Massachusetts Institute of Technology
Artificial Intelligence Laboratory

Working Paper 257 May 1984

TEMPEST -- A Template Editor for Structured Text

Peter Sterpe

Abstract

This paper proposes an editing tool named TEMPEST (TEMPlate Editor for Structured Text) whose goal is to extend a text editing environment by using templates to incorporate into it some knowledge of the structure of the text that is being edited. TEMPEST's functionality is focused on the structural aspects of text editing that are not well supported by typical text editors. In addition, it uses a text-based approach which affords a wide range of applicability. A scenario is given to illustrate its use.

A.I. Laboratory Working Papers are produced for internal circulation, and may contain information that is, for example, too preliminary or too detailed for formal publication. It is not intended that they should be considered papers to which reference can be made in the literature.

© MASSACHUSETTS INSTITUTE OF TECHNOLOGY 1984

Table of Contents

Introduction

This proposal describes an editing tool whose goal is to extend a text editing environment by using *templates* to incorporate into it some knowledge of the structure of the text that is edited. The editor's functionality will be focused on the structural aspects of text editing, particularly the "cut and paste" style of editing in which a document is built up in pieces and the pieces reshuffled until the correct form is achieved. The emphasis on structure will enable the user to work not only at the character level, but also in terms of logical pieces of the text.

The editor, named TEMPEST (TEMPlate Editor for Structured Text), will be written as an extension to the MINCE[1] text editor. The extension consists of a library of templates and a set of commands for template manipulation. A template is a skeletal form for a piece of text. Some of the text is supplied; other parts, known as the template's *slots*, must be filled in by the user. A typical use of TEMPEST will involve bringing a template into the buffer and filling (and possibly rearranging) its slots through a combination of direct editing and TEMPEST commands. The resulting document is kept as an ordinary text file in the user's environment and may be used (but not edited) by other programs. TEMPEST retains its knowledge of the document's structure from one session to another.

TEMPEST expects to provide several advantages over ordinary editing. Editing with templates allows the user to think of a document abstractly, in terms of its functional parts, rather than its words and characters. For example, one would request the editor to "go to the return address" rather than perform a "reverse string search" on some of the characters in the return address. It is felt that a more abstract view of editing makes the user's work less tedious and more productive. Following are the key features of TEMPEST:

Structured Text -- TEMPEST views the text both as a stream of characters and as a tree of named parts (i.e., the slots of a template). This means that the user may operate directly on the structure without having to move blocks of text. This also means, however, that the user can edit the text directly without having to be concerned with the structure. In fact, all changes to the text are also interpreted in the context of the structure, as if structural commands had been issued. This is in contrast to typical structure editors in which free form insertions and deletions are not allowed.

Cut and Paste -- Documents are often written with a "cut and paste" style of editing. TEMPEST supports this by providing commands for collecting the parts of a template into a whole, even if they are out of order or if some are missing. Sections of text can also be tagged for later reordering, just as one would number paragraphs in the margin. These features are present in TEMPEST in order to capture some of the aspects of pencil and paper editing which are used to control structure and which are typically not available in text editors.

Constraints -- TEMPEST can exert some control over the values of a template's slots, either by noting errors or by propagating values. This frees the user from managing bookkeeping details. The constraint facility could also be expanded to monitor the structure of a template and prevent illegal insertions, deletions, or reordering.

TEMPEST's structural knowledge is text based, and is therefore not specific to any one domain. This is a restriction on the depth that TEMPEST can achieve, but it also enables TEMPEST to have a broad range of applicability.

---

[1] MINCE is a product of Mark of the Unicorn, Inc.

An issue of concern is how to achieve significant functional power while still basing the editing upon the text of a document, rather than on some more complex underlying representation which would involve considerable processing overhead. The motivation behind this is to host TEMPEST on a modestly powerful and easily accessible machine, through which many diverse users might benefit from such a tool. Unfortunately such a machine would be incapable of responding quickly under a heavy processing load, so the text based approach is a necessary restriction.

The next section of this proposal shows a scenario in which TEMPEST is used to create a business letter and a short program. After the scenario is a section on the format and usage of the template library, followed by a sketch of how the implementation of TEMPEST will proceed. The final two sections discuss the relationship of TEMPEST to some similar programs and its limitations and possible extensions.

Scenario

In this scenario, the user constructs a business letter to be processed by the SCRIBE [Reid 80] text formatter. TEMPEST is used to instantiate and fill a template *business-letter* which represents the correct format for a SCRIBE FormalLetter and contains the necessary SCRIBE formatting commands. The relationship between the template and the SCRIBE commands is discussed in Screen 2.

Each step of the scenario is represented by a view of the screen after the command is executed. A summary of what is typed at each step is shown on one or more command lines above the screen. Text in bold italics is supplied by TEMPEST in response to a control sequence which is shown at the beginning of the command line; other text is supplied by the user. Each control sequence begins by typing "ESC" (the ESCAPE key), followed by "T", followed by an abbreviation for the desired command. The user need only type enough characters of the command name to specify it uniquely. A space terminates the control sequence, after which TEMPEST prompts with the command name in full. Within the screen view, any characters that have changed due to the command are shown in bold italics.

In Screen 1 the user invokes the USE LANGUAGE command. All templates in TEMPEST are associated with a particular language. For example, the library may contain many business-letter templates, but this command ensures that the one for the language SCRIBE will be used. Any templates used later in the scenario will also have SCRIBE as their associated language.

The benefit of attributing a language to a template is that different versions of a template may be referred to by the same name. This frees the user from always having to use the language name when choosing a template. This can be especially helpful when writing programs with templates, since each programming language has its own representation for certain common elements -- for example, a comment in C is delimited differently than a comment in FORTRAN.

The language feature does not change the behavior of TEMPEST. Thus, it is not like a language mode as would be found in some editors, EMACS for example [Stallman 81]. It is essentially a convenience feature for keeping multiple versions of a template under the same name. In general, any language or other editing modes in use have no effect upon TEMPEST's behavior, although they will have their prescribed effect upon the MINCE editor that is hosting TEMPEST.

<the user begins editing the new file LETTER5.MSS>
<ESC>-T-U *USE LANGUAGE* Scribe

■

TEMPEST (*Scribe*) Main: LETTER5.MSS

In Screen 2, the INSERT command is used to enter the business-letter template into the buffer. In general this command is for placing a template into the text where there isn't a template or a slot already.

The screen view illustrates the typical form of a template. Constructions of the form "{..., ...}" are slots. The text after the comma is the name of the slot, and the text before the comma is an explicit default value for the slot. If a slot has no default, only the slot name will appear between the braces. Any commas in the default text must be preceded by a backslash ("\").

The text of this template is a combination of SCRIBE formatting commands and slots. SCRIBE is a high level text formatter with which the user creates documents of types known to the program through a database of predefined *document types*. The parts of a document are called its *environments*. The document to be created is specified with the Make command (e.g., @Make(FormalLetter)), and each environment is filled by enclosing its text in named delimiters (e.g., @Begin(Body) <text of body> @End(Body)). (Note that SCRIBE commands are preceded by the '@' character.) The user must specify the environments in the proper order.

A FormalLetter has the following environments: Letterhead, DateLine, InsideAddress, Body, ComplimentaryClose, SignatureBlock, and Notations. Each environment requires one or more pieces of information, each of which is represented in the template as a slot. These slots guide the user in correctly constructing a FormalLetter. In general, the slots correspond to the logical parts of the text, but they may also be used to contain other information the user must specify. For example, the printer slot is used to specify an output device for the "@Device" command.

Notice that the two slot names on line 15 are distinguished by the words "of the salutation." This is an indication that these slots are actually part of another template which is nested inside the business-letter at this position. This is described more fully at Screen 6.

<ESC>-T-IN *INSERT* a business-letter

```
@Device(Dover,the printer})
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
{the from-address}
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
{the to-address}
@End(InsideAddress)
@Begin(Body)
Dear {the title of the salutation} {the name of the salutation}:
@Blankspace(1 line)
{the text}
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
{the sender}
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)
```

TEMPEST (Scribe) Main: LETTER5.MSS

The DOCUMENT command furnishes a brief description of the structure of the named template. This facility describes the template as it exists in the library, not as it exists in the buffer. It is intended as an on-line reference manual to refresh the user's memory on how to use a template. The display temporarily covers part of the screen. Some of the features in the documentation are described below.

SLOT:TYPE

This section lists the slots (in order) and their type. Every template and slot has a type which is assigned to it at definition time. All the templates known to TEMPEST are arranged in a tree structure; type B is compatible with type A if B is a descendant (at any depth) of A in the type tree. A slot can only be filled with a value of a compatible type. Users are free to create new types when defining templates. Not all the slots listed here actually appear in the text of the template, for example, the salutation and the date. These are slots which have an "implicit default", as described below.

CONSTRAINTS

At the time a template is defined, value constraints may be placed on some or all of its slots. For example, the values of two slots could be constrained to be equal. In order that they be easy to express, value constraints only apply to two slots. However, a chain of constrained slots can be set up by involving some slots in more than one constraint.

In this screen, all the constraints are of the same nature -- the constrained slots must have the same value. Notice that two slots are mentioned which do not yet appear: the name of the to-address and the title of the to-address. These are mentioned in the (likely) event that the to-address is implemented with a template which has a "name" slot and a "title" slot. If these slots are put in the text, the constraints will be enforced and the slots' values will propagate; otherwise the constraints are ignored.

In the case of an equality constraint, whenever one slot is given a new value, TEMPEST propagates that value to its partner. This constraint can only be defeated by using the FORCE command which will implement a slot without checking constraints. TEMPEST enforces inequality constraints by refusing to accept a value that does not meet the constraint. Again, the FORCE command can be used to override value constraints.

DEFAULTS

This section lists slots with defaults. A default value that is simply text is placed in double quotes. If the default is a template, its name is given, unquoted, in lower case; if it is a built-in function, the function name is given followed by parentheses to indicate that it is a function. The type of the default, EXPLICIT or IMPLICIT, is also given. See Screen 4 for a more detailed discussion of defaults.

<ESC>-T-DO *DOCUMENT THE TEMPLATE* business-letter

DOCUMENTATION OF THE TEMPLATE BUSINESS-LETTER

SLOT:TYPE

printer:TEXT	*from-address:ADDRESS*	*date:DATE*
to-address:ADDRESS	*salutation:SALUTATION*	*text:DOC-SECTION*
closing:TEXT	*sender:NAME*	*business-title:TEXT*
identification-initials:TEXT		

CONSTRAINTS

Value(the sender) = Value(the name of the from-address)
Value(the title of the salutation) = Value(the title of the to-address)
Value(the name of the salutation) = Value(the name of the to-address)

PARAMETERS

None

DEFAULTS

printer:	*EXPLICIT*	*"Dover"*
date:	*IMPLICIT*	*CurrentDate()*
salutation:	*IMPLICIT*	*formal-salutation*

COMMENT

This template implements a "modified block" business letter.

■

TEMPEST (Scribe) Main: LETTER5.MSS

Screen 4 shows the use of the ACCEPT command to keep an explicit default. An explicit default is one in which the default value is included with the slot name. For example, on the first line of Screen 2, the default value "Dover" is provided with the printer slot. There are also "implicit" defaults, which are used for slots whose value is likely to be the same all the time. In that case, the slot itself is not shown with a default. Rather, the default value is placed directly in the text of the template, as if there were no slot there at all. The DOCUMENT or SHOW commands can be used to remind the user that there is, in fact, a slot there. All defaults are suggestions and can be overridden by the user.

When the default is accepted, the surrounding slot text disappears. On the command line, TEMPEST prompts "ACCEPT THE PRINTER" although the control sequence did not mention the printer. In general this command assumes the user wishes to accept the next unaccepted default (i.e., the first unaccepted default encountered when proceeding forward through the text from the current cursor location).

It should also be mentioned that the cursor has progressed to the from-address slot. After most TEMPEST commands, the cursor is moved to the next unfilled slot in anticipation that the user will next focus attention there.

<ESC>-T-AC *ACCEPT THE PRINTER*

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
{the from-address}
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
{the to-address}
@End(InsideAddress)
@Begin(Body)
Dear {the title of the salutation} {the name of the salutation}:
@Blankspace(1 line)
{the text}
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
{the sender}
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)
```

TEMPEST (Scribe) Main: LETTER5.MSS

The IMPLEMENT command is used for giving values to slots. It has more than one usage, one of which is shown here. In this instance, the value is the template my-address whose slots were implemented with implicit defaults when the template was defined. TEMPEST assumes that the user wishes to implement the from-address since that was the next unimplemented slot when the command was invoked.

As shown in Screen 3, there exists the constraint that the sender should equal the name of the from-address. Since the name of the from-address was just implemented, its value propagates to the sender slot where the text "M. K. Bayer" appears. (The fact that "M. K. Bayer" is indeed the name slot of my-address is implicit here. The DOCUMENT command could be used to verify that the my-address template does in fact have a name slot.)

<ESC>-T-IM *IMPLEMENT THE FROM-ADDRESS AS* my-address

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
M. K. Bayer
362 Memorial Dr.
Cambridge, MA  02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
{the to-address}
@End(InsideAddress)
@Begin(Body)
Dear {the title of the salutation} {the name of the salutation}:
@Blankspace(1 line)
{the text}
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)
```

TEMPEST (Scribe) Main: LETTER5.MSS

Screen 6 gives an example of implementing a slot with a template whose slots have not already been filled. It is not apparent that the slots of the personal-address template are not really slots of the business-letter template, since they are not distinguished in any way. This is in contrast to the formal-salutation, whose title and name slots are distinguished by the words "of the salutation."

Because the business-letter was defined with its salutation already implemented as a formal-salutation, those slots are represented with more explicit names as a cue to the user. Since the user chose to use the personal-address, TEMPEST assumes he knows where its slots come from (or that he will use the EXPLAIN command if he forgets). In general more explicit slot names are only used for templates that are included as part of another template at definition time. This is done to reduce visual clutter.

This screen illustrates one more naming convention. Notice that the first line of the personal-address is broken into four slots, while the second line contains only one. Conceivably the number-and-street could be broken into a few slots as well, but this was not done because the various parts are not likely to be referred to separately. While one might refer to the person as "Mitchell" or "Mr. Bayer", one would never refer to his address as "362" or "Memorial", so it would only create extra overhead to separate the number-and-street into two or three slots.

<ESC>-T-IM *IMPLEMENT THE TO-ADDRESS AS* a personal-address

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
M. K. Bayer
362 Memorial Dr.
Cambridge, MA  02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
{the title} {the first-name} {the middle-initial} {the last-name}
{the number-and-street}
{the city}, {the state}  {the zip}
@End(InsideAddress)
@Begin(Body)
Dear {the title of the salutation} {the name of the salutation}:
@Blankspace(1 line)
{the text}
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)
```

TEMPEST (Scribe) Main: LETTER5.MSS

In Screen 7, several slots are implemented without invoking the IMPLEMENT command by name. (This is an implicit usage of the IMPLEMENT command.) To do this, the cursor is moved onto the text of a slot, anywhere between the opening brace ("{") and the closing brace ("}").

```
{the title} {the first-name} {the middle-initial} {the last-name}
```

Next, the first character of the value is typed. Once this happens, the slot name disappears.

```
M{the first-name} {the middle-initial} {the last-name}
```

Each subsequent character is inserted to the right of the last as it is typed.

```
Mr.{the first-name} {the middle-initial} {the last-name}
```

All characters typed are taken as part of the slot value until a command is issued which causes the cursor to move off of the slot. Typically the TEMPEST control sequence for "next slot" (<ESC>-N) will be used to terminate a slot implementation. The cursor advances to the first character of the next slot, which for an unimplemented slot is its opening brace ("{").

```
Mr. {the first-name} {the middle-initial} {the last-name}
```

Although the IMPLEMENT command was not invoked explicitly, TEMPEST will interpret these actions as if it had been. As a consequence of this, value propagation takes place between the title of the to-address and the title of the salutation, and also between the last-name of the to-address and the name of the salutation, since these slots are constrained to have equal values.

<direct editing of the inside address>

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
M. K. Bayer
362 Memorial Dr.
Cambridge, MA  02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL  60609
@End(InsideAddress)
@Begin(Body)
Dear Mr. Whiting:
@Blankspace(1 line)
{the text}
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)
```

TEMPEST (Scribe) Main: LETTER5.MSS

This command produces a list of unimplemented slots. The display temporarily covers part of the screen.

<ESC>-T-W *WHAT'S LEFT?*

SLOTS NOT YET IMPLEMENTED

the text *the business-title* *the identification-initials*

■

362 Memorial Dr.
Cambridge, MA 02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL 60609
@End(InsideAddress)
@Begin(Body)
Dear Mr. Whiting:
@Blankspace(1 line)
{the text}
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)

TEMPEST (Scribe) Main: LETTER5.MSS

This screen illustrates the use of the special TEMPEST form "list." A list of any kind of template is simply zero or more of them in succession with no other templates intervening. In this case, the list elements are block-paragraphs. A block-paragraph is a particular style of paragraph that is typed flush with the left margin. The first line is not indented.

It is not obvious from the text that there is a list of templates anywhere. At the moment, there is only a slot for one paragraph-body. TEMPEST understands that there really is a list there, however, and will produce another paragraph-body slot if the user invokes the ANOTHER command. It will not be necessary to say where the next paragraph should go -- TEMPEST will automatically place it after the last one.

<ESC>-T-IM ***IMPLEMENT THE TEXT AS*** a list of block-paragraph

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
M. K. Bayer
362 Memorial Dr.
Cambridge, MA  02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL  60609
@End(InsideAddress)
@Begin(Body)
Dear Mr. Whiting:
@Blankspace(1 line)
{the paragraph-body}
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)
```

TEMPEST (Scribe) Main: LETTER5.MSS

In this step the paragraph-body is implemented by direct editing as was done in Screen 7. Since no command is issued which would move the cursor off of the paragraph-body, TEMPEST keeps the cursor at the end of the typed-in text.

<direct editing>

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
M. K. Bayer
362 Memorial Dr.
Cambridge, MA  02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL  60609
@End(InsideAddress)
@Begin(Body)
Dear Mr. Whiting:
@Blankspace(1 line)
It has come to my attention that you would like us to design a
piano for your pet parakeet. While I am sure that Arthur would be a
fine musician, may I suggest that you choose an instrument which
requires less dexterity? Perhaps Arthur would enjoy taking up
whistling.  I understand that birds are naturals at it, and just think
of the money you would save on sheet music.■
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)
```

TEMPEST (Scribe) Main: LETTER5.MSS

The ANOTHER command is used exclusively with lists to generate another element in a list. Since TEMPEST knows where the list of block-paragraphs is situated in the buffer, it automatically places a new one in the correct place. In addition, it places a blank line between the first and second block-paragraphs. TEMPEST knows to separate block-paragraphs with a blank line because this was specified with the definition of the block-paragraph template.

<ESC>-T-AN *ANOTHER BLOCK-PARAGRAPH*

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
M. K. Bayer
362 Memorial Dr.
Cambridge, MA  02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL  60609
@End(InsideAddress)
@Begin(Body)
Dear Mr. Whiting:
@Blankspace(1 line)
It has come to my attention that you would like us to design a
piano for your pet parakeet. While I am sure that Arthur would be a
fine musician, may I suggest that you choose an instrument which
requires less dexterity? Perhaps Arthur would enjoy taking up
whistling.  I understand that birds are naturals at it, and just think
of the money you would save on sheet music.

{the paragraph-body}
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)
```

TEMPEST (Scribe) Main: LETTER5.MSS

In Screen 12, the second paragraph-body is implemented by direct editing.

<direct editing>

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
M. K. Bayer
362 Memorial Dr.
Cambridge, MA  02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL  60609
@End(InsideAddress)
@Begin(Body)
Dear Mr. Whiting:
@Blankspace(1 line)
It has come to my attention that you would like us to design a
piano for your pet parakeet. While I am sure that Arthur would be a
fine musician, may I suggest that you choose an instrument which
requires less dexterity? Perhaps Arthur would enjoy taking up
whistling.  I understand that birds are naturals at it, and just think
of the money you would save on sheet music.

I regret that we cannot help you, but I'm afraid we do not
specialize in birds.  Nevertheless I wish you success in getting
Arthur started on his musical career.■
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)
```

TEMPEST (Scribe) Main: LETTER5.MSS

The REIMPLEMENT command is used to change the implementation of a slot. In general, when one template is reimplemented with another, the implementations of any commonly named slots are carried over to the new template. In this case, each element of the list of semi-block-paragraphs is also a paragraph-body, so no text is lost by reimplementing. However, the text is converted to the proper format for semi-block-paragraphs automatically. Note the insertion of the characters "@ @ @ @ @ ". These are actually five commands instructing Scribe to insert "significant blanks" which will cause the paragraph to be indented. (Scribe would ignore regular leading blanks, but significant blanks are treated as letters.)

This type of reimplementation can cause the buffer to become rather ugly. For example, the insertion of indentation can force a word to hang off the end of a line. It is intended that TEMPEST will be used in conjunction with text or program formatters, which are capable of smoothing over such details. Refer to the Extensions and Limitations section for a more detailed discussion of this problem.

<ESC>-T-REI ***REIMPLEMENT*** the text as a list of
semi-block-paragraphs

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
M. K. Bayer
362 Memorial Dr.
Cambridge, MA  02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL  60609
@End(InsideAddress)
@Begin(Body)
Dear Mr. Whiting:
@Blankspace(1 line)
@ @ @ @ @ It has come to my attention that you would like us to design a
piano for your pet parakeet. While I am sure that Arthur would be a
fine musician, may I suggest that you choose an instrument which
requires less dexterity? Perhaps Arthur would enjoy taking up
whistling.  I understand that birds are naturals at it, and just think
of the money you would save on sheet music.

@ @ @ @ @ I regret that we cannot help you, but I'm afraid we do not
specialize in birds.  Nevertheless I wish you success in getting
Arthur started on his musical career.
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
{the business-title}
@End(SignatureBlock)
@Begin(Notations)
{the identification-initials}
@End(Notations)


TEMPEST (Scribe) Main:  LETTER5.MSS
```

The commands in Screen 14 complete the construction of LETTER5.MSS. The first command provides actual quoted text to the IMPLEMENT command (the third usage of this command). The REMOVE command is used to delete a slot from the buffer and also from the internal representation of LETTER5 -- the slot no longer exists and cannot be referenced. Note that the identification-initials slot consisted not only of the line containing the slot name, but also the lines of text above and below that line. All three of these lines were deleted. In general, a slot may have associated text surrounding it. This is described in the Library section of the proposal.

<ESC>-T-IM *IMPLEMENT THE BUSINESS-TITLE AS* "Customer Service Director"
<ESC>-T-REM *REMOVE* the identification-initials

```
@Device(Dover)
@Make(FormalLetter)
@Style(LeftMargin 1.5in)
@Style(RightMargin 1.5in)
@Begin(Letterhead)
M. K. Bayer
362 Memorial Dr.
Cambridge, MA  02139
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL  60609
@End(InsideAddress)
@Begin(Body)
Dear Mr. Whiting:
@Blankspace(1 line)
@ @ @ @ @ It has come to my attention that you would like us to design a
piano for your pet parakeet. While I am sure that Arthur would be a
fine musician, may I suggest that you choose an instrument which
requires less dexterity? Perhaps Arthur would enjoy taking up
whistling.  I understand that birds are naturals at it, and just think
of the money you would save on sheet music.

@ @ @ @ @ I regret that we cannot help you but I'm afraid we do not
specialize in birds.  Nevertheless I wish you success in getting
Arthur started on his musical career.
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
Customer Service Director
@End(SignatureBlock)
```

TEMPEST (Scribe) Main: LETTER5.MSS

This command produces a description of the text as it currently exists in the buffer. It shows how a document was built by listing information about which slots are present and what their values are. Only the most important slots in a template are mentioned by the DESCRIBE command; these are marked when the template is defined.

Typically the description does not descend more than one level in the template hierarchy. For example, it is stated that "the from-address is my-address", but the various parts of my-address are not elaborated. It is possible and indeed may be better to expand this description to cover two levels. This will be investigated during the thesis implementation.

<ESC>-T-DE *DESCRIBE LETTER5.MSS*
<the user saves LETTER5.MSS, exits TEMPEST, and processes it with SCRIBE>

LETTER5 is a business-letter in language SCRIBE.
The from-address is my-address.
The date is CurrentDate().
The to-address is a personal-address.
The text is a list of semi-block-paragraph.

■

```
@End(Letterhead)
@Begin(DateLine)
May 1, 1984
@End(DateLine)
@Begin(InsideAddress)
Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL  60609
@End(InsideAddress)
@Begin(Body)
Dear Mr. Whiting:
@Blankspace(1 line)
@ @ @ @ @ It has come to my attention that you would like us to design a
piano for your pet parakeet. While I am sure that Arthur would be a
fine musician, may I suggest that you choose an instrument which
requires less dexterity? Perhaps Arthur would enjoy taking up
whistling.  I understand that birds are naturals at it, and just think
of the money you would save on sheet music.

@ @ @ @ @ I regret that we cannot help you but I'm afraid we do not
specialize in birds.  Nevertheless I wish you success in getting
Arthur started on his musical career.
@End(Body)
@Begin(ComplimentaryClose)
Sincerely yours,
@End(ComplimentaryClose)
@Begin(SignatureBlock)
M. K. Bayer
Customer Service Director
@End(SignatureBlock)
```

TEMPEST (Scribe) Main: LETTER5.MSS

On the following page is the output resulting from processing LETTER5.MSS through SCRIBE.

35

M. K. Bayer
362 Memorial Dr.
Cambridge, MA 02139

May 1, 1984

Mr. Clarence P. Whiting
1234 Fifth St.
Chicago, IL 60609

Dear Mr. Whiting:

It has come to my attention that you would like us to design a piano for your pet parakeet. While I am sure that Arthur would be a fine musician, may I suggest that you choose an instrument which requires less dexterity? Perhaps Arthur would enjoy taking up whistling. I understand that birds are naturals at it, and just think of the money you would save on sheet music.

I regret that we cannot help you but I'm afraid we do not specialize in birds. Nevertheless I wish you success in getting Arthur started on his musical career.

Sincerely yours,

M. K. Bayer
Customer Service Director

The remaining screens of the scenario illustrate the use of TEMPEST to create a small program in CLU [Liskov 81].

In this screen, the user begins a new file, changes the language in use to CLU, and requests that a procedure template be inserted into the buffer. An interesting feature of this template is that there are two slots with the name "proc-name." Though slots usually have unique names, it is not illegal for two or more slots to have the same name. In the case of programming language templates, this can be very useful for representing variables or procedure names. One problem that arises, however, is that there is ambiguity involved when referring to the proc-name slot. TEMPEST resolves this by defining all named references as referring to the first slot following the cursor that matches the given name. If necessary, the search for the named slot wraps around from the end to the beginning of the buffer. Thus, it may take two requests to move the cursor to the correct proc-name slot, depending on where the cursor is originally.

```
<the user begins editing new file COUNT.CLU>
<ESC>-T-U       USE LANGUAGE Clu
<ESC>-T-IN      INSERT a procedure
--------------------------------------------------------------------------------

   {the proc-name} = proc({the name of the formal}:{the type of the formal})
        returns({the return-type of the returns})
        signals({the signal-name of the signals})

   {the body}

   end {the proc-name}
```

```
TEMPEST (Clu) Main:  COUNT.CLU
--------------------------------------------------------------------------------
```

The proc-name, the name of the formal, and the type of the formal are implemented by direct editing. Note that the value of the proc-name has propagated to the proc-name slot in the last line of the procedure.

<direct editing>

```
   count = proc(source:string
        returns({the return-type of the returns})
        signals({the signal-name of the signals})

   {the body}

   end count
```

TEMPEST (Clu) Main: COUNT.CLU

In Screen 18, the user requests another formal-parameter (the name of the formal and the type of the formal from Screen 16 were actually part of a list of formal-parameters -- each formal-parameter is a template with a slot for a name and a type). Note that a comma is inserted between the two list elements. All templates that may be used in a list are defined with instructions for how to place them in a list correctly.

Note also that the new slot names do not say "of the formal", while the original formal-parameter slot names did. This is because the original was a default, and its origin was not explicitly known to the user. Since the user requested the new formal-parameter, there is no need to annotate its slot names.

<ESC>-T-AN *ANOTHER FORMAL-PARAMETER*

```
   count = proc(source:string,{the name}:{the type})
        returns({the return-type of the returns})
        signals({the signal-name of the signals})

   {the body}

   end count
```

TEMPEST (Clu) Main: COUNT.CLU

In Screen 19, the name, type, and the return-type are implemented by direct editing.

```
<direct editing>

--------------------------------------------------------------------------------

   count = proc(source:string,target:char)
        returns(int
        signals({the signal-name of the signals})

   {the body}

   end count
```

```
TEMPEST (Clu) Main:  COUNT.CLU
--------------------------------------------------------------------------------
```

In Screen 20, the user removes the signal clause, which is actually a list of signal-name templates since a procedure may signal several exceptions. This procedure will not signal any exceptions, so this clause is unnecessary.

<ESC>-T-REM ***REMOVE*** the list of signal-name

```
   count = proc(source:string,target:char)
        returns(int)


   {the body}

   end count
```

TEMPEST (Clu) Main: COUNT.CLU

This command illustrates the use of a parameterized template. The template item-counter has two slots which act as formal parameters. When the template is used in an IMPLEMENT command, values may be specified for those parameter slots. In this case, the text values "target" and "source" were given, and were substituted for their corresponding slots throughout the template.

A template may have any number of parameters, all of which are specified when the template is defined. The values specified after the keywords "of" and "and" are associated with their formals positionally, so no parameter may be skipped. It is allowable to give too few values, in which case substitution will only take place for as many values as are given. Slots which act as parameters are no different than other slots.

<ESC>-T-IM *IMPLEMENT THE BODY AS* an item-counter of
"target" and "source"

```
count = proc(source:string,target:char)
     returns(int)


{the counter}:int := 0

for {the element}:{the element-type} in
    {the iter-type}${the iter-name}(source) do
    if {the element} = target
       then {the counter} := {the counter} + 1
    end
end

return({the counter})


end count
```

TEMPEST (Clu) Main: COUNT.CLU

In Screen 22, the remaining slots are implemented by direct editing. Value propagation takes place between the various instances of the element and counter slots. Note that there was very little typing required to construct this procedure because of value propagation.

<direct editing>

```
count = proc(source:string,target:char)
     returns(int)


cntr:int := 0

for c:char in
    string$chars(source) do
    if c = target
       then cntr := cntr + 1
    end
end

return(cntr)

end count
```

TEMPEST (Clu) Main: COUNT.CLU

The Library

The template library is a user-readable, user-modifiable database of template definitions. It is structured on two levels: templates are first grouped according to their associated language; within a language group, templates are arranged in a tree according to their type. Aside from this explicit structuring, there is also an implicit structure imposed on the library in that many templates are merely fragments intended as slot fillers for other templates. For example, address templates are not particularly useful by themselves but are implicitly associated with templates for business letters and other correspondence. To make full use of TEMPEST requires a good knowledge of these implicit associations.

Defining Templates

Users may modify the library by defining new templates or changing the definition of existing ones. A template is defined by filling the slots of a unique template known as the "template-definer" (shown below). The COMPILE command is run on the completed template-definer to enter the definition into the data base.

```
NAME --
{the name}

TYPE --
{the type}

LANGUAGE --
{the language}

TEXT --
{the text}

DEFAULTS --
{the name of the default} = {the kind of the default} | {the value of the default}

CONSTRAINTS --
{the constraints}

PARAMETERS --
{the parameters}
```

Figure 1: The Template-Definer

Following is the completed template-definer for the CLU procedure template of scenario Screen 16.

```
NAME --
procedure

TYPE --
program-module < root

LANGUAGE --
Clu

TEXT --
<{the proc-name:TEXT}> = proc(<{the formals:LIST OF FORMAL-PARM}>)
                         returns(<{the returns:LIST OF RET-TYPE}>)
                         signals(<{the signals:LIST OF SIG-NAME}>)

<{the body:LIST OF STATEMENT}>

end <{the proc-name:TEXT}>

DEFAULTS --
formals =  Implicit | formal-parameter
returns =  Implicit | return-type
signals =  Implicit | signal-name

CONSTRAINTS --


PARAMETERS --
```

Figure 2: The Definition of the CLU Template "Procedure"

The NAME and LANGUAGE sections of the template are self explanatory. The remaining sections, however, deserve considerable explanation, and are discussed below.

TYPE --

The template is of type "PROGRAM-MODULE." The remainder of the line indicates that this template should be placed immediately subordinate to the root of the CLU type subtree.

TEXT --

This section defines how the text and slots of the templates will appear. There are three special notations used here: the angle brackets ("<...>") delimit slots; the braces ("{...}") delimit slot names; and the colon separates the slot name from the slot type. Notice that if a slot may contain a list, this is specified with its type.

In the scenario, braces were used to delimit slots, and it appeared that a slot consisted only of the text between the braces. This is not necessarily the case. There may be text surrounding the braces which is actually part of the slot. Recall Screen 14 of the scenario in which three lines of text were deleted when the identification-initials slot was removed -- the lines before and after the slot name were the slot's associated text. In the slot definition, any such text would be placed inside the angle brackets to indicate that it belongs with the slot. In this example, no slots have any associated text, so only the slot name appears between the angle brackets.

The text inside the braces is the slot name, followed by a colon, followed by the slot type. The slot name is displayed between the braces when the template is used for editing, as was the case throughout the scenario. The only exception to this is when the slot is given a default, in which case the slot names of the default template are displayed.

DEFAULTS --

Each line of the default section has three slots that constitute a subtemplate. To express more than one default, a new subtemplate can be obtained with the ANOTHER command. The kind of the default must be "Implicit" or "Explicit." The value of the default may be text, a call to a built-in function, or a template. In this example, all the defaults are templates.

CONSTRAINTS --

Constraints are expressed through calls to built-in functions. Built-in functions can also be used to produce slot values. These functions usually return a reference to a string, a slot, or a template, and can have any number of parameters. Their exact usage and utility has not yet been determined, but will be investigated during the thesis implementation. The complexity of expressible constraints is determined by the power of the built-in functions, which will be limited but expandable.

In this example, no constraints need be expressed (the slots in this template having the same name will automatically have the same value). The constraints slot has been cleared with the CLEAR command. (Like REMOVE, CLEAR deletes the slot from the text, but does not remove it from the internal representation.)

PARAMETERS --

This template has no parameters, so the parameters slot has been cleared with the CLEAR command. If any of the slots had been intended as parameters, their names would have been placed in a list.

Examining the Library

Ideally, the library should have a query facility with which a user could, for example, find all templates of a particular type, view some subtree of types, or see which template names contain a certain keyword (much like the "Apropos" command in EMACS [Stallman 81]). However, since this implementation of TEMPEST is intended only as a prototype, it will not require an elaborate inquiry facility. As a compromise due to time constraints, a BROWSE command will be implemented which will produce a list (indented to reflect the hierarchy) of a given branch of the template hierarchy.

The Implementation

This section outlines the major steps in the implementation of TEMPEST and gives an approximate timetable for their completion. The code for the thesis will be written in the C programming language and will be developed and run on an IBM personal computer. As was stated in the Introduction, TEMPEST will be an extension to the MINCE text editor.

Timetable

The implementation can be broken down into the following stages, each of which is given an approximate completion time.

* **Command Interface -- Preliminary** 1 week
  The preliminary command interface will provide the ability to dispatch on certain keystrokes. There will be no argument parsing at this stage; instead, each command will prompt for and process its own arguments. This is a gateway facility so functions can be added and tested.

* **Internal Interfacing** 3 weeks
  This will involve designing a number of data structures for representing templates. The data structures will vary during the course of the implementation, but finding a way to represent templates in the buffer so they interact well with the editor is a one time task.

* **Commands** 7 weeks
  Commands will be added once the necessary framework is in place. The INSERT, IMPLEMENT, and COMPILE commands will be first so it will be possible to have a template in the buffer and work with it. Not all the commands will necessarily be implemented at this stage.

* **Constraints** 3 weeks
  This stage will implement a general facility for "hooking" constraint checking to the appropriate commands, and will also implement value constraints.

* **Built-in Functions** 1 week
  Built-in functions will be written as C functions and called internally by implementation and constraint routines.

* **Command Interface -- Final** 3 weeks
  The final interface will have a limited parser for command arguments, supporting a somewhat natural command language as was used in the scenario.

The total projected time for these steps of the implementation is 18 weeks, which comes to the end of August if implementation starts in early May. This is an intentionally optimistic schedule, timed to coincide with the end of the second graduate co-op assignment. It is expected that implementation will continue beyond the end of August, particularly if any of the possible extensions are attempted.

Implementation Issues

A significant problem in the implementation of TEMPEST will be in integrating templates into the text. An ideal method would be to imbed templates directly in the text buffer and delimit them with some sort of "invisible" buffer mark which would not print and which would not disturb the operation of the editor. A proposed solution to this problem is to use some or all of the Meta characters (i.e., those characters whose decimal ASCII value is greater than 127) as marks and to modify the printing routines to handle these characters specially. The goal is to avoid modifying the incremental redisplay algorithm.

The "<ESC>-T" command interface should be relatively simple since MINCE command routines are invoked from an easily modifiable dispatch table. Somewhat more difficult will be the command line parser, but this will be made simpler by ensuring that the command syntax is context free. An issue to be resolved is the treatment of command arguments; MINCE makes these globally available to its action routines, but it may be preferable to pass them directly to TEMPEST's routines.

It is expected that handling the effect of direct edits on the structure of the text will be the most difficult aspect of the implementation. Particular attention must be paid to the effect of deletions that involve only part of the text of a slot, especially if the buffer marking scheme is used.

The constraint facility will also be difficult to implement. One question is how to deal with the implicit usage of the IMPLEMENT command -- if a slot is changed by direct editing, this must be recognized so constraint enforcement can take place.

Related to constraints will be the built-in functions, which will have to operate on abstract data types such as slots and templates. There are semantic issues to be decided here such as the distinction between an object and its name. For example, does "the return-address" refer to the name of the return-address slot or the slot itself? It will be important to handle this question correctly and consistently in order to make the built-in functions as general and powerful as possible.

Related Work

The fundamental idea behind TEMPEST, editing with templates, is drawn from the Knowledge Based Editor (KBE), a tool for creating and editing programs [Waters 82]. TEMPEST's templates and slots correspond to the KBE's cliches and roles. A major part of the KBE for which there is no analog in TEMPEST, however, is the idea of a *plan*. A plan is an abstract representation of a program in which data flow and control flow knowledge is explicitly represented. KBE commands, which seemingly operate on the text of a program, actually operate on its plan. The manipulation of plans rather than text is what gives the KBE its power.

TEMPEST does not use plans to represent the text in the editing buffer. Indeed, one of TEMPEST's constraints is that it be text based -- an admitted sacrifice of depth in hopes of achieving greater breadth. TEMPEST is thus similar to the KBE in its use of templates, but differs in the type of functionality it can offer and the way it must achieve that functionality.

Because of its hierarchical treatment of text, TEMPEST also resembles structure-oriented text editors, for example, ED3 [Stromfors 81] and the Document Editor [Walker 81]. These structure editors operate in a top-down fashion in which the user issues commands to arrange his text according to a desired hierarchy. The commands are usually tree oriented, involving insertion, deletion, and reordering of tree nodes.

A drawback of the top-down approach is that it forces the user to proceed in a particular order. Reshuffling the tree nodes is thus difficult. Using ED3, for example, the user would have to execute several low level tree restructuring commands in order to do this, since ED3 has no knowledge of structure. TEMPEST, on the other hand, will have a template reflecting the ultimate structure of the user's document, and with one command can reorder the text nodes correctly, thereby overcoming the restriction imposed by proceeding top-down. The addition of even a modest knowledge of structure can provide greater power and ease of use than would be found with a typical structure editor.

Limitations and Extensions

Limitations

An admitted limitation of TEMPEST is that it is text based. This prevents it from exhibiting significant power based on data flow or control flow knowledge. It is expected that this limitation will be compensated for by a gain in generality.

The most notable limitation is in the way TEMPEST manages the screen. When a slot is filled, TEMPEST inserts the template text into the buffer exactly as it appears in the template definition, without exercising control over the format. This can work well in many cases, but it can also result in haphazardly formed text, particularly in the case of programs, which are difficult to read when not properly indented.

Ideally TEMPEST should have a method for recognizing and correcting format flaws such as too-long lines or extraneous white space. One solution might be to include with a template a description of its general format as well as that of the text surrounding it. This is disadvantageous, however, in that it places the burden of description on the designer of new templates.

Extensions

The complex commands are, with the exception of the REIMPLEMENT command, an extension to TEMPEST though their implementation is planned. In general, they support bottom-up template construction and, as such, are separate from the other commands.

The TAG and REORDER commands are intended for numbering and reordering pieces of text which may or may not include slots. This is analogous to numbering paragraphs in the margin of a rough draft.

The COPY command is used for obtaining a reference to a slot where one did not exist originally. For example, in the text of a business letter one might wish to refer to the sender of the letter, at which point the COPY command could be used to insert a copy of the sender slot. This creates a new sender slot which implicitly has the same value as all other slots of the same name. This could be particularly useful when referring to policy numbers or variable names, for example.

The GATHER command is used for placing the disjoint parts of a template into the correct order. This allows the user to work bottom-up. For example, rather than inserting a report template into the buffer, the user could insert first a section template, then an appendix template, then another section template, etc., and finally request TEMPEST to GATHER them into a report. TEMPEST would then try to fit the templates in the buffer into a report template. This operation might be useful for filling a template whose constraints could not be properly processed unless all its slots were filled at once.

The final complex command, MERGE, is similar to the REIMPLEMENT command in that it attempts to combine two templates. The motivation behind such a command is to make it relatively easy to change from one structure to another rather than forcing the user to explicitly break down and rebuild the hierarchy. The exact behavior of the MERGE command will be determined during the thesis implementation.

The most difficult type of extension to TEMPEST would involve text recognition, thereby imbedding

some "intelligence" in TEMPEST. For example, if a user inserts a blank line in a paragraph slot, it would be helpful for TEMPEST to understand that the paragraph has just been split. Currently TEMPEST would not know this and would incorrectly include the blank line and the succeeding text as part of the paragraph. It is hoped that time will permit some exploration of this extension since it is consistent with TEMPEST's motivation of aiding the user through structural knowledge.

TEMPEST might also be extended by exploring new domains. One possible domain is graphics. It is conceivable that TEMPEST could manipulate templates that contain graphics characters. Such templates could be used for improving the appearance of graphs or tables. Interestingly, TEMPEST would need greater format control to work with graphics, so this extension might force the solution of the screen management problem.

Another extension would be to add more special forms to TEMPEST in addition to "list." For example, one could imagine a form for "m by n table." If these more complex forms were combined with graphics templates, it would be possible to construct a large number of complex documents.

References

[Liskov 81]
Liskov, Barbara et. al.
CLU Reference Manual.
MIT Laboratory for Computer Science, 1981.
This manual appeared as part of the Springer-Verlag Lecture Notes in Computer Science

[Reid 80]
Reid, Brian K. and Walker, Janet H.
SCRIBE Introductory User's Manual.
Unilogic, Ltd., 1980.

[Stallman 81]
Stallman, Richard M.
EMACS Manual for TWENEX Users.
MIT Artificial Intelligence Laboratory, 1981.

[Stromfors 81]
Stromfors, O. and Jonesjo, L.
The Implementation and Experiences of a Structure-Oriented Text Editor.
In *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, pages 22-27.
ACM, 1981.

[Walker 81]
Walker, Janet H.
The Document Editor: A Support Environment for Preparing Technical Documents.
In *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, pages 44-50.
ACM, 1981.

[Waters 82]
Waters, Richard C.
The Programmer's Apprentice: Knowledge Based Program Editing.
IEEE SE-8(1), January, 1982.