MFS: a modular text formatting system

by JAMES D. MOONEY
West Virginia University
Morgantown, WV

ABSTRACT

This paper presents the design goals and architecture of the Modular Formatting System, currently being developed at West Virginia University. MFS applies to formatters the principles of separation of function used in many successful program systems. A small central kernel forms the basis for a family of formatting systems, tailored to specific applications and environments.

MFS is intended to support a wide spectrum of applications drawn from the experience of commercial composition, word processing, and research systems. It does not rely on any specialized terminal or input source characteristics. Output devices from line printers through high resolution typesetters are handled in a uniform manner. Users can exercise detailed control over document appearance, or work exclusively with styles predefined through macros.
INTRODUCTION

Computer-aided composition of documents is an application which has developed rapidly since the early report preparation systems such as RUNOFF; programmed photo-composers from such companies as Star Parts, Photon, Compugraphic, Harris, and Mergenthaler; and mainframe formatting systems such as PAGE-1. Each of these lines has produced many more advanced descendants. TROFF is a powerful formatter in the RUNOFF family. Commercial systems now automate almost all the composition tasks of complete books or newspapers. Research systems such as \textsc{TEX} and Scribe have arisen which provide ambitious solutions to complex composition problems. The development of "word-processing" systems has made a new mode of interactive document preparation available in many environments, such as business offices, where it would not have been considered before.

As the notion of using a computer to prepare documents becomes more accepted and ubiquitous, many such systems are entering new environments where their full potential is only gradually discovered. It is characteristic that users will, if possible, apply this potential to tasks not thought of when the system was first acquired. New categories of documents may be processed; new, more sophisticated output devices may be obtained, and a variety of more (or less) powerful terminals may be added as text entry and editing becomes a more democratic process.

Expansion of the use of document preparation systems brings many persons into contact with the system who will use it only if use is extremely easy and natural, and few special codes or procedures need to be learned. At the same time, familiarity brings increased demands; and many users, once content with a typewritten product, develop the discernment of professional typographers. While taking full advantage of the automatic processing available from a powerful formatter, they may also want hairline control of the document's final appearance, and of the formatter's behavior in various specific situations.

These considerations demonstrate the need for a document processing system which, over its lifetime, can deal with a considerable diversity of input and editing mechanisms, target output devices, applications, and user demands for flexibility or convenience. Some existing systems are quite flexible along some of these dimensions, but no system known to the author appears to be sufficiently general in all ways. The Modular Formatting System (MFS) is an attempt to address this need.

DESIGN ISSUES AND GOALS

The overall objective of MFS is to synthesize the essential capabilities of previous formatting systems in a concise and modular system which can be adapted to a great variety of applications and environments. MFS establishes four specific goals of generality: input source independence, target device independence, support for a wide spectrum of applications, and user-selected levels of flexibility or convenience. Each of these goals will be discussed below.

Input source independence

Many formatting systems are limited to use with specific terminal types. This is true especially of word processors, which are most often supplied by the terminal vendor. In contrast, many user environments will progress from a few simple terminals to many diverse input sources as their system usage evolves.

The likely variations in input source characteristics include coding format and degree of interactivity. The coding conventions of any particular terminal can be mapped into a standard form such as ASCII by an input module matched to the terminal. If the original input includes graphic information such as specific fonts, special characters, or spacing, this information also must be converted to a suitable command stream by the input module.

Like other software systems, the early document processing programs were all designed to process complete marked-up documents in batch mode. The rise of screen-oriented editors with some formatting capabilities (word processors) parallels the rise of the interactive user interface. In this environment, users can modify their documents incrementally, and see immediate results from their formatting instructions.

On the other hand, interactive computing has not made popular such tools as incremental (interpretive) compilers, except for very simple languages and programs. Most compiling is still performed on complete programs without interaction. A similar pattern exists for document formatting. Editing programs insert and delete text, and perhaps carry out simple line filling incrementally. More elaborate formatting, often requiring knowledge of a wide context, is invoked by distinct commands on well-defined units of text.

A further impediment to interactive document creation occurs since there is most often a mismatch between the abilities of the terminal and the target output device to represent text. Some applications target very simple printers, and some terminals have high-resolution, bit-mapped displays, but in most cases the terminal can only approximate the final output appearance. This approximately processed text, while useful as a proof reference, may be more difficult to work with than the original, marked-up document.

For these reasons, MFS can be viewed essentially as operating on a batch of input, which may be digested in its entirety before output is produced. It should not, however, ignore the
possibility that it was invoked interactively. Decisions may arise in processing, such as questionable line breaks or hyphenation, meaningless commands, or impossible typographic situations. Where appropriate, the relevant modules of MFS should be able to query the user for a solution to these problems.

**Target device independence**

Only a few existing formatters (e.g. SCRIBE) are comfortable with a full spectrum of output devices. TEX relies on a variety of special characters and use of its own fonts, and is not easily mapped down to simpler devices. TROFF drives a single sort of typesetter, and has a separate version, NROFF, for simple printers. Word processors know little of multiple fonts and precise spacing control. The limitations of vendorsupplied systems are evident.

A major goal of MFS is to be able to drive a wide class of output devices, making full use of their typographic capabilities, and to defer the choice of device as much as possible while processing.

The simplest devices are those of the "line-printer" class with a single character font and fixed character and line spacing. Advanced printers, phototypesetters, etc. may provide many fonts, precision spacing, and other variables. For much routine text processing the simplest printers are adequate, and they are heavily used. But advanced devices are appearing side by side with line-printers in a growing number of environments. The choice of device for each job is then a matter of taste and economics.

A user who works with a variety of target devices will prefer to view them through a common formatter, controlled by a uniform coding mechanism.

In most processing, the user will have in mind a specific target device and configuration when the document is prepared. Indeed, character selection and many creative format decisions must usually be made in view of the capabilities of the output device. However, in some applications a document’s life cycle will go beyond immediate printing on a single device. It may be printed from time to time on several devices; it may be distributed to various sites to be printed in a variety of environments; it may be printed on an unexpected device if the intended target is unavailable. In these situations it is generally acceptable to reprocess the source document through the formatter with the actual target specified; but it may not be acceptable to require manual re-editing of the document.

These considerations impose the following requirements on MFS:

1. It should be able to access all foreseeable capabilities of the various devices for character selection, transformation, positioning, etc.
2. It should allow the user to prepare documents for a *virtual target* not necessarily bound to any single real device.
3. It should, like SCRIBE, try to provide a reasonable interpretation if the specified characters or processing functions are not available on the specified device.

Further, it should not be necessary (as in TEX) to specify characters in a way which depends on their grouping within the output device. Users should be free to work with *virtual fonts*, character groupings formed for the convenience of the application.

**Wide spectrum of applications**

The primitive functions available through various formatters vary widely, directly affecting the class of application problems which can be addressed. Many recent formatters offer valuable solutions to complex problems, e.g. math formula layout, compiling indexes and bibliographies, etc. At the same time, capabilities which have proven invaluable in commercial composition have not been carried through to newer systems. Word processors may offer interesting interfaces to data processing functions, yet never address many typographic needs.

All of the possible capabilities serve ultimately to define mappings from input strings (text and command information) to output strings (character specifications and positions). However, many of these mappings are highly context dependent. A principal goal of MFS is to identify a set of primitive functions which support as large a consistent set of capabilities as possible, and to provide these functions in the MFS kernel.

The following is a partial list of capabilities which have been proven useful by various formatting systems. Preserving these capabilities has significant implications for a formatter architecture. Syntactic issues, which have less impact, are considered in the next subsection.

**Line breaking and justification**

For some typographers, the best justification algorithm is a very personal matter. We may consider a single line or entire paragraphs in making decisions. Excess space may be distributed among word breaks and between letters using various strategies. There should be provision for overriding "quad spaces" before, after, or within any text, and for filling these spaces with characters which may need to meet some criteria for alignment.

**Hyphenation**

Good hyphenation is highly subjective, and the algorithm must be variable. Technical and geographic words may have to be added to the dictionary for specific jobs. Different languages require entirely different approaches, occasionally within the same job. This need is recognized in most commercial systems, while research systems have treated even simple hyphenation as an afterthought.

**Multiple environments**

Ability to save and switch the prevailing collection of formatting parameters is necessary for clean merging of intermixed text such as footnotes. This facility exists in most systems only in a limited or specialized form.
Deferred input sequences

It is often necessary to specify a sequence of commands or text to be deferred until some typographic condition occurs, e.g., after 6 inches of text. This ability, found to some degree in most commercial systems, supports picture cuts, running heads, floating tables, and other important structures. The deferred input mechanism is found in limited form in TROFF, and missing from TEX and SCRIBE.

Deferred output

The ability to save text after processing, for insertion into the output at a later point, is the software analog of “cut and paste.” This function, important for footnotes, floating tables, etc., is well developed in the “diversions” of TROFF and the “boxes” of TEX.

Text measurement

The ability to preprocess text to determine its dimensions or other characteristics before output is a special case of Deferred Output. These measurements may be needed for problems such as footnote fitting, large initial capitals, vertical justification, etc. Many systems provide only the ability to determine the width of a short string.

Numeric variables

Various capabilities are made possible by use of numeric variables (normally integer) which can be set or tested and manipulated with simple arithmetic. Examples are numbering of pages and other text items, or setting conditions which will be tested.

Readable state variables

As a special case of numeric variables, it is valuable to be able to determine and test parameters of the current typographic context, such as current coordinates on the output page.

Special displays

Various applications have requirements to allow manipulation of specialized display material in a convenient and appropriate manner. Examples are mathematical formulas as addressed by TEX, and layout of advertising copy which is a major concern of commercial systems.

Multilingual support

A comprehensive formatter must address the specialized needs of specific languages. Problems not arising in English text may include setting right to left (Arabic) or top to bottom (Chinese); very large alphabets; positioning accents and building up characters from parts; or hyphenation which modifies the resulting partial words.

Transformation systems

Many formatters include specialized subsystems which (eventually) generate output text after digesting various kinds of input information. Examples include the bibliographies and indexes of SCRIBE and the numerical calculations of some word processors, as well as number-to-string conversions and assorted manipulations of the date and time. Some of these functions require knowledge of the typographic context, and cannot be viewed as preprocessors.

Conditionals

In general a conditional is a transformation system yielding a logical value instead of an output string. This value may be used to select between two different well-defined input sequences.

Graphic material

Some output devices are able to generate graphic images as well as text characters. A formatter is not likely to contain a picture definition facility; but it should be able to accept picture descriptions in which the primitives known to the output system, whether dots, vectors, or higher subpictures, are presented as special characters. The resulting graphics may be mixed with text; and interaction with systems such as PIC®, as preprocessors or transformation systems, should be a possibility.

User-selected flexibility and convenience

The user’s wishes for an ideal view of the system include various tradeoffs. Flexibility, convenience, error tolerance, economy of input, and readability of marked-up text are all desirable characteristics which cannot be maximized at the same time. Markup may be desired to be independent of the output device, or to take advantage of features of a particular output device. To meet these needs the user interface should be easily customized.

A part of this interface is the domain of the input or editing system rather than the formatter; and advanced input systems may hide much markup and generate it automatically in any required format. However, the formatter needs to be concerned with the flexibility of the interface presented to very simple input systems.

The basic approach to meeting these needs is to provide a flexible, low-level command language and a powerful macro facility. The command language alone provides all possible direct control of formatting details, while the macro facility makes possible selected levels of abstraction culminating in a minimum-markup, non-procedural language such as GML. Both TROFF and TEX provide good examples of powerful...
command and macro systems, but neither meets all of the following desirable requirements:

1. The complete set of input characters which will have special meaning, such as command prefixes, should be user-selectable. It should be possible to keep the number of such characters extremely small.
2. All names for commands, macros, variables, etc. should be user-selectable. The range of possible names should run from single characters to long, descriptive names.
3. Command and macro arguments, or the range of text affected, may consist of a single character or many lines of input. Any range should be selectable by suitable bracketing. Commands should be acceptable anywhere regardless of input line endings.
4. The user or application should be able to dynamically define virtual fonts, complete mappings from input text characters into the set of presumably available output characters. This mapping should include both single input characters and strings (named characters, ligatures, etc.). It should hide any differences in the actual access mechanism for the selected character set.
5. The significance of spaces and line endings in the input text should be consistent and intuitive in any context.

In addition, to support a modular organization of the documents themselves, there should be a nested inclusion mechanism for the component files of the input text. This mechanism should make use of string variables to avoid embedding system-dependent file names in the document itself.

SYSTEM ARCHITECTURE

The MFS architecture is designed as a small, central kernel interacting with a collection of processing modules. Like the kernel of an operating system, the MFS kernel is intended to provide only essential, primitive support functions while making few restrictions on possible processing mechanisms. This organization is shown in Figure 1.

The division of the system into modules provides a clear separation of functions, isolating capabilities that may be independently varied. The hyphenation system may be replaced without impact on the rest of the system. Similarly, MFS may be transported to a different computing system with changes only in isolated modules.

The system kernel is responsible for control, sequencing, and storage. Input, initially from the standard source, is accepted by the input translator and passed to the kernel as a sequence of primitive commands. This input may include macro text to be deferred until a given condition occurs. The kernel will store the text and flag the appropriate internal variables. When a change to the required condition occurs, the saved macro will be rerouted through the input translator.

The received text is then "justified" in the context of the current environment. The kernel maintains a collection of named control records representing the environments, which can be switched on request. The information in these records includes format control parameters, current state descriptors, and user-controlled variables. Some information also remains in a global environment which is not switched.

The justification module constructs lines, determining line endings and processing various spacing parameters. Several algorithms of varying sophistication may be selected. The extent to which a single algorithm can be parameterized, and the degree to which this module can be unbound from the kernel, are subjects of current investigation.

Transformation systems are handled as distinct modules using a standard interface. They are viewed as collections of special commands which can accept arguments and insert text into the processing stream. Mechanisms such as indexing can thus be "piggybacked" onto MFS even if they were not planned in the original design.

The operating system interface module provides the only link to the underlying computing system, enhancing portability. Input and output routines, file name translation, date and time, and current invocation environment and options are all processed through this module.

The MFS output is a sequence of character selection and positioning codes in a device-independent format. However, processing at any time is done with the expectation of a particular output device and configuration. Characters are accessed with the understanding that they will be available on the actual output. In addition to character tables, a device table is used to determine significant characteristics of the output device. The minimum spacing increments are provided, for example; if these are coarse, it may be preferable during justification to round values at every stage of calculation. The ability of the output mechanism to back up, horizontally or vertically, may affect the processing strategy.

In a given environment, current output may be routed directly to the output stream or saved in a buffer. The deferred outputs may then be placed in the output stream at a later stage, perhaps with offset to a different position.

The output file, perhaps at a later time, is processed by an output translator which assembles coding for the particular output device, and generates the physical copy.

CONCLUSION

We have presented an architecture for a text formatting system, MFS, which is currently being developed at West Virginia University. The MFS design is more flexible than known current systems. It covers a wide range of applications although the program modules may be compact. Any of a wide
range of output devices can be driven with suitable translators and tables. Device characteristics are not built into the formatter.

A system with this architecture would enhance the interchangeability of document files and would be especially useful in environments where diverse applications may exist and a variety of output devices may be available over a period of time.

In addition, due to its modularity, MFS will serve as a test bed in which different hyphenation algorithms, command languages, etc., may be tried and compared. With other aspects of the formatter held constant, the effects of changes in a particular subsystem can be more easily studied.

REFERENCES

MANAGEMENT ISSUES/
DECISION SCIENCE SUPPORT SYSTEMS