Structured Document Preparation System *AutoLayouter*

Yoshiyuki Miyabe, Hiroshi Ohta, and Kazuhiro Tsuga
Matsushita Electric Industrial Co., Ltd., 1006 Kadoma, Kadoma-shi, Osaka 571 Japan
+81-6-906-4600. CSNET: `miyabe%isl.mei.co.jp@uunet.uu.net`

Abstract

We have developed a structured document preparation system *AutoLayouter*, which consists of an easy-to-use structured editor and a Japanese LaTeX based formatter.

Not only have we designed better user interfaces, but we have introduced a simple document structure. A document produced with *AutoLayouter* is a one-dimensional list with each node corresponding to a logical component of the document. This use of the simple structure largely contributed to making the editor easy to use but powerful enough both for editing the document structure and its contents, which may contain finer substructures.

Since we use the simple structure, we had to append various macros to complement the differences between our structure and the LaTeX begin-end environments. We also developed a device driver which converts `dvi` files to Kanji PostScript files.

Introduction

In Japan, most of the commonly used document processing systems are Japanese word processors, designed originally to produce beautiful documents without having to resort to hand writing. The most serious problem in early Japanese word processors was to make Japanese input efficient, easy, and fast. This problem has been almost solved by the development of efficient Kana (Japanese alphabet) to Kanji (Chinese character) conversion algorithms. Now we face the second step in document processing of Japanese.

From the beginning Japanese word processors have been designed with some layout facilities, using the fact that Japanese characters usually have the same width; thus spaces and tabs may be used to align them. In addition, Keisen characters, special line characters, were created to solve other alignment problems and to create tables of any shape.

One direction for the advancement of Japanese word processors is to augment the layout facilities, in order to format documents more flexibly, as English desk top publishing systems do.

On the other hand, our analysis of the Japanese word processor market led us to conclude that Japanese word processors are used with a great variety of documents and, therefore that some objective other than just the beautification of documents is indicated.

Among computer software people, Japanese versions of TeX and LaTeX are being used as replacements for Japanese word processors. But due to the complex syntax and the scarcity of supporting tools, it is extremely difficult for the typical users of word processors to take advantage of TeX's automated layout.

Basic Concepts of *AutoLayouter*

We developed *AutoLayouter* as a TeX based document preparation system whose objective is to give TeX's power to the users of Japanese word processors, and to induce them to write documents in more logical ways, similar to those used in making documents with LaTeX.

If the main task is to print documents, it is reasonable to use paper-oriented document preparation systems such as Japanese word processors or desk top publishing systems; but when the object is to manage documents, other means are required. Even in this case, if the size of a document is too large for it to be developed by only one person, one has to look for a better way to produce it.

Furthermore, when requirements of managing and delivering documents increase, working with such paper-oriented document preparation systems can cause serious problems:

Document file contains layout information mixed with text. In the word processor file, formatting information is mixed in with the text. For instance, special codes are used to specify the size and location of titles. But when manipulating the file in order, for example, to search the contents or reuse them, these codes interfere and must often be removed so that the text can be retrieved.

Document file format is formatting system dependent. When delivering a document to another person to be reworked, file format codes must be compatible for both sender and receiver. Maintaining such compatibility often creates impediments to further development of the system itself.

The *AutoLayouter* project represents an attempt to change the document prepartion style in Japan. We focused first on logical contents of documents, and put layout of the documents aside temporarily.

Architecture

Figure 1 shows the architecture of *AutoLayouter.*

We put structured document files in the kernel of the system, and in the near future, we shall develop a document database to manage them. To create the document files, we developed a structured editor. The editor checks the document structure whenever it is modified, and warns the user of invalid operations. So the document structure is always properly guaranteed.

The structured files are processed by a formatter that uses the Japanese version of LaTeX. We also developed various device drivers including a previewer and a PostScript converter for `dvi` files. The whole system was developed on Panasonic's Unix workstation BE using X Window System.

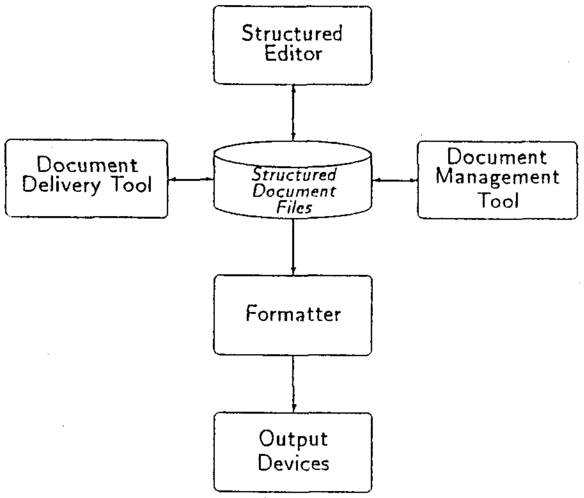


Figure 1: Architecture of *AutoLayouter*

Structured Editor

Problem of structure-driven editing. Mark-up languages such as SGML and LaTeX, when used with a typical text editor, have the following advantages:

Document portability. Users may select any text editor to make documents, so they can edit the documents on any machine.

Editing efficiency. Since mark-ups are natural extensions of the process of inputting the text, users can edit marked-up documents almost as efficiently as normal documents.

Unfortunately, the use of these mark-up systems may result in the creation of documents containing fatal syntax errors. Because the syntax rules are too complicated for many users, error correction may require more time than is reasonable.

Since our aim was to develop *AutoLayouter* for users of Japanese word processors, rather than for programmers, we developed a structured editor which may be used without any knowledge of markup languages; and, best of all, it never produces syntax errors.

Two kinds of structured editors may be considered:

Hypertext type. This type displays the document structure and the document contents in separate windows. So the editor normally consists of two parts, one of which handles the tree structure of the document while the other is concerned with the text contents of each tree node. Although one can view the document structure easily, it is difficult to read the contents of the entire document smoothly.

Tag embedded type. Roughly speaking, this type extends the character set to include mark-up tags which specify the document structure. Usually all the tags look similar, which makes it difficult to distinguish an important tag from unimportant ones. To solve this problem, some systems use different fonts for contents with different structures or they align the important tags outside the text field.

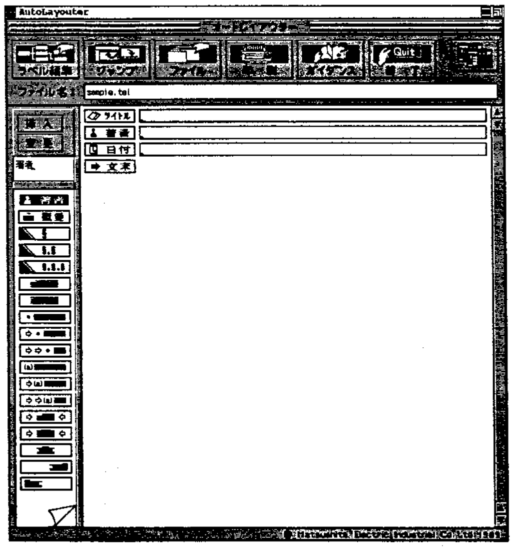
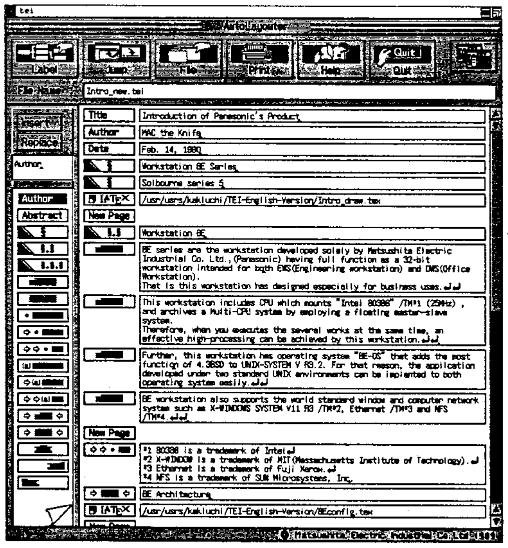


Figure 2: Snapshot of editor screen (in Japanese)



Intro_new.tei

Title: Introduction of Panasonic's Product

Author: MAC the Knife

Date: Feb. 14, 1990

§ Workstation BE Series

§ Solbourne series 5

LaTeX: /usr/users/kakiuchi/TEI-English-Version/Intro_draw.tex

New Page

§.§ Workstation BE

BE series are the workstation developed solely by Matsushita Electric Industrial Co. Ltd.,(Panasonic) having full function as a 32-bit workstation intended for both EWS(Engineering workstation) and OWS(Office Workstation).
That is this workstation has designed especially for business uses.

This workstation includes CPU which mounts "Intel 80386" /TM*1 (25MHz) , and archives a Multi-CPU system by employing a floating master-slave system.
Therefore, when you executes the several works at the same time, an effective high-processing can be achieved by this workstation.

Further, this workstation has operating system "BE-OS" that adds the most function of 4.3BSD to UNIX-SYSTEM V R3.2. For that reason, the application developed under two standard UNIX environments can be implented to both operating system easily.

BE workstation also supports the world standard window and computer network system such as X-WINDOWS SYSTEM V11 R3 /TM*2, Ethernet /TM*3 and NFS /TM*4.

New Page

*1 80386 is a trademark of Intel.
*2 X-WINDOW is a trademark of MIT(Massachusetts Institute of Technology).
*3 Ethernet is a trademark of Fuji Xerox.
*4 NFS is a trademark of SUN Microsystems, Inc.

BE Architecture

LaTeX: /usr/users/kakiuchi/TEI-English-Version/BEconfig.tex

Figure 3: Snapshot of editor screen (in English)

Document structure. We first designed a framework of document structures. We took the approach that the document structure provides ways to view the contents; and that different views may make it possible to perform appropriate processing. One view may show the logical or semantic structure of the document, and another its layout structure. These two structures must be distinct. The next design requirement is that the document structure should be simple enough to be understood by users of Japanese word processors.

Based on these initial ideas, we made a hierarchy of document structures as follows:

1) Logical structures indicate the semantics of the text. That is, logical structures may be used not only in formatting the document but searching its contents from a database and translating between different documents. The editor displays logical structures as iconic labels, outside the text field.
2) Structures specifying layout information only, such as indentation or font changes, may be embedded in the text as mark-ups. We modified a text editor so that it can handle these mark-ups as normal characters.
3) Footnotes and references are exceptional logical structures embedded in text, but should be distinguished from the layout structures.

Snapshots of the editor screen are shown in Figure 2 and Figure 3.

Table 1: Logical structure component attributes

Classification	Attributes
Component Id.	Label Name
Rules	Max. Occurrences Min. Occurrences Contents Type
Display Mode	Priority Levels Default Lines

Structure definition. The structure definition of a document type is quite simple when compared with a full-scale SGML.

An entire document is a one-dimensional list of logical structure components such as title and section, where the logical structure is specified using restricted regular expressions. Each component of the structure has attributes shown in Table 1. In the table, Maximum and Minimum occurrences for a component give the restrictions that apply to the regular expressions for components of the structure. Contents type can be text, file name, integer, PostScript, and LaTeX.

Substructures, such as layout structure to indicate indentation, font changes, and so on, are specified separately from the logical structure. The substructures are embedded in the contents text of the logical components in mark-up form. Mark-ups for the layout structures have a type that is one of quasi-character, begin-end, and toggle. Quasi-character type inserts layout objects such as skips and arrows. Begin-end type locally replaces a property of the characters between the marks, and toggle type changes a current property after the mark. Type information for the marks provides the formatter with instructions for recovering from mark-up errors.

Architecture. We show the system diagram of the structured editor in Figure 4, components of which are explained in detail below.

Structure definition parser. This parses structure definition files of the specified document type, and generates rules to be used in the structure editor.

Structure rule checker. When the user edits a structure component, the structure editor always asks the structure rule checker if the desired editing operation may produce an illegal structure. If the answer is affirmative, the structure rule checker returns the reason to the structure editor, and the editor displays a panel which explains why the desired operation is rejected.

Structure editor. Each component of the logical structure is displayed as an iconic label outside the text field, corresponding to its text contents. One can edit labels with mouse operations. For instance, to insert a label, the user clicks the insert button on the label panel, and selects the desired label from the label palette. Then the user specifies a label on the editor screen, and the desired label is inserted. This design is similar to that of a hypertext editor, mentioned above.

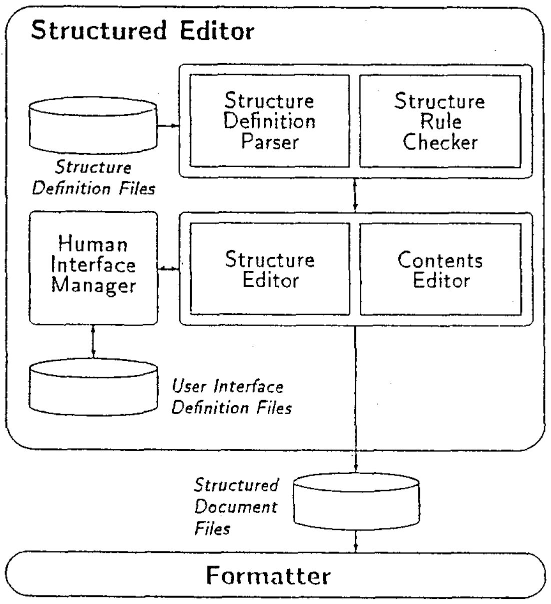

Structured Editor
Structure Definition Parser
Structure Rule Checker
Structure Definition Files
Human Interface Manager
Structure Editor
Contents Editor
User Interface Definition Files
Structured Document Files
Formatter

Figure 4: System diagram of the structured editor

Contents text editor. The contents text editor is a typical text editor customized using X Window System's text widget. Kana-to-Kanji conversion, which is managed in a front end processor, is performed in a special window, and the converted text is inserted by the text widget. Mark-ups specifying fine structures such as layout structures and references are inserted from a pop-up panel. Editing operations for the mark-ups can be restricted to maintain legal nesting of begin-end type mark-ups.

User interface manager. The editor's menus and panels may be changed to suit the type of document being edited. This is done using the on-line manual, which explains how to use labels and tags of the type of document currently being edited. We parameterized all the menus and panels which depend on the document type, and stored the interface definitions in the definition file. Each time the document type changes, the manager consults the definition file and resets the menus and panels appropriately.

Features. Our structured editor supports several editing styles:

Top-down editing. When the editing process consists of filling in the blanks in an existing form, the labels corresponding to the items on the form appear on the screen as default labels. When articles are being edited, the default structures include title, author, date, and abstract. Users may construct any document structure simply by using the mouse to insert or delete labels and to specify the field to which the new ones apply.

Bottom-up editing. Marking up a document is an example of bottom-up editing. We support replace, merge, and the splitting of labels by selecting these options from a pull down menu. Other useful forms of bottom-up editing involve cut-and-paste of text strings.

Outline editing. Outline editing is not really an editing feature, it is more like a different mode for viewing a document. When editing, users may select one of three display modes: normal, one-line, and selective. One-line mode displays only the first line of the text corresponding to each label, which allows the user to scan items thoughout the document. Similarly, selective mode displays only important labels, such as sections, and their contents, where the labels that are important are so designated in the label definition file.

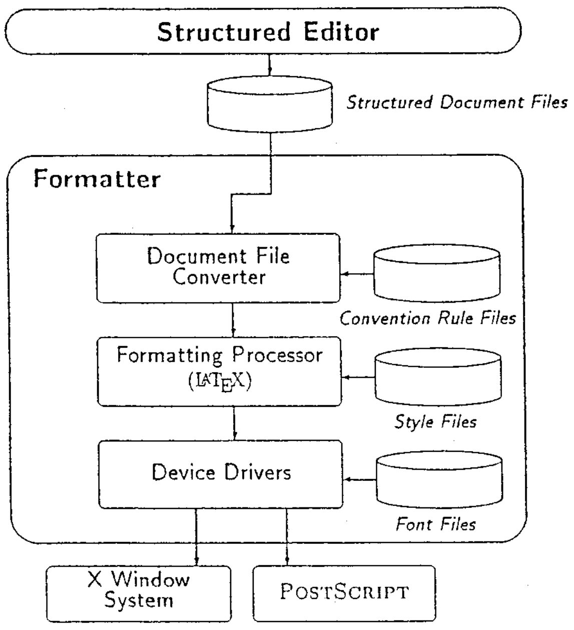


Figure 5: System diagram of the formatter

Formatter

Architecture. The system's block diagram of the formatter is shown in Figure 5. A detailed explanation of the components in the figure is as follows.

Formatting processor. We use the Japanese version of LaTeX as a formatting processor. TeX provides basic language facilities, such as macro definitions, as well as a simple formatting model which recursively constructs component boxes.

LaTeX appends various parts and environments that are convenient for writing articles, that simplify and speed up the preparation of style files.

Structured document file. The file consists of labels and their text contents as described in the document structure section.

Document file converter. This converter reads the structured document file, and scans each label and its contents. The conversion rules are stored in the conversion rule file of the document type. As described above, each label is converted to its corresponding macro, and its contents become the arguments of the macro. When the conversion is performed, contents are checked to verify that they satisfy the conditions required for their label, for example, kind of text type (text, integer, file name, reserved word, and so on).

Figure 6 shows how the document file converter works. A typical line of the structured document file is

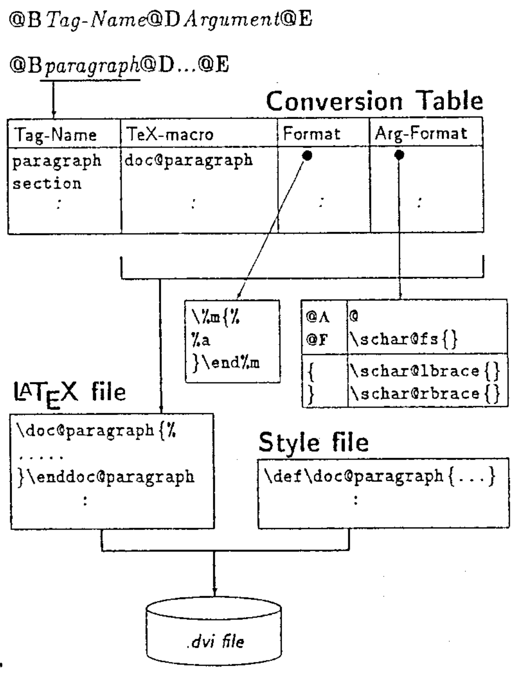
```
@BTag - Name@DArgument@E
```

The document file converter reads the conversion rules written in the conversion table and makes a LaTeX file.

Style file. On the basis of the document type, the formatter selects a corresponding style file. On the whole, conversion from logical structure to layout structure is a one-to-one mapping. One exception is represented by sequential restraints involving logical labels such as the requirement that the label "caption" must always be followed by the label "table". In such a case, style parameters will be modified. In order to allow for such conditional layout, we created a function to trace the sequence of input labels.

Device drivers. We developed the following device drivers to output `dvi` files to screen and printers:

Previewer. The previewer displays printer images of the formatted document on a X Window System. Since the physical resolution of the CRT differs greatly from that of current laser printers, we display the characters on the screen using fewer dots, scaling the same font used by the printer. When scaling the font, we maintain the quality of the display font by using an anti-aliasing technique, in which the gray scale of each dot is calculated in accordance with the number of black points in the sampling area of the original



```
\%m{%
%a
}\end%m
```

```
@^   @
@F   \schar@fs{}
{    \schar@lbrace{}
}    \schar@rbrace{}
```

```
\doc@paragraph{%
.....
}\enddoc@paragraph
```

```
\def\doc@paragraph{...}
```

Figure 6: Document file converter

font. Furthermore, the previewer displays a small magnifying window on the preview window, so that a finer image of the formatted document is also available.

Printer drivers. In addition to supporting raster image printers, we developed a converter from a `dvi` file to PostScript source code. With this converter, graphics files written in Encapsulated PostScript can be imported. In order to supply various fonts to the raster image devices, mentioned above, we also developed a font manager based on Japanese outline fonts. For English fonts, we use those supplied with TEX.

Concluding Remarks

Development of *AutoLayouter* is the first step in the construction of our new Japanese document processing system. The kernel of the system involves structured document files and, for this, we have developed an easy-to-use structured editor and a LATEX-based formatter.

In the future we plan to construct a document database in which structured documents are stored. We will also need a style file editor. At this time, style files are programmed directly using LATEX or TEX, which may prevent users from changing or creating their own styles; one possibility being considered is a WYSIWYG editor to specify layout.

Acknowledgements

The authors would like to thank T. Ohno and R. Kurasawa, who developed Japanese TEX, and Lincoln Durst for his suggestions on this paper.

Bibliography

Adobe Systems Incorporated. "Encapsulated PostScript Files", Specification Version 2.0. Mountain View, California, 1989.

ISO 8879. "Information Processing — Text And Office Systems — Standard Markup Language (SGML)". Geneva ISO, 1987.

Kurasawa, Ryoichi. "Japanese TEX at ASCII Corporation" (*in Japanese*). Proceedings of TEX Users Group Japan, TX-97-5 (September 1987).

Lamport, Leslie. *LATEX: A Document Preparation System.* Reading, Massachusetts: Addison-Wesley, 1983.