic can, in theory, be implemented.

There are characteristics that are impossible to implement in \TeX . The category that controls page breaks, called keeps, has characteristics keep, widowct, and orphanct. The first is a toggle (0 or 1) that inhibits the breakability of the entire e-i-c. The other two are integers that control the number of widow or orhan lines to be kept together if the element must break. But \TeX only gives widow/orphan control for page breaks between

the first two and the last two lines of a paragraph. So the best transformation is only approximate:

```
<keeps keep=t widowct=a orphanct=b>  ──→
\ifcase t \interlinepenalty=10000
\else
  \widowpenalty=\ifnum a>1 10000 \else 150 \fi
  \clubpenalty= \ifnum b>1 10000 \else 150 \fi
\fi
```

The lettersp category concerns kerns between letter pairs. T_EX can be made to do “track kerning” in limited circumstances but the process is inefficient and the conditions under which it can be used are limited. There seems to be no point in attempting to implement this capability.

The quadding category controls justification of lines within an element. Among other possibilities, it gives the FOSI designer the power to request that paragraph lines be ragged on the inside margin only or the outside margin only. But T_EX cannot justify the lines of a single paragraph based on which page they fall on, at least not in a one pass system. This is yet another esoteric request that would not cause a book designer to lose any sleep if it were glossed over.

Still other FOSI capabilities can be implemented by using extensions to T_EX. The category that controls underscoring and overstriking (`hlight`) may require a T_EX extension or some driver assistance via `\special` commands. This same category gives control over the background and foreground colors.

7. T_EX Capabilities Not Expressible In a FOSI.

It is interesting to note that just as there are FOSI capabilities that are not possible to implement by T_EX, there are T_EX capabilities that cannot be described in a FOSI.

The `plain.tex` package already provides many typographical parameters to which the FOSI designer will have no access. Only parameters and capabilities that may need to be used in the middle of a document will be listed, since the macro package can set up the other parameters easily. The list includes: horizontal kerning; `\vboxes` and `\hboxes` to any fixed dimension; the capabilities of `\halign`, `\valign` and simple tabbing; mathematics and all parameters related to mathematics;

`\looseness`, `\parshape` and the paragraph hanging parameters; `\lineskip` and `\lineskiplimit` control; `\topskip`; multilingual hyphenation patterns; marks of various flavours; and `\xspaceskip` although interword space can be adjusted.

By adding macro packages the shortcomings of the FOSI grow. Add to the list: mixed multi-column modes on one page, although spanning to one column is possible; precise control of figure placement and many insert categories; side by side paragraphs; “picture” modes; multiple levels of footnotes; marginal notes; paragraph line numbering; and the list can go on.

In general the major advanced capabilities that \TeX has over FOSI capabilities are macro expandability, conditionals, and the ability to define custom output routines. For the time being, these are not serious limitations. It is more important to find an interim solution to the arbitrary DTD formatting problem. The FOSI-driven \TeX formatting engine provides a good solution. Its wide acceptance in the SGML community would also mean a wide acceptance of \TeX . A factor that would weigh strongly in \TeX 's favor.

Bibliography

- [DSSSL91] ISO/IEC JTC 1 INFORMATION TECHNOLOGY, “Information Technology — Text and Office Systems — Document Style Semantics and Specification Language (DSSSL) — DRAFT”, *International Organization for Standardization, International Electrotechnical Commission*, Ref. no. UDC 681.3.06 : 519.767, 1991.
- [SGML86] ISO WG8, “Information Technology — Text and Office Systems — Standard Generalized Markup Language (SGML) ISO 8879:1986(E)”, *International Organization for Standardization*, Ref. no. ISO8879:1986/A1:1988(E), 1988.
- [M28001A] US DoD, “MIL-M-28001A — Markup Requirements and Generic Style Specification for Electronic Printed Output and Exchange of Text. (CALs)”, *Naval Publications and Forms Center*, 20 July 1990.