ABSTRACT:

PUB is an advanced text justifier and page formatter intended primarily for use by programmers. It can automatically number pages, sections, figures, footnotes, etc. and can print their numbers in Roman numerals as well as in digit or letter form. It can generate cross references, tables of contents, and indexes. Page layout is flexible, and allows multiple column output. Line formatting includes tabs, underlining, superscripts, subscripts, centering, and justification. Macros programmed in a SAIL-like string-processing language can generate text to be printed in the document. The output of the compiler is a file which can be printed on the terminal, on the line printer, or on microfilm.

ACKNOWLEDGMENTS:

Les Earnest created the concept of the Document Compiler and specified most of its capabilities.

Dan Swinehart provided invaluable advice and aid throughout the development of PUB.

Russ Taylor programmed the FR-88 preprocessor.

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract SD 183.
# TABLE OF CONTENTS

## SECTION

### 1 INTRODUCTION

| 1.1 | PURPOSE | \( \ldots \) | 1 |
| 1.2 | CAPABILITIES | \( \ldots \) | 1 |
| 1.3 | OPERATION | \( \ldots \) | 2 |
| 1.4 | COMPILER OUTPUT | \( \ldots \) | 4 |
| 1.5 | READING THIS MANUAL | \( \ldots \) | 5 |
| 1.6 | TUTORIAL FOR BEGINNERS | \( \ldots \) | 5 |
| 1.7 | SAMPLE COMPILATION | \( \ldots \) | 9 |

### 2 TEXT CONVENTIONS

| 2.1 | ILLEGAL CHARACTERS | \( \ldots \) | 13 |
| 2.2 | COMMAND AND TEXT LINES | \( \ldots \) | 13 |
| 2.3 | PARAGRAPHING | \( \ldots \) | 14 |
| 2.4 | FILL MODE | \( \ldots \) | 14 |
| 2.5 | JUSTIFICATION | \( \ldots \) | 14 |
| 2.6 | BREAKING AT BLANK LINES | \( \ldots \) | 15 |
| 2.7 | BREAKING AT TABS | \( \ldots \) | 15 |
| 2.8 | BREAKING AT CARRIAGE-RETURNS | \( \ldots \) | 15 |
| 2.9 | NOFILL MODE | \( \ldots \) | 15 |
| 2.10 | HORIZONTAL SPACE COMPACTION | \( \ldots \) | 16 |
| 2.11 | VERTICAL GROUPING | \( \ldots \) | 17 |
| 2.12 | MODE AND SWITCH SETTING COMMANDS | \( \ldots \) | 17 |
| 2.13 | WORD BREAKS | \( \ldots \) | 18 |
| 2.14 | CONTROL FUNCTIONS | \( \ldots \) | 18 |
| 2.15 | CONTROL CHARACTER ACTIVATION | \( \ldots \) | 21 |
| 2.16 | COMPUTED TEXT | \( \ldots \) | 23 |

### 3 MACROS

| 3.1 | CALLING A MACRO FROM A COMMAND LINE | \( \ldots \) | 25 |
| 3.2 | CALLING A MACRO FROM A TEXT LINE | \( \ldots \) | 25 |
| 3.3 | TEMPLATES | \( \ldots \) | 26 |
| 3.4 | MACRO DECLARATION | \( \ldots \) | 27 |
| 3.5 | MACRO CALLS | \( \ldots \) | 27 |
| 3.6 | OMISSIONS | \( \ldots \) | 28 |
| 3.7 | RECURSIVE MACROS | \( \ldots \) | 29 |
# TABLE OF CONTENTS

## 4 EXPRESSIONS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>DATA TYPE</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>VARIABLES</td>
<td>30</td>
</tr>
<tr>
<td>4.3</td>
<td>AUTOMATICALLY DECLARED VARIABLES</td>
<td>31</td>
</tr>
<tr>
<td>4.4</td>
<td>SIMPLE EXPRESSIONS</td>
<td>32</td>
</tr>
<tr>
<td>4.5</td>
<td>GENERAL EXPRESSIONS</td>
<td>36</td>
</tr>
<tr>
<td>4.6</td>
<td>CONSTANTS</td>
<td>34</td>
</tr>
</tbody>
</table>

## 5 STATEMENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>STATEMENT TYPES</td>
<td>35</td>
</tr>
<tr>
<td>5.2</td>
<td>BLOCK</td>
<td>35</td>
</tr>
<tr>
<td>5.3</td>
<td>COMPOUND STATEMENT or CLUMP</td>
<td>35</td>
</tr>
<tr>
<td>5.4</td>
<td>ONE PARAGRAPH SCOPE</td>
<td>36</td>
</tr>
</tbody>
</table>

## 6 DECLARATIONS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>SCOPE</td>
<td>37</td>
</tr>
<tr>
<td>6.2</td>
<td>MODE DECLARATIONS</td>
<td>37</td>
</tr>
<tr>
<td>6.3</td>
<td>TABS DECLARATION</td>
<td>37</td>
</tr>
<tr>
<td>6.4</td>
<td>INDENT DECLARATION</td>
<td>38</td>
</tr>
<tr>
<td>6.5</td>
<td>PREFACE DECLARATION</td>
<td>39</td>
</tr>
<tr>
<td>6.6</td>
<td>DOUBLE SPACING</td>
<td>39</td>
</tr>
<tr>
<td>6.7</td>
<td>AREA DECLARATION</td>
<td>39</td>
</tr>
<tr>
<td>6.8</td>
<td>MARGIN CONTROL</td>
<td>40</td>
</tr>
<tr>
<td>6.9</td>
<td>PLACE DECLARATION</td>
<td>42</td>
</tr>
<tr>
<td>6.10</td>
<td>TEMPORARY AREAS</td>
<td>42</td>
</tr>
<tr>
<td>6.11</td>
<td>COUNTER DECLARATION</td>
<td>43</td>
</tr>
<tr>
<td>6.12</td>
<td>DOUBLE SPACING</td>
<td>55</td>
</tr>
</tbody>
</table>

## 7 IMPERATIVES

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>ASSIGNMENT STATEMENT</td>
<td>45</td>
</tr>
<tr>
<td>7.2</td>
<td>CONDITIONAL STATEMENT</td>
<td>45</td>
</tr>
<tr>
<td>7.3</td>
<td>NEXT COUNTER VALUE STATEMENT</td>
<td>45</td>
</tr>
<tr>
<td>7.4</td>
<td>NEXT PAGE STATEMENT</td>
<td>47</td>
</tr>
<tr>
<td>7.5</td>
<td>READINGS AND FOOTINGS</td>
<td>48</td>
</tr>
<tr>
<td>7.6</td>
<td>SECTIONING</td>
<td>48</td>
</tr>
<tr>
<td>7.7</td>
<td>COMMAND CHARACTER STATEMENT</td>
<td>49</td>
</tr>
<tr>
<td>7.8</td>
<td>PORTION DEMARCATION</td>
<td>49</td>
</tr>
<tr>
<td>7.9</td>
<td>SEND STATEMENT</td>
<td>50</td>
</tr>
<tr>
<td>7.10</td>
<td>RECEIVE STATEMENT</td>
<td>52</td>
</tr>
<tr>
<td>7.11</td>
<td>REQUIRE STATEMENT</td>
<td>53</td>
</tr>
<tr>
<td>7.12</td>
<td>SKIP STATEMENTS</td>
<td>53</td>
</tr>
<tr>
<td>7.13</td>
<td>PAGE FRAME STATEMENT</td>
<td>55</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.14 BREAK STATEMENT</td>
<td>55</td>
</tr>
<tr>
<td>7.15 CONTINUE STATEMENT</td>
<td>55</td>
</tr>
<tr>
<td>7.16 DEVICE STATEMENT</td>
<td>55</td>
</tr>
<tr>
<td>7.17 UNIMPLEMENTED STATEMENTS</td>
<td>56</td>
</tr>
<tr>
<td>8 COMMENTS</td>
<td></td>
</tr>
<tr>
<td>9 LABELS AND CROSS-REFERENCES</td>
<td></td>
</tr>
<tr>
<td>9.1 CROSS-REFERENCES</td>
<td>59</td>
</tr>
<tr>
<td>9.2 LABELS</td>
<td>60</td>
</tr>
<tr>
<td>10 RESPONSES</td>
<td></td>
</tr>
<tr>
<td>10.1 TEXT RESPONSES</td>
<td>63</td>
</tr>
<tr>
<td>10.2 TRANSITION RESPONSES</td>
<td>64</td>
</tr>
<tr>
<td>11 FOOTNOTES</td>
<td></td>
</tr>
</tbody>
</table>

## APPENDICES

### THOROUGHLY EXPLAINED EXAMPLES

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 SECTIONING MACROS</td>
<td>68</td>
</tr>
<tr>
<td>A.2 SAMPLE TABLE OF CONTENTS</td>
<td>69</td>
</tr>
<tr>
<td>A.3 ONE-LEVEL INDEXES</td>
<td>70</td>
</tr>
<tr>
<td>A.4 TWO-LEVEL INDEXES</td>
<td>72</td>
</tr>
<tr>
<td>A.5 KWIC INDEXES</td>
<td>74</td>
</tr>
</tbody>
</table>

SUBJECT AND COMMAND INDEX ... I
SECTION 1
INTRODUCTION

1.1 PURPOSE

PUB is a compiler which translates a manuscript into a document.

A "manuscript" is an SOS file containing the text of a publication interspersed with control characters and commands. The function of the compiler is to arrange this text on formatted pages under the direction of the control characters and commands.

Output is written on a ".DOC" file known as the "document". TYPE, SPOOL or PRINT it. Under certain conditions, the document can be edited with TECO (see Section 7.16), but it is wiser to make corrections in the manuscript and then rerun the compiler.

MANUSCRIPT ←—— DOCUMENT
COMPILER

1.2 CAPABILITIES

PUB provides the usual capabilities of a simple text-justifier, including:

>Page numbering
>Optional justification to the right margin
>Centering of titles
>Headings and footings
>Control of Spacing and Indentation
>Underlining

In addition, it features advanced documentation capabilities, such as:

1.1 - 1.2
The command language is a dialect of SAIL Algol, providing the user with:

> Block structure
> Conditional command execution
> Conditional text inclusion
> Arithmetic and String calculations
> Embedded Source Files

1.3 OPERATION

The compiler is "two-pass". The first pass interprets the manuscript and outputs several intermediate files with extensions ".PUG", ".PUZ", and ".PUI". The second pass is a separate program automatically called which makes one pass through the ".PUI" files and produces the document. The ".PUG" and ".PUZ" files are automatically deleted at the end of Pass One. The ".PUI" files are read by Pass Two, then deleted (unless deletion is waived by a switch setting -- see below).

The first pass typically requires 48K of core and runs at the rate of 15 pages per minute. The second pass needs 18K and runs at 50 pages per minute. If the system should crash during Pass Two, and you wish to restart without rerunning Pass One, R PUB2. "R PUB2" also gives you an opportunity to specify a different output device than originally requested.

You may run the document compiler with or without RPG. A manuscript file which is "PUB" or a document file extension which is ".DOC" need not be mentioned. Examples:
INTRODUCTION

command typed

<table>
<thead>
<tr>
<th>command typed</th>
<th>input file</th>
<th>output file</th>
<th>switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>.COM FOO</td>
<td>FOO.PUB</td>
<td>FOO.DOC</td>
<td></td>
</tr>
<tr>
<td>.COM FOO-BAZ</td>
<td>BAZ.PUB</td>
<td>FOO.DOC</td>
<td></td>
</tr>
<tr>
<td>.COM /PUB FOO.0OF+BAZ.ZAB</td>
<td>BAZ.ZAB</td>
<td>FOO.0OF</td>
<td></td>
</tr>
<tr>
<td>.COM /PUB FOO.0OF</td>
<td>FOO.0OF</td>
<td>FOO.DOC</td>
<td></td>
</tr>
<tr>
<td>.COM FOO(DN)</td>
<td>FOO.PUB</td>
<td>FOO.DOC</td>
<td>/D /N</td>
</tr>
</tbody>
</table>

Do not specify more than one input file nor more than one output file. Multiple input files are handled using the REQUIRE statement (see Section 7.11).

WARNING (RPG users): If in addition to file X.PUB you have a file X.SAI (or X.FAI, etc.) and if you type "COM X", RPG will do a SAIL (or FAIL) compilation.

SOLUTION: Type "COM X.PUB".

FURTHER WARNING (RPG users): If you have a file X.REL which is newer than X.PUB, then RPG won't do any compilation. SOLUTION: Type "COM /COM X.PUB".

Presently available switches are:

nS  Change intra-paragraph line spread: 1S=Single Space, 2S=Double Space, etc
B   Big Document -- allocates hashed space for 3000 identifiers instead of 1000
H   Huge Document -- allocates maximum symbol table space (8191)
D   DEBUG -- prints approximate source file line numbers alongside the output
L   Line Printer output planned (Default)
M   Microfilm output planned (produces FRBB command file -- see Section 7.16)
T   Terminal output planned (may produce different output than /L)
Y   Yes! Automatically delete ".PUI" files after Pass 2 (Default)
N   No! Don't delete them (thus allowing Pass 2 to be re-run with a different DEVICE)
A   Ask me whether to delete them when it's time.
Z   Compiler Debug: see compiler (PUB.SAI[2, TES])
1.4 COMPILER OUTPUT

During Pass one, whenever PUB reads a "Page Mark" or Form Feed, it types out the manuscript page number. During Pass Two, whenever PUB writes a Form Feed, it types out a count of the number of output pages so far.

During Pass One, PUB also types out error messages. After each message, an arrow (->) or a question mark (?) is typed out. You may then respond as in SAIL. If you are not familiar with SAIL, here is a summary of responses:

<table>
<thead>
<tr>
<th>Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>Continue compiling after this error.</td>
</tr>
<tr>
<td>LF</td>
<td>Continue compiling, and after subsequent errors, continue automatically.</td>
</tr>
</tbody>
</table>

**NOTE:** The above two responses are not valid after "?"

<table>
<thead>
<tr>
<th>Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E&lt;file&gt;</td>
<td>Edit file &lt;file&gt;.</td>
</tr>
<tr>
<td>E&lt;CR&gt;</td>
<td>(RPG only) Edit the manuscript file.</td>
</tr>
<tr>
<td>S&lt;CR&gt;</td>
<td>Start Over.</td>
</tr>
</tbody>
</table>

In DEBUG mode (the /D switch), some error messages are also output in the right margin of the document.

Whenever PUB prints a line number in an error message or in the right margin of the document, it is of the form:

```
<file id><line no.>/<page no.>[<macro line no.>/<macro page no.>]
```

- If the line is from the manuscript, then <file id> is empty. If it is from another file (such as one of those "PUG" files mentioned above), then <file id> is the first three letters of the first name of that file.

- If the file is in SOS format, then the <line no.> is always five digits, including leading zeroes if necessary. If it is from a TECO or TVEDIT file, then PUB generates line numbers 1, 2, 3, etc. without leading zeroes. If it is from a "PUG" file, then the <line no.> is the line number/page number of a relevant line in your manuscript or in some other source file.

- If a macro was being expanded, the line number and
page number of its definition are noted within square brackets. The file that contained the definition is not noted, so if can't find it in your manuscript, it probably came from PUBSTD.DFS, the PUB standard macro file, or from one of your REQUIRE files (see Section 7.11).

Examples:

02300/10 Manuscript file, page 10, line 2300 (SOS)
IND02300/10/1 INDEX.PUG file, page 1, comes from Manuscript 2300/10
PUB25/1 PUB Standard Macro File, Page 1, Line 25 (TVEDIT)

1.5 READING THIS MANUAL

This manual is ordered such that as soon as you have learned enough of the system to solve your particular problem, you can skip the rest of the manual. Read at least through Section 2.3 before making any such assumption.

In describing the syntax of PUB commands, the following meta-linguistic symbolism is employed:

<...> These brackets delimit the name of a syntactic entity
[...] These delimit optional components of the command
...|... This character separates alternatives for the same command component

Cross-references in this document are always to subsections. Subsection numbers are printed at the bottom of each page.

1.6 TUTORIAL FOR BEGINNERS

PUB is a string processing language. Characters can be formed into words, words into lines, lines into paragraphs, paragraphs into columns, columns into "areas" (such as the title areas at the top and bottom of the page), areas into pages, pages into sections,
sections into "portions" (such as the Table of Contents or the Appendices), and portions into a document. The methods of building larger units from smaller ones are flexible and under control of the programmer.

The most trivial use of PUB is to create a document that is exactly identical to the manuscript, adding only headings and footings. PUB assumes that the format of each output page is as follows. The first three lines are a Heading area of which only the first is usually used. The next 48 lines (lines 4 to 51) are the Text area. Line 52 is blank and line 53 is a Footing area. The Footing area is for page numbers and other reference information, not for footnotes. (Footnotes are placed inside the Text area, towards its bottom).

The width of each page is assumed to be 69 characters. It is assumed that there is only one column of text output. Therefore, the longest an output line can be is 69 characters.

The number of lines in each area, the number of columns of text, and the number of characters in each column may be changed by declaration. See Section 6.7 and Section 7.13 for details. If all you want to do is widen the page, e.g., to 75 characters, then make the following line the first of the manuscript:

```
.PAGE FRAME 53 HIGH 75 WIDE
```

The "Dot" in column one indicates that this line is a Command Line for PUB to obey and should not be printed in the document. If you would rather a different character in column one served for this purpose, see Section 2.2.

PUB assumes that you want no headings and footings. It will leave the heading and footing areas blank unless you specify otherwise. The simplest way to specify headings and footings is with the standard macros "EVERY HEADING" and "EVERY FOOTING". These macros can specify a title for the left edge, center, and right edge of the heading area and the footing area. For example, the macro call:

```
.EVERY HEADING(PRELIMINARY REPORT, PARACYBERNETIC PHENOMENA, 1972)
```

1.6
will print on the top line of every page the three titles shown.

To make the top line of every page display the title "PRELIMINARY REPORT" on the left and the current date on the right, use:

.EVERY HEADING(PRELIMINARY REPORT, ,(DATE))

The consecutive commas indicate that the center is empty. The curly brackets around the word DATE specify that DATE is not a title to be printed but rather a variable to be evaluated.

To make the number of each page appear bottom center, use:

.EVERY FOOTING((PAGE), )

To learn about fancier headings and footings, you will have to read most of this document, especially Section 7.5 and Section 10.2.

PUB will number your pages 1, 2, 3, etc. unless you declare otherwise. To learn how, see Section 6.11. A new page of output will be started for one of two reasons: the previous page is full, or a page-changing command is executed. The most common page-changing command is:

.NEXT PAGE

This always starts a new page, even if one was just started and nothing is on it yet. To start a new page only if there is something on the current page, use:

.SKIP TO COLUMN 1

To leave a few blank lines in the document, use:

.GROUP SKIP 18

The SKIP command is described in Section 7.12.

PUB processes text a paragraph at a time. The usual way to indicate the end of a paragraph in your manuscript is by a blank line (that is, just a CR and LF). Other ways are discussed elsewhere in this manual, beginning at Section 2.3.

Usually, PUB packs a paragraph as tightly as possible by filling up each output line with words before 1.6
beginning a new line. This is called "filling". Then 
PUB inserts extra spaces between words to eliminate a 
ragged right edge from the paragraph. This is called 
"justifying".

PUB will not indent the first line of paragraphs 
unless an indent command appears in the manuscript, 
e.g.:

.INDENT 6

To learn more about indentation and related matters, 
see Section 6.4 through Section 6.7.

PUB assumes you would like your text lines filled out 
and justified. You can selectively exempt blocks of 
your manuscript from this fate. Each such block must 
be delimited by "\BEGIN" and "\END". After "\BEGIN", 
 specify a formatting mode to persist for the duration 
of the block. Examples:

Fill, but don't Justify  Don't even fill  Copy verbatim
\BEGIN NOJUST  \BEGIN NOFILL  \BEGIN VERBATIM
\END  \END  \END

In "NOJUST" mode, lines will be filled but not 
justified. In "NOFILL" and "VERBATIM" modes, they 
will not even be filled. The difference between 
NOFILL and VERBATIM is that VERBATIM is both faster 
and dumber. It is faster because it copies the lines 
of the manuscript to the document without looking at 
them, except to see if there is a dot in column one. 
It is dumber because it does not recognize text 
control characters that PUB normally recognizes, and 
it does not reformat in any way. Thus, VERBATIM is 
used to copy blocks of manuscript that are already 
formatted perfectly.

Filling allows the manuscript to be quite ragged, a 
great convenience in editing. Short lines are 
lengthened and long lines are shortened. Be careful 
in NOFILL or VERBATIM mode that no manuscript 
line is longer than 69 characters (or whatever limit you have 
declared), or some of it will be lost and an error 
message given.
1.7 SAMPLE COMPILATION

You now know enough to use PUB for simple purposes. The next three pages show a sample manuscript and the document that PUB produced from it.
It has been observed that the Sigma 3 in Horsetown, Mass. and the
CDC 6600 in Liverless, Cal. tend to have parity errors at the same
time. When records were compared by
Miss Minnie Messer, Director of the Horsetown Chamber of Commerce
Computation Facility,
and Mr. Solomon Crunch of Liverless Hospital's Organ-Transplant
Inventory Project, it was shown that
the correlation of parity error occurrences was 0.8, with a
probability of random coincidence of <.0000081.

Miss Messer and Mr. Crunch revealed these discoveries at the
Universal Users Union meeting in Cranchville, Tenn. after they
arrived two hours late for Mr. Crunch's scheduled talk there.
They said that in the excitement of discovery
the meeting slipped their minds.

This report has motivated this author to undertake
a wider survey to determine if similar
phenomena have occurred elsewhere. The author has
solicited Miss Messer's assistance in this survey,
but without the cooperation of the entire computing community,
it is unlikely that sufficient
data
can be collected. Therefore, we request that interested
parties tabulate the exact times of occurrence of parity
errors on their computer during the 7 day period
1200 April 18 to 1200 April 25 and send it to:
Paracybernetic Society
Boise Institute of Technology
Boise, Idaho

Results of the study will be presented at the next UUU meeting
in December.
It has been observed that the Sigma 3 in Horsetown, Mass. and the CDC 6600 in Liverless, Cal. tend to have parity errors at the same time. When records were compared by Miss Minnie Messer, Director of the Horsetown Chamber of Commerce Computation Facility, and Mr. Solomon Crunch of Liverless Hospital's Organ-Transplant Inventory Project, it was shown that the correlation of parity error occurrences was 0.8, with a probability of random coincidence of \(<0.0000001\).

Miss Messer and Mr. Crunch revealed these discoveries at the Universal Users Union meeting in Cranchville, Tenn. after they arrived two hours late for Mr. Crunch's scheduled talk there. They said that in the excitement of discovery the meeting slipped their minds.

This report has motivated this author to undertake a wider survey to determine if similar phenomena have occurred elsewhere. The author has solicited Miss Messer's assistance in this survey, but without the cooperation of the entire computing community, it is unlikely that sufficient data can be collected. Therefore, we request that interested parties tabulate the exact times of occurrence of parity errors on their computer during the 7 day period 1200 April 18 to 1200 April 25 and send it to:

Paracybernetic Society
c/o Dan Mation
Boise Institute of Technology
Boise, Idaho

Results of the study will be presented at the next UUU meeting in December.
SECTION 2
TEXT CONVENTIONS

2.1 ILLEGAL CHARACTERS

If the following characters occur in the manuscript, they may cause problems:

- '177 Rubout
- '175 Altmode
- '13 Vertical Tab

These characters are ignored in the manuscript:

- '0 Null
- '14 Form Feed (except after LF, where it is a page mark)

A manuscript prepared using SOS or TVEDIT is legitimate if it has no vertical tabs. If it is prepared using TECO, be sure as well that every line ends with CR LF.

Tabs are expanded to the appropriate number of spaces, as in the line editor.

2.2 COMMAND AND TEXT LINES

Every line with a dot (".") in column 1 is processed as a command line. Every other line is processed as a text line.

If you would prefer a different character to introduce command lines, e.g., $, use the command:

COMMAND CHARACTER "$" ;

2.1 - 2.2
2.3 PARAGRAPHING

The text is processed in logical units called "paragraphs". A paragraph is the accumulation of words from one or more text lines, terminated by the occurrence of a paragraph break.

A paragraph break can be caused by the command BREAK as well as by several other commands. In addition, a paragraph break can be caused by signals in the text.

Paragraphing conventions can be varied by the use of various mode and switch settings.

2.4 FILL MODE

In FILL mode, the compiler puts as many words as can fit on each output line before beginning a new line -- each line is "filled" with words.

For convenience of exposition, names have been given to the parts of the output paragraph in FILL mode. The first line is called the crown, and the rest of the lines are called the vest. It is common to indent the crown and not the vest, or vice versa, or to indent them different amounts. The last line of the vest is called the hem. While the other lines of the paragraph can be subject to justification, the hem line is never justified.

2.5 JUSTIFICATION

In FILL mode, whether or not output lines are justified to the right margin is determined by the setting of the "ADJUST-NOJUST" switch; but the hem line is never justified in any case.
2.6 BREAKING AT BLANK LINES

In FILL mode, a BREAK is caused by every blank line (just CR-LF). Redundant blank lines have no effect. To disable this response, see Section 10.1.

2.7 BREAKING AT TABS

Tabs are converted to an appropriate number of spaces by PUB. If you would like a single tab at the beginning of a text line to cause a paragraph break, put this command at the beginning of your manuscript:

.TABBREAK

This makes text lines that are indented exactly 8 spaces cause a break. Since an initial TAB is converted to 8 spaces, it will also break. The inverse of TABBREAK is TABSPACE.

2.8 BREAKING AT CARRIAGE-RETURNS

In FILL mode, PUB usually converts every carriage-return at the end of a text line to a space. If it is at the end of a sentence, it converts the carriage-return to two spaces. If you would rather that every carriage-return ending a text line caused a paragraph break, use this command:

.CRBREAK

The inverse of CRBREAK is CRSPACE.

2.9 NOFILL MODE

In NOFILL mode, one input line produces one output line, even if it falls far short of the right margin; if the line is too long, characters will be lost. Every carriage-return ending a text line in NOFILL
mode causes a paragraph break. The commands ADJUST, CRSPACE, and TABBREAK do not affect this mode.

NOFILL mode has several variations, which differ mainly in the final treatment of the output line:

NOFILL: (Standard variation) No special treatment.

CENTER: Center the output line between the margins.

FLUSH RIGHT: Shove it against the right margin.

FLUSH LEFT: Shove it against the left margin.

JUSTJUST: Justify it by the insertion of spaces.

VERBATIM: Copy text lines to output exactly as written, without changing indentations. Ignore all control characters {α#$&*\³-}. Dot in column 1 still signals a command line.

This mode speeds text processing when no formatting is needed.

SUPFRIMPOSE [n]: Suppress the LF after each line, except every n'th line.

< t < |  #  |  n <= |  % 7  Z
|  > ? /  u =>  \\ - -
abedefghijklmnopqrstuvwxyz
ABCDEFHIJKLMNOPQRSTUVWXYZ

2.10 HORIZONTAL SPACE COMPACTION

In COMPACT mode, redundant spaces in the input line are discarded. In RETAIN mode, they are retained.

More precisely, in COMPACT mode, a sequence of two or more spaces is reduced to one space, except at the end of a sentence, where it is only reduced to two spaces, and at the beginning of a text line, where all spaces are discarded. Of course, spaces may be re-inserted during justification.
2.11 VERTICAL GROUPING

In GROUP mode, all output lines are forced to appear in an unbroken column. Its inverse is APART mode. If the bottom of the output page is encountered while in GROUP mode, the whole group is moved to the top of the next column or page.

It is unwise to use GROUP mode for more than a few lines that must appear undivided in the output. Example:

```
.GROUP
EQUATION 6
  m = vt
.APART
```

This would prevent "Equation 6" from appearing at the bottom of one page and "m = vt" at the top of the next.

2.12 MODE AND SWITCH SETTING COMMANDS

The commands that select the paragraphing mode are FILL, NOFILL, CENTER, FLUSH RIGHT, FLUSH LEFT, JUSTJUST, VERBATIM, and SUPERIMPOSE [n], where n [optional] is any "SAIL" expression.

The commands ADJUST and NOJUST set the FILL-mode justification switch. TABBREAK, TABSPACE, CRBREAK, and CRSPACE control the reaction to initial tabs and to carriage-returns in FILL mode.

The commands RETAIN and COMPACT, GROUP and APART select the space compaction and column grouping modes.

At the outset of compilation, the settings in force are:

```
.FILL ADJUST TABSPACE CRSPACE RETAIN APART
```

WARNING: Switching to NOFILL or to any of its variations automatically switches to RETAIN mode. This can be overridden by following the command by COMPACT.
2.13 WORD BREAKS

The compiler processes the input paragraph in units called words. A "word" is roughly defined as a sequence of consecutive visible characters. More precisely, the compiler concatenates characters into a word until the occurrence of a word break.

A word break is normally signalled by any of the following in a text line:

> A space (or a sequence of spaces)
> A carriage-return (the end of an input line)
> The control character β, if activated (as well as some others)

Word breaks serve two functions.

(1) FILLING: In generating an output paragraph, the compiler puts as many words as can fit on each line before beginning a new line. The paragraph will only be divided for this purpose at word breaks.

(2) JUSTIFYING: During justification, extra spaces will only be inserted at word breaks.

2.14 CONTROL FUNCTIONS

Several control functions can be invoked by control characters in the text line. The user can choose the characters he wishes to serve these functions. For simplicity of exposition, a suggested character has been designated for each function.

The following control functions are available. If the description states: "BLA mode only", then in other modes, the character is treated no differently than an alphanumeric.

β Cause a word break.
α Pretend the next character is alphanumeric (this lets you slip in control characters);
# Same as α<space>; sneaks spaces into words.

2.13 - 2.14
FILL mode only, and only at the end of a word: marks the end of a sentence. This means:

(1) If followed by two or more spaces, COMPACTion only reduces the spaces to two instead of one.
(2) If followed by a carriage-return, replaces the CR by two spaces instead of one.

FILL mode only, and not the first character of a word: a hyphen. This means:

(1) If at the end of a line, prevents the word break that usually occurs there.
(2) Otherwise, permits the word to be broken after the hyphen in case it overfills the output line.

\ TAB. Insert spaces up to the next tab stop set by the TABS command (see Section 6.3).

\^n MOVE RIGHT. Leave n spaces in the output line.

\^n MOVE LEFT n characters. Create new characters (these due to Jerry Agin):

\^n M 0 V.

\$n TAB. Insert spaces up to but not including the n-th character position from the left margin (flush left is n=1). Both kinds of TAB (\ and $n) inhibit insertion of all justification spaces to their left.

For $n and \^+n, n may be any "SAIL" expression enclosed in parentheses. In the case of a single letter variable or a one-digit number, the parentheses may be omitted. RESTRICTION... $\{expression\}$ may not appear as the last thing on a text line.

Right Flush. If a \ or $n occurs later in the line, slide the text that is between here and there flush against the column designated by the tab. Otherwise, slide the rest of this paragraph against the right margin, inhibiting justification.

Center. If a \ or $n occurs later in the line, center the text that is between here and there halfway between the current column and the column designated by the tab. Otherwise, center the following text between the margins, inhibiting justification.
TEXT CONVENTIONS

\[\times\] Repeat. 'x' is any character. If \(\times\) precedes \(\Delta\), \(\Delta n\), \(\rightarrow\), or \(\leftarrow\), then use the character \(x\) instead of spaces as filler to the left of the affected text (i.e., \(\times\) puts \(\ldots\) up to the tab column).

\[\times\times\times\ldots\text{Use the repeating string xyzxyz... as filler.}\]

\[\_\ldots\_\] Underline. In the text between brackets, every visible character is underlined. "" is only a control character when paired with "\". If you want spaces underlined as well in a title, simply replace each space by an underbar.

\[\_\text{underline one word. The following word (consisting of letters and digits only) is underlined.}\]

\[\pi\] Write a wierd character. The line printer and FR-80 have these but your terminal doesn't.

To get
\[
\begin{align*}
&\text{(center dot)} \pi \\
&\text{(gamma)} \pi G \text{ or } \pi g \\
&\text{(apple delta)} \pi A \\
&\text{\(\_\ldots\_\) (integral)} \pi \text{~ (tilde)} \\
&\text{(plus-minus)} \pi \text{~} \\
&\text{(circle plus)} \pi +
\end{align*}
\]

(Examples not present in PUB.TES[S,DOC])

Switch to Command Processing. To return to text processing, use \). See Section 2.16.

\[\times\] Subscript. The text \(x\) is lowered a line (an extra line is reserved for it). \(x\) may be any single character except "" or """, or may be a sequence of characters bracketed between \([\ldots]\). A subscript must appear all on one line.

\[\times\] Superscript. Analogous to Subscript, except \(x\) may also be the single character "". Collectively, superscripts and subscripts are called somescrlpts.

These may combine and nest in interesting ways. Super/subscripts may not nest more deeply than 4.

\[x&y\] Align. "&" is only considered a control character if it is the first character following a somescrlpt \(x\).
Its effect is to align the leftmost characters of x and y by backspacing after printing x and before printing y, and to align the rightmost characters by forward spacing to the end of the longer of x and y when done. x must be a somescript. y may be any single character except "[", "*", or "]", or may be another somescript, or may be any sequence of characters enclosed between [ ]. Examples:

To get...Write
\[\text{To get...Write}\]
\[\begin{align*}
\text{j} & \quad \text{x}+j+1 \quad \text{y} \quad 1+j \\
\text{i} & \quad \text{x}+i+2 \quad \text{SUM} \quad 1+[t]+[\text{SUM}] \\
\text{2} & \quad \text{x}\times 3 \quad \text{x} \quad x+i+2 \\
\text{1} & \quad \text{t} \quad \text{1}.& \times \quad x \quad x+i+2 \\
\end{align*}\]

2.15 CONTROL CHARACTER ACTIVATION

At the outset of compilation, the only control characters recognized are the punctuators ".", ",", and "?" when they appear at the end of a word, and hyphen ("-") when it appears in the middle of a word. NOTE: Not even "{" is recognized unless you activate it. However, "}" is permanently active, because it is not a text control character but really a command language delimiter.

To turn on any control function using its suggested character ("a" for example), use the command:

\[\text{.TURN ON "a"}\]

To activate the control function using a different character, such as "+" or "β", write:

2.15
.TURN ON "%" FOR "α"
.TURN ON "β" FOR "α"

To de-activate a control character, such as "?" or "%", use:

.TURN OFF "?"
.TURN OFF "%"

One TURN command may have several operands separated by commas:

.TURN ON "%" FOR "α", "@" FOR "¬", "¬", "\"
.TURN ON "*", ",", "[", ",", ",", ",&"

The latter may be abbreviated:

.TURN ON "*[]"&"

There is a subtle problem that arises when control characters have been turned on inside a macro and the macro wants to turn off only the ones that weren't on previously. If it has done a TURN ON "*", for example, it can not undo it with TURN OFF "*", for that would always TURN it OFF whether or not it was originally on. There are two ways to do it right. One is to use block structure (see Section 5.2), but sometimes this is impossible, because blocks cause paragraph breaks. The other way is to use the command TURN ON|OFF without arguments. Its effect is to cancel the previous TURN ON|OFF that had arguments, was in the current block, and has not yet been cancelled.

For purposes of smooth exposition, in the rest of this manual it is assumed that you have turned on all the control functions and used all the suggested characters for them. Remember, though, that only the punctuators are turned on unless you command otherwise!

It is recommended that you don't turn on any characters except in short blocks where you know you'll remember they're on. Ideal places are inside macro bodies (See Section 3.1).
The control character "{" will switch to command processing right in the middle of a text line. This allows the execution of any PUB commands without first causing either a paragraph break or a word break. If you want a word break before the {, force it by means of a \ or a space.

The command most commonly executed after "{" is the Computed Text Command. A variable, a constant, or a parenthesized expression occurring in isolation is evaluated and its string value is then scanned by the text scanner as text. Example:

\[ A = \{A\}, \text{ and } B = \{ (2+2) \} \]

If the value of \( A \) was the string "u3", this would output:

\[ A = 3, \text{ and } B = 4. \]

Caution: If a \{...\} construct extends onto a second input line, be sure to begin the continuation line with a "\." -- or you won't be in the command scanner!

Caution: A string constant following the word \texttt{BEGIN} is considered a block name, as in \texttt{SAIL}, and not computed text. If you put computed text after \texttt{BEGIN}, precede it by a semicolon.

Note that a ")" (which can not be \texttt{TURNed ON} and \texttt{OFF}) switches to text processing from command processing at any time. This allows a Short Text Line to be put right into a command line:

\[ .\texttt{NOFILL } I \texttt{ am a short text line}.(\texttt{BREAK FILL} \]

which is the compact equivalent of:

\[ .\texttt{NOFILL} \]
\[ I \texttt{ am a short text line}.\texttt{FILL} \]

To summarize, there are two ways to switch from command to text processing:
and two ways to switch from text processing to command processing:

> New line with "." in column 1
>{

2.16
SECTION 3

MACROS

3.1 CALLING A MACRO FROM A COMMAND LINE

It is common to repeat an identical sequence of commands many times in a manuscript, e.g.,

\texttt{.FILL ADJUST COMPACT TURN ON "8(a8"}

You could define the following macro:

\texttt{.MACRO FAC < FILL ADJUST COMPACT TURN ON "8(a8" >}

and then whenever you want the whole sequence executed, write simply:

\texttt{.FAC}

The body of the macro enclosed between < and > is substituted for its name and then scanned by the command scanner.

A macro may also have arguments:

\texttt{.MACRO T(CHR) < NOFILL TURN ON "CHR" >}

which when called by:

\texttt{.T(a)}

expands to:

\texttt{. NOFILL TURN ON "a"}

3.2 CALLING A MACRO FROM A TEXT LINE

It is also common to repeat an awkward sequence of control characters many times in the manuscript, e.g.,

\texttt{X\{i&+[j,k]}

You could define the following macro:

\texttt{.MACRO XIJK =X\{i&+[j,k]" >}

but to call it you must get to the command scanner:

\texttt{(XIJK)}

3.1 - 3.2
When the body is substituted for the name, this becomes a string constant inside curly brackets:

\{"X+\&[j,k]\}\)

so it is considered computed text and is scanned by the text scanner, producing the output:

\(X:\ j,k\)

Again, parameters can be used:

\.

MACRO S(X,I,J,K) \("X+\&[j,k]\)"

If called by \(X(i,j,k)\) this would expand to the same as above, but called by

\{S(Y,t_1,\text{URG})\}

it would expand to \"Y+\&[t_1,\text{URG}]\" which would output:

\(Y:\ t_1,\text{URG}\)

Some macros are declared for you in an automatically loaded file PUBSTD.DFS (see Section 7.5). Other useful macros are declared in a file PUBMAC.DFS that you can REQUIRE (see Section 7.11) to be loaded; they are described in PUBMAC.TES[UP,DOC].

3.3 TEMPLATES

A "template" occurs as a part of several commands, including the macro declaration, and requires some explanation. Its purpose is to store away a piece of program for later, instead of immediate, execution.

Syntax:

c <piece of program> 

The <piece of program> may include other templates, and may span several lines. However, several rules should be followed:

1. The template should begin and end on a command line.
(2) Every $3$ or unpaired $<$ or $>$ must be preceded by a $3$.

(3) People who change COMMAND CHARACTER in mid-manuscript, note: the character beginning command lines within the template must be the one current at the time of declaration -- not the one current when you invoke it later.

3.4 MACRO DECLARATION

Syntax:
MACRO <name> [[<formal parameters>]] [;] <template>

The <name> may be one or two identifiers. The <formal parameters> if present, are in the following form:
( [<]identifier> , ... )

Each parameter preceded by $<$ is a value parameter, and when called, the actual parameter that corresponds is evaluated and the value is substituted for each occurrence in the <template> of the <id>. Every other parameter is a literal parameter, and the actual parameter is substituted unevaluated. Example:

MACRO PLUG IN(X, <Y>) = "X --- Y"

If called by: (PLUG IN(Peter Pauper, V)) where V is a variable whose current value is "Henrietta", the macro expands to: ("Peter Pauper --- Henrietta")

3.5 MACRO CALLS

A macro can be called from anywhere in a command line as well as inside curly brackets on a text line. A value parameter must be a "SAIL" expression (see Section 4.1), while a literal parameter may take any of three forms (form 2 is recommended):

(1) A string enclosed in "..." in which every " inside is represented by ". It must appear all on one line.

(2) A string enclosed in [ ... ] in which no ] appears. It may span several command lines.
(3) A string not beginning with " or | and containing no comma or right parenthesis. It must appear all on one line. Leading spaces are ignored.

Thus, all the following are equivalent:

```
PLUG IN ("Peter Pauper", V)
PLUG IN ([Peter Pauper], V)
PLUG IN (Peter Pauper, V)
"Peter Pauper --- Henrietta"
```

### 3.6 Omissions

Any actual parameter may be omitted; its value is then NULL. Furthermore, if all parameters that follow it are also omitted, all their terminal commas may be omitted.

In a macro call -- but not in a macro declaration -- the parentheses around the actual parameter list may be omitted (both -- not just one). However, care must be taken if this is done. In particular, the comma that follows each parameter must appear on the same line as the parameter. Also, the third form of literal parameter may not contain any of the following:

```
, ) ] ; } c CR
```

This convention is useful for macros to generate section headings; a typical call is:

```
.SEC SPECIAL FEATURES
```

which is equivalent to:

```
.SEC([SPECIAL FEATURES])
```

and would presumably go to a new page, update the section number, print the heading centered, and send it to the Table of Contents for inclusion there. How to do all this will unfold gradually.
3.7 RECURSIVE MACROS

If a macro is declared "RECURSIVE MACRO ...", then when its call occurs in the FALSE part of a conditional, the compiler processes the actual parameters as usual, but does not expand the macro body. Instead, the whole call is replaced by NULL.

For macros not declared RECURSIVE, their bodies are expanded even in the FALSE part of a conditional. This prevents syntactic anomalies, but attempts at recursion will create infinite looping.

Recursive macro hackers: If you call a RECURSIVE macro after a conditional statement, an empty statement must intervene:

```
. IF A<0 THEN B ELSE C ; ; RECMAC(...)
```
SECTION 4

EXPRESSIONS

4.1 DATA TYPE

PUB has only one data type: String. A string consists of zero or more ASCII characters.

Strings may participate in arithmetic operations. The string "764" is automatically converted to the integer 764 for this purpose. The result of the operation is similarly converted back to a string. Because of this convention, the data type Integer exists for all practical purposes.

4.2 VARIABLES

Variables may be declared by the command:

.VARIABLE A, BETA, C

This would make A, BETA, and C local to the innermost block in which the declaration occurred. The variables are of course type String.

Any variable that is assigned but not declared is automatically Global.

All declared variables are initialized to NULL, the string of length zero.

The name of a variable is an identifier. The first character of an identifier should be a letter. The letter may be followed by letters, digits, , and !. The identifier is terminated by any other character, including CR. Before looking up an identifier in its symbol table, PUB changes all lower case letters to upper case and changes ! to .

In syntactic descriptions in this manual, <id> will
denote any identifier, and \( <v> \) will denote a variable in particular.

### 4.3 AUTOMATICALLY DECLARED VARIABLES

Some variables are automatically declared for you by the compiler, in a block outside your outermost block. A few of these are read-only -- it is not possible to assign them a value by an assignment statement.

- **CHAR**
  - (read-only) The number of characters so far printed on the current output line, not counting justification spaces. This is sometimes an overestimate.

- **LINE**
  - (read-only) Value is zero if the current output column is empty. Otherwise, the sequential number of the last line output.

- **COLUMN**
  - (read-only) Value is zero if the current page is empty. Otherwise, the sequential number of the last column in which output went.

- **CHARS**
  - (read-only) The number of character positions remaining to be filled on the current output line.

- **LINES**
  - (read-only) The number of unused lines remaining in the current output column.

- **COLUMNS**
  - (read-only) The number of unused columns remaining in the current page.

- **TOPLINE**
  - (read-only) The line number on this page that starts the current area. (see Section 6.7).

- **FILLING**
  - (read-only) FALSE ("9") if in a NOFILL mode. TRUE if in FILL mode -- "1" if ADJUST, "-1" if NOJUST.

- **NULL**
  - (read-only) The empty string ("*").

- **TRUE**
  - (read-only) "1"

- **FALSE**
  - (read-only) "0"

- **SPREAD**
  - The intra-paragraph output line spacing. SPREAD=1 is single spacing, SPREAD=2 is double spacing, etc.

Its initial value is 1 unless otherwise specified in the (nS) CUSP option.

- **INDENT1, INDENT2, INDENT3** Control paragraph indentation (see Section 6.4).
- **LMARG, RMARG** The left and right margins (see Section 6.8).
- **DATE** Today's date, in the form October 16, 1848.
Initialized from the system DATE UUO just before your manuscript is compiled. Also available individually are MONTH, DAY, and YEAR. To fully capitalize the month or date, use the t operator described in Section 4.4 (e.g., tMONTH).

TIME  The time your compilation began, in the form 16:47.

FILE  The first name of your manuscript file. Useful for headings.

PAGE  The current page number, initially NULL. Don't declare this LOCAL!

SKIP  (read-only) May be set by extreme substring operators (see SAIL manual, 9-43). To examine the left and right halves, examine SKIPL and SKIPR.

FOOTSEP  Before the first footnote (if any) in each column, a blank line is usually left. This is because FOOTSEP is initially NULL. If you assign it another value, then that string will be printed instead.

4.4 SIMPLE_EXPRESSIONS

Most of the SAIL <string expression> syntax has been implemented. Exceptions: function designators (but there are macro calls); exponentiation; LDB, ILDB, and LOP; shifts and masks; LEAP operators.

PUB has three unary operators of its own: "t" capitalizes its argument (only lower case letters and underbar are affected). EVEN and ODD are predicates which tell whether the low order bit of the last character of a string is off or on. They are used primarily to distinguish left and right facing pages:

IF ODD PAGE THEN etc.

All the available operators are listed in the table below. If operator op1 is listed above operator op2 then op1 is performed before op2.
Expressions

Assignment expressions and conditional expressions are allowed. If used alone to compute text, enclose them in parentheses, otherwise they will look to the compiler like assignment statements or conditional statements, and the text processor won't scan their values.

In syntactic descriptions, <e> will denote a general expression -- simple, assignment, or conditional, while <se> will denote a strictly simple expression.
PUB recognizes three forms of constants: quoted, decimal, and octal. WARNING: CR will terminate any form, so don't continue a quoted constant on a new line.

A quoted constant \(<qc>\) is a sequence of characters surrounded by quotes ("..."). To include a quote in the represented string, write two quotes (\"\") in the constant.

A decimal constant \(<dc>\) is a sequence of decimal digits.

An octal constant \(<oc>\) is an apostrophe (') followed by a sequence of octal digits. The value of the octal number is taken modulo 256 and converted to a one 8 character ASCII string. The following octal codes are illegal: '0, '11-'15, '175, '177.
5.1 STATEMENT TYPES

Syntactically, any command can be a statement (abbreviated <s> in this manual). There are two categories of commands: declarative and imperative. A declaration is limited in its scope by block structure, while an imperative isn't.

5.2 BLOCK

Syntax:
BEGIN [qc] <s> ; ... END [qc]

The optional block name [qc] should be the same in both places. Mismatches will be flagged. Missing END's are calamitous to say the least!

Note that declarations don't have to be the first thing in the block. The scope of a declaration is the remainder of the innermost block in which it occurs.

Both BEGIN and END cause a paragraph break. Neither produces output, not even a blank line.

The semicolons between statements may be omitted as long as no ambiguity is presented thereby.

5.3 COMPOUND STATEMENT or CLUMP

Syntax:
START [qc] <s> ; ... END [qc]

Declarations are not local to clumps, and START and END never break. Thus:
IF TRUE THEN START FILL ; ADJUST END ;
sets the modes FILL and ADJUST and leaves them set after END. If BEGIN had been used instead of START, the scope of the modes would have extended only to END.

5.4 ONE PARAGRAPH SCOPE

Syntax:
ONCE

The command ONCE breaks and begins a local scope which is implicitly terminated by the end of the next paragraph. There must not be an END provided to match it. If a BEGIN is encountered before a paragraph breaks, then everything that has been declared in the ONCE’s scope is donated to the scope of the BEGIN’s block and the ONCE is forgotten. Examples:

.ONCE FLUSH RIGHT
George

is equivalent to:

.BEGIN FLUSH RIGHT
George
.END

and

.ONCE FILL
.ONE FLUSH RIGHT
George
.BEGIN NOJUST
Harry
.END

is equivalent to:

.BEGIN FILL NOJUST
Harry
.END

You will find the ONCE construct useful in writing macros and in formatting odd paragraphs here and there. Note that ONCE is a complete statement, unlike BEGIN, which introduces a statement that extends to the matching END. Thus, you can say "IF X THEN ONCE ELSE i-1+l" but you can’t say "IF X THEN BEGIN ELSE i-1+l".

5.4
SECTION 6
DECLARATIONS

6.1 SCOPE

(For the third time:) A declaration is in force until the occurrence of an overriding declaration or until the end of the innermost block in which it occurred (see Section 5.2 through Section 5.4).

6.2 MODE DECLARATIONS

Most of these have already been described. Some mode declarations cause paragraph breaks and some don't. The ones that first Break are: FILL, all the NOFILLS, ADJUST, NOJUST, GROUP, and APART. The ones that don't are: CRBREAK, CRSPACE, COMPACT and RETAIN.

6.3 TABS DECLARATION

Syntax: TABS <e>, ...

The expressions are sorted in algebraic ascending order and duplicates are eliminated. These then become the tab stops for the \ control character. If you tab when the target is already passed, no spaces are inserted, but justification to the left of the current position in the output line is inhibited anyway. Tab distances are measured from the left margin.

6.1 - 6.3
6.4 INDENT DECLARATION

Syntax:
INDENT [<e>] [<e>] [<e>]

This command can change the values of three parameters that control paragraph format: INDENT1, INDENT2, and INDENT3. They can also be changed directly by assignment statements. Every change is local to the current block.

INDENT1 and INDENT2 specify indentation from the left margin.

INDENT1 specifies the value of the crown indentation, which affects only the first line of output paragraphs.

INDENT2 specifies the value of the vest indentation, which affects non-first lines in FILL mode.

In NOFILL, SUPERIMPOSE, and JUSTJUST modes, INDENT1 is observed, and INDENT2 is ignored. Old PUB users note: this is a change from previous conventions.

INDENT3 specifies indentation from the right margin in FILL mode. It is ignored in NOFILL modes, and in text affected by the control characters »", """, and "".

Lines in VERBATIM, FLUSH LEFT, CENTER, and FLUSH RIGHT modes are never indented.

If any argument is omitted, the corresponding parameter remains unchanged.
6.5 PREFACE DECLARATION

Syntax:
PREFACE <e>

PREFACE N will cause every output paragraph to be preceded by an implicit SKIP N. There are actually two Preface numbers maintained -- one for FILL mode and one for NOFILL modes. The one you change by this command depends on the current mode.

Initially, the FILL Preface is 1 and the NOFILL Preface is 0.

You may want to vary the Prefaces according to SPREAD, e.g., PREFACE 1+SPREAD.

6.6 DOUBLE SPACING

Syntax:
SINGLE SPACE
DOUBLE SPACE
TRIPLE SPACE

These are standard macros that set SPREAD to 1, 2, and 3, respectively. At the end of the current block, the old value is restored. SPREAD controls the spacing between lines within a paragraph.

6.7 AREA DECLARATION

Syntax:
(TEXT|TITLE) AREA <id> [LINE[s] <e> [TO <e>]] [CHAR[s] <e> [TO <e>]]
[IN <e> COLUMNS <e> WIDE|APART]

Each page is divided into Areas into which the compiler will place output text. The areas HEADING, TEXT, and FOOTING are pre-declared for you as follows:

6.5 - 6.7
.AREA TEXT LINES 4 TO 51;
.TITLE AREA HEADING LINES 1 TO 3;
.TITLE AREA FOOTING LINE 53;

Each area is a rectangle that must lie within the bounds set by the last PAGE FRAME Statement (see Section 7.13). If the CHARS clause of the AREA declaration should be omitted (as it is in the predeclared areas), then the full width of the Page Frame is assumed. Similarly, if the LINES clause should be omitted, full page height is assumed.

An area is assumed to be a TEXT AREA unless "TITLE AREA" is stated explicitly. The only difference between a TEXT AREA and a TITLE AREA is the action taken when so many lines are written in it that it overflows. In the case of a TEXT AREA overflow, all areas are "closed", the whole page is written out, and a new page is begun. In the case of a TITLE AREA overflow, an error message is issued.

An area has one column unless an IN..COLUMNS clause is included, e.g.,

.AREA LISTING LINES 30 TO 50 IN 2 COLUMNS 7 APART

Either how wide each column is or how far apart they are can be specified.

Associated with each area are a left and right margin, located initially at the left and right edges of each column. How they can be moved inward and back out is explained in Section 6.8.

The bottom line of a TEXT AREA is not used except to print the last line of a paragraph. This convention prevents the appearance of "widows" at the top of a page.

6.8 MARGIN CONTROL

Syntax:
NARROW <se> [ ,<e>] <paragraphs and statements> WIDEN

NARROW and WIDEN cause paragraph breaks. NARROW L,R moves the left and right margins of the current PLACE area inward L and R, respectively. Their initial positions are determined by the AREA declaration. L and R may be negative. If either is omitted, it is
assumed 8. The matching WIDEN restores the margins to their former positions.

Alternatively, you can have:

\text{WIDEN (se) [\,<e>] <paragraphs and statements> NARROW}

which moves the margins outward. Both kinds can nest.

These commands change the automatically declared variables \text{LMARG} and \text{RMARG} (the Left and Right Margins). They can also be changed directly by assignment statements; such changes are local to the current block and to the current NARROW-WIDEN nest of the current PLACE area.

Upon exit from a block, if the margin positions are different than they were at the beginning, then they are restored and instead of the usual Break, END causes a \text{CONTINUE} (See Section 7.15), which treats the next line as a vest instead of a crown line.

Every paragraph is output in the following format:

\[ \begin{array}{cccc}
\text{A-1} & \text{A} & \text{Left-margin} & \text{B} & \text{Right-margin} \\
\text{I1} & \text{I2} & \text{I3} & \text{I4} & \text{I5} \\
\text{L} & \text{L} & \text{L} & \text{L} & \text{L} \\
\text{P blank lines} & \text{P blank lines} & \text{P blank lines} & \text{P blank lines} & \text{P blank lines} \\
\end{array} \]

\text{The format suggested by the diagram above may be achieved by the following declarations:}

\text{.AREA TEXT CHARS A TO B ;}
\text{.NARROW L, R ;}
\text{.FILL ; PREFACE P ; INDENT I1, I2, I3 ;}

The automatically declared variable \text{CHAR} and all tab stops count characters from the left margin; they may at times be negative. It is possible to produce output past the right margin by setting tab stops out there.
6.9 PLACE DECLARATION

Syntax:
PLACE <area id>

This breaks and switches output to the named area. This may be done many times on the same page; output
to each area always picks up where it left off.

If a column of the current Place Area overflows, output continues at the top of the next column. When
the last column overflows, either a new page is begun (if a TEXT AREA overflowed) or an error message is
given (if a TITLE AREA overflowed).

6.10 TEMPORARY AREAS

This section is complicated. Only read it if you need
to create temporary areas within a page. To do this, you need to know where you’re up to on the page so
far. If there’s not enough room for the new area, you will want to go to a new page. Then a temporary area
(local to a block) should be declared, and text placed in it. Finally, you may need to find out where you’re
up to on the page again before you exit that block, skip over the temporary area, and resume regular output.

A more convenient way to handle this may be available
in a future version of PUB in the form of a BOX FRAME
declaration. In the meantime, you should be able to
get by as follows.

To determine the top line on the whole page that the
current PLACE area uses, look at TOPLINE. Within that
area, lines are numbered starting from 1 -- but if
TOPLINE is 6 that means "line 1" of the area is really
the 6'th line on the page. We can distinguish P-lines
(numbered relative to the whole page) from A-lines
(numbered local to the area); A-line 1 might be P-line
6.

To determine the column number and A-line number of
the last line output in this area, look at COLUMN and

6.9 - 6.10
LINE. To determine how many empty columns remain in this area, look at COLUMNS; for how many empty lines remain in this column, look at LINES. Note that LINE+LINES can be less than the full height of the area, because a few lines at the bottom may be occupied by footnotes.

To declare a two-column area 10 lines high for a table starting just below the last output line, write:

```
.IF LINES < 10 THEN NEXT PAGE ;
.BEGIN
.AREA AXE LINES TOPLINE+LINE TO TOPLINE+LINES+9 IN 2 COLUMNS 5 APART
.PLACE AXE
   <table text>
.END
.GROUP SKIP 10
```

Adding TOPLINE in the AREA declaration is necessary because it expects P-line numbers.

6.11 COUNTER DECLARATION

Syntax:

```
COUNT <id> [INLINE] [FROM <e>] [TO <e>] [BY <e>] [IN <counter id>]
   [PRINTING <e>|<template>]
```

This is PUB's excuse for a FOR statement. The user can declare certain identifiers to be counters, and arrange them in hierarchies. A counter is anything you want numbered sequentially. Typical counters are SECTION, APPENDIX, SUBSECTION, PAGE, FOOTNOTE, EQUATION, TABLE, FIGURE, NOTE, and REFERENCE. A common hierarchy is:

```
SECTION
 / \ 
SUBSECTION PAGE

FOOTNOTE
```

The declarations for this hierarchy might be:

```
.COUNT SECTION FROM 1 TO 9
.COUNT SUBSECTION FROM 1 TO 99 IN SECTION
.COUNT PAGE FROM 1 TO 99 IN SECTION
.COUNT FOOTNOTE INLINE FROM 1 TO 9 IN PAGE
```

6.11
The "IN" clause specifies the hierarchical parent counter; it must have been previously declared.

The range of each counter is specified by the FROM, TO, and BY clauses; if omitted, FROM 1, TO 18, and BY 1 are assumed. The counter need not ever attain the TO-value; it is just an upper limit used to determine the maximum number of characters needed to print the counter in case it is referred to in a Forward Cross-Reference (see Section 9.1).

The INLINE option suppresses a BREAK which is otherwise automatically generated before each stepping of the counter.

The PRINTING clause is explained in Section 7.3.
SECTION 7
IMPERATIVES

The commands in this section are non-declarative. That means that their scope is not limited by block structure. An occasional exception to this rule is the assignment statement; when the assigned variable is SPREAD, LMARG, RMARG, INDENT1, INDENT2, or INDENT3, then the change is local to the current block.

7.1 ASSIGNMENT STATEMENT

Syntax:
\(<v> = <e>\)

Example:
.SPREAD - T = r - 4 ;

Assignment statements don't break.

7.2 CONDITIONAL STATEMENT

Syntax:
IF \(<e>\) THEN \(<s>\) ELSE \(<s>\)

Example:
.IF ODD PAGE OR LINES < 10 THEN START
this text line
.END ELSE START
that text line
.END

"IF", "THEN", and "ELSE" don't break.

7.3 NEXT COUNTER VALUE STATEMENT

Syntax:
NEXT \(<\text{counter id}>\)
This breaks (unless INLINE appeared in the designated counter's COUNT declaration), and the counter is stepped to its next value. The first time it is called, the counter's value is initialized.

Every COUNT declaration automatically declares two variables with special properties, and initializes them to NULL. The first variable has the same name as the <counter id>, e.g., "SUBSECTION", and its value is always the decimal integer value of the counter. The other variable's name is constructed by appending an "!" to the <counter id>, e.g., "SUBSECTION!". The two variables are called the counting or "C-value" and the printing or "P-value", respectively.

The precise effect of the statement "NEXT U" in terms of these variables is as follows.

(1) If U=NULL, then U is set to the FROM value of the COUNT declaration; otherwise, U is incremented by the BY value of the COUNT declaration.

(2) The P-value is computed under control of the PRINTING clause of the COUNT declaration (just how will be explained shortly). In addition, the P-value is assigned to the variable "!" as a convenient abbreviation.

(3) If the counter has hierarchically subordinate counters, all their C-values and P-values are set to NULL; thus, when you say NEXT SECTION, SECTION becomes a new number while SUBSECTION, PAGE, and FOOTNOTE become NULL (assuming the hierarchy of the recent example). Exception: If "PAGE" was one of the subordinate counters, after it is cleared it is immediately initialized.

The computation of the P-value depends on the PRINTING clause of the COUNT declaration. There are three cases.

Case One The PRINTING clause is omitted. P-value is the same as the C-value.

Case Two PRINTING <template>. After each C-value is computed, the expression inside the template is evaluated, and the result is the P-value. Thus, the template should contain an expression which involves the C-value. Examples:

7.3
.COUNT FOOTNOTE TO 9 IN PAGE PRINTING $[1 TO FOOTNOTE]$;
When FOOTNOTE=1, this sets FOOTNOTE=**; when FOOTNOTE=5, this sets FOOTNOTE=*****.

.COUNT SUBSECTION IN SECTION PRINTING $<$SECTION! & "." & SUBSECTION$>$;
When SECTION!=4 and SUBSECTION is counted to 7, SUBSECTION becomes 4.7.

Case ThreePRINTING <e>. The expression is evaluated only once, at
declaration time, and its value serves as a pattern for
conversion of the C-value to the P-value. For example, the
pattern "(i)" will produce P-values such as (iv) and
(xlii), and the pattern "!-A" will produce P-values such as
6-A and 33-L. The pattern should evaluate to a string of
the form:
<prefix> [ ! <middle> ] <format> <suffix>
where <prefix>, <middle>, and <suffix> may be empty, and
<format> must be 1, a, A, i, or I. This is roughly
equivalent to the template:
<<prefix>>&amp;! & <middle>&amp; CONVERT(<counter id>,<format>) & <suffix>>

Here’s what happens. The P-value becomes this pattern
after substituting the parent counter’s P-value for "!" (if
present) and after converting the C-value to the specified
format and substituting it for <format>. Before explaining
formats, let’s give a simple example:

.COUNT SUBSECTION IN SECTION PRINTING "1.1";
Here, <prefix> and <suffix> are empty and <middle> is ".".
Format "1" is no-conversion, so the C-value is substituted
unchanged for the ".1". The P-value of SECTION is
substituted for the ".1". Say SECTION! is 6 and SUBSECTION
is stepped to 4; then SUBSECTION! becomes 6.4.
The various <format> conversions are:
1
No conversion
I
Upper case Roman, i.e., 59 = LIX
1
Lower case Roman
A
Capital letters A,...Z,AA,BB,...ZZ,AAA,...
a
Little letters

7.4 NEXT PAGE STATEMENT

This variety of Next Statement has special properties.
Before stepping the PAGE counter, it "closes" and
outputs all the areas on the current page and sets up a fresh page frame.

Conversely, when a page is closed due to text area overflow or "SKIP TO COLUMN 1", etc., the page counter is automatically stepped before the new page is opened, as if the command NEXT PAGE had been given.

7.5 HEADINGS AND FOOTINGS

As soon as the first line is ready to be placed in a page, the page is "opened". There are fancy ways you can create fancy headings and footings at that time or at other times (see Section 10.2). For simple cases, the following command (which is actually a call to a built-in macro) is provided.

Syntax:

```
.EVEN|ODD|EVERY HEADING|FOOTING ( <text>, <text>, <text> )
```

On left-facing (EVEN), on right-facing (ODD), or on all pages (EVERY); on the top line (HEADING) or the bottom line (FOOTING); the three <text>s will be printed at the left edge, the center, and the right edge of the page, respectively. A <text> may not contain quote ("). Example:

```
.ODD HEADING({DATE}, MACHINES THAT WONDER, (SUBSECTION!})
```

This would cause to print on the top line of every right-facing page something like:

```
July 14, 1789    MACHINES THAT WONDER
```

7.6 SECTIONING

The correct way to start every new section on a new page is:

```
.NEXT PAGE ; NEXT SECTION ;
```

If the order of these were reversed and if SECTION were the parent of PAGE, then NEXT SECTION would change PAGE's value before the old page got written out.

7.5 - 7.6
7.7 COMMAND CHARACTER STATEMENT

Syntax:
COMMAND CHARACTER <qc>|<v> ;

The parameter must evaluate to a single character string. That character subsequently assumes the function of dot in column 1. Since this command is not a declaration, its scope is global; i.e., it is not affected by block nesting. Don't omit the semicolon!

7.8 PORTION DEMARCATION

Syntax:
PORTION <id>

The manuscript may be (ought to be) divided into portions, such as TITLEPAGE, CONTENTS, THESIS, APPENDICES, NOTES, INDEX, and BIBLIOGRAPHY.

Precede each portion by a Portion Demarcation, e.g.,

PORTION NOTES

The compiler will assure that each portion starts on a fresh page and ends with a paragraph break.

In general, the order of appearance of the portions in the manuscript becomes their order of appearance in the document. However, this is not always convenient. For example, the entries for PORTION CONTENTS are usually generated during the processing of the other portions (see Section 7.9); therefore, PORTION CONTENTS must appear at the end of the manuscript. To show where you would like its output to be inserted in the document, use the statement:

INSERT <id>, ...

which is sort of a "forward portion demarcation".

Example:
7.9 SEND STATEMENT

Syntax:
SEND <portion id> [ ; ] <template>  

With every portion is associated a file called its generated file. A generated file is in manuscript format. The idea is that you can avoid manual preparation of certain portions of your manuscript (such as the table of contents and the index) by making PUB prepare them mechanically.

The generated file of a portion may remain empty, or may be written on during the processing of portions that appear earlier in the manuscript. For example, the generated files for PORTION CONTENTS and PORTION INDEX are usually generated during the processing of earlier portions.

The SEND statement writes a template onto the end of the generated file for the named portion. The portion must appear later in the manuscript. Before sending the template, the compiler substitutes for every occurrence of:

\{ <v> \}

the (unquoted) value of the variable <v>.

Thus, if S=6 and PAGE=26, then the statement:

7.9
.SEND CONTENTS c
Section (S) -- Findings \[\rightarrow (PAGE)
\]

would send the following data to the file associated with PORTION CONTENTS (CRLF shown here for clarity):

\(<\text{CRLF}\>
Section 6 -- Findings \[\rightarrow 26<\text{CRLF}\>

If a similar SEND were performed at the beginning of every section and subsection of the manuscript, then by the time the compiler reached PORTION CONTENTS, its generated file would be in perfect manuscript format, e.g.:

Section 1 -- Introduction \[\rightarrow 1
Section 2 -- Remarks \[\rightarrow 6
(\text{etcetera})
Section 6 -- Extraneous Factors \[\rightarrow 26
(\text{etcetera})
Section 413 -- Conclusions \[\rightarrow 864

The empty command lines are of course harmless; their inclusion was caused by the method of conforming with one of the rules of writing templates: a template must end on a command line. Of course, a template alternatively could end with a text line terminated by "\"{\".

The subtleties of generated files are discussed in the Appendices to this manual. If you would rather postpone becoming an expert on the technique, you can still obtain a table of contents and index by the use of someone else's macros. These macros (STANDARD FRONT and STANDARD BACK) are stored in file PUBMAC.DFS[1,3] and described in file PUBMAC.TES[UP,DOC].

7.9
7.10 RECEIVE STATEMENT

Syntax:
RECEIVE [<qc>|<v>]

This statement substitutes the generated file of the current portion for the statement and causes the compiler to compile it. Thus, PORTION CONTENTS typically consists of a few mode setting commands, a title, and then the statement "RECEIVE".

If the optional parameter is present, it should evaluate to a string of one or two characters, say "L" or "LR". The presence of the parameter causes the generated file to be alphabetized before it is compiled. Entries in the file should be in the form: 

<8 or more characters> L <Key> [ R <8 or more characters> ]

PUB will alphabetize the entries by the <Key>s using the ASCII collating sequence, except lower case letters and "" are ranked with the corresponding upper case letters and "#". Then it will read the alphabetized file, including the L and R characters.

WARNING: The characters L and R must be chosen carefully. They may not occur in the generated file in any role except as key delimiters. Furthermore, the characters ",", ",", ",", ",", may not be used. It is legal for L and R to be the same character.

If your index is very large, PUB may exhaust SAIL string space trying to alphabetize it. The error message "STRING SPACE EXHAUSTED" will be typed out. Unfortunately, the only way to get more string space is to REEenter the program and go through the ALLOC ritual (see SAIL manual 14-22). The original allocation is presently 4000 words; try 7000 or 10000.

To generate a simple index, use SEND statements such as:

.RECEIVE "@"
The file generated will be quite funny-looking, but will result in the document:

```
...
Keyshmurd ...................................... 9
Keyword ......................................... 15
...
```

Rather than all those hideous dots, it is suggested that the number immediately follow or precede the word.

To learn how to create two-level and KWIC indexes, see Appendix A.3.

### 7.11 REQUIRE STATEMENT

**Syntax:**

```require <qc>|<v> SOURCE FILE```

The parameter must evaluate to a file name, e.g., "BRAIN.BAZ[H,AHA]". The file must be in manuscript format. It will be substituted for the REQUIRE statement and compiled. If the last line of the file is a text line, PUB will still be scanning for text when it resumes the original file, so don't put a semicolon after the REQUIRE statement -- put a CR.

REQUIREd files may REQUIRE other files. You will run out of channels if the number of open SOURCE FILES, generated files (extension ".PUG") and pass one output files (extension ".PUI") exceeds 15.

### 7.12 SKIP STATEMENTS

**Syntax:**

```
[GROUP] SKIP [<se>]
SKIP TO LINE <e>
SKIP TO COLUMN <e>
```
SKIP N breaks and leaves N blank lines in the document. If N is omitted, SKIP 1 is assumed. Note that N must be a simple expression -- no assignment or conditional expression is allowed unless it is parenthesized.

Blank lines that would appear at the top of any column are automatically suppressed; if you want to force them to appear (say, to leave space for a drawing), use GROUP SKIP.

If you are going to run off the same document with different values of SPREAD (i.e., sometimes single space and sometimes double space), you should write all your KIP statements with this factor taken into account, e.g., SKIP 2*SPREAD-1.

SKIP TO LINE N outputs blank lines up to but not including the N'th line (A-line) in the current column of output text, unless the N'th line has already been passed, in which case it goes to the N'th line in the next column. The most common use of this statement is SKIP TO LINE 1, which assures you are at the top of a clean column.

SKIP TO COLUMN N outputs blank lines until it reaches the top of the N'th column of this or the next page. If the compiler is already at the top of the N'th column, this statement does nothing. The most common use is SKIP TO COLUMN 1, which goes to a new page unless a new page has just been begun. (To go to a new page regardless, use "NEXT PAGE" -- See Section 7.3).
7.13 PAGE FRAME STATEMENT

Syntax:

```
PAGE FRAME <e> HIGH <e> WIDE
```

This statement causes a break, goes to a new page, and sets up its size as specified (number of lines, number of characters). No output can appear outside the bounds set by this statement. Area declarations normally follow it.

7.14 BREAK STATEMENT

Syntax:

```
BREAK
```

If a paragraph has been started, BREAK terminates it. Breaks are implicitly caused by many other commands and conditions.

7.15 CONTINUE STATEMENT

Syntax:

```
CONTINUE
```

Sometimes you would like to insert an indented quotation or equation in the middle of a paragraph of prose. After the quotation, a paragraph break is necessary to terminate it. Unfortunately, the break also causes the next output line to be treated as a crown line (in FILL mode). If you would rather continue the interrupted paragraph than start a new one, use the command CONTINUE. It breaks, but causes the next paragraph to have no crown, as long as no regular breaks intervene.

7.16 DEVICE STATEMENT

Syntax:

```
DEVICE LPT|MIC|TTY
```
This statement is equivalent to setting the /L, /M, or /T switch when starting up PUB. It has no effect on Pass One output. However, Pass Two reacts to the switch setting as follows.

DEVICE MIC makes the document file be in FR-80 command format. Pass Two will output two temporary files with extension ".RPG" and automatically run a program called "TXTF80", written by Russ Taylor, which generates the document file. Simply copy this file to a tape and bring it to the FR-80 operator at Lockheed.

DEVICE LPT and DEVICE TTY produce identical output unless the page frame is higher than 53 lines. In that case, DEVICE LPT uses strange control characters instead of line feeds to inhibit page-ejection. DEVICE TTY assures that only line feeds are issued.

Files produced under DEVICE TTY can be edited with TEO. Hardly any document file can be edited with SOS or TVEDIT, which don't like CR's without LF's (used for underlining and superimposing).

7.17 UNIMPLEMENTED STATEMENTS

The following commands have not been implemented, but ought to be, so their names are reserved words:

CASE statement -- SAIL-style
ALIGN statement -- it would align output to side-by-side areas
PACK statement -- it would align the bottoms of parallel columns
BOX FRAME STATEMENT -- to insert a table in a corner of the page
LOCK statement -- to protect lines from being moved by a BOX FRAME
SHOW statement -- to output to the console during compilation
NOPRINT mode -- would suppress output
PRINT mode -- would renew output

LDX output will soon be available. This will require a few new commands and automatically declared variables, probably the following:

7.17
DEVICE LDX
FONT <font-name> ON <file-name> -- would declare a font
RASTER <e> UNITS PER INCH -- to specify raster density
BELOW, ABOVE, ONLEFT, ONRIGHT -- like LINES, LINE, CHAR, CHAR
CHARACTER <name> <font-name> <coordinates> -- define character
SECTION 8
COMMENTS

There are two ways to get comments into a command line -- not in the middle of an identifier or constant, however.

Syntax:
COMMENT anything but semicolon ;

This one is probably familiar. The other one is:
<< anything but two greater-than-signs in a row >>

Comments may extend over several command lines. If you leave out the terminator, you'll lose everything up to the next text line.

Comments have no effect on the document. Why would you want them? Maybe you could write English comments on a French manuscript to help you find your way around. Or maybe you've got complicated macros that need explanation. Or you left out a bunch of text and want to leave a reminder in the manuscript. Don't ask me why -- I just implemented them.

To get a comment into a text line, use curly brackets around a regular comment {<< Like this >>}. Continuation lines must be command lines.
SECTION 9
LABELS AND CROSS-REFERENCES

9.1 CROSS-REFERENCES

A cross-reference is like computed text in that it causes characters to be processed by the text scanner, and in that it normally appears between curly brackets in the midst of a text line. First there will be an example, and then an explanation.

... invented by Owsley (See (“Page!” PURE)) ...

This might produce output such as:

... invented by Owsley (See Page 3-7) ...

The identifier PURE is called a label and must be defined once and only once in the manuscript, either before or after the cross-reference. Every label has a value, much like a variable, but the value is assigned only once, at the place the label is defined. Several cross-reference statements may reference the same label. Labels will be explained further in Section 9.2.

There is a major problem in handling forward-references in a text-justifier. The justifier must know how many characters there will be in the value before it can decide where to break off the output line and how many spaces to insert during justification. PUB solves this problem by two slightly awkward hacks. One hack is that you must state or imply an upper limit on the number of characters, so it can leave enough room for it and not break off the line too soon. The other hack is that PUB has two passes. In the second pass, the actual values are substituted, and only then are lines justified.

To "state or imply" the upper limit, you have a choice of three forms of cross-reference.

Form 1: [ <e> ] <label id>
Those are really square brackets!!!! Write them in the manuscript, really! The expression evaluates to a number which is the maximum number of characters, and the value printed is the value of the label <label id>. If this is a backward reference (unless to the current page), the number is ignored, so you can say just: {[]} <label id>. You can't just say the label name.

**Form 2:** <counter id>[!] <label id>
Those square brackets mean the "!" is optional -- don't write them in the manuscript! This form means that the value of the label will be the C-value (if there is no "!") or P-value (if there is an "!") of the counter <counter id>. PUB can compute the upper limit at declaration time by looking at the TO clause of the counter's COUNT declaration and at the PRINTING clause, so you don't have to state it. In case the PRINTING clause used a template, the upper limit is determined by a not quite foolproof heuristic: suitably large numbers are substituted temporarily for the C-value of the counter, for both values of its parent, and for !, and the expression is evaluated.

**Form 3:** "<counter id>[!]" <label id>
This is identical to form 2, except before sending the value of the label to the text scanner, it sends the string (without the "!") and a space. Thus, the following are exactly equivalent:

... See Section {Section! X} ...
... See {"Section! X} ...

The macro used for cross-references in this manual was:

```plaintext
.MACRO YON(LBL) <"Section " ; SUBSECTION! LBL >
```

which was called like this:

... lots of useless words (See {yon someplace}). Lots more ....

### 9.2 LABELS

The syntax of <label definition> and <label definer> are defined (::=) as:

**Syntax:**

9.2
<label definition> ::= <label id> : <label definer>
<label definer> ::= [NEXT] <counter id>[i] \ | \ <e> \ | \ <text line>

Here is an example of each kind of <label definer>:

.INTRO: NEXT SECTION
.LAPLACE: IF ODD PAGE THEN PAGE ELSE PAGE+1
.EXPLANATION: This is the line that is labelled by the label "EXPLANATION".

The first kind of definer makes the value of the label be the current C-value (if there is no "!" or P-value (if there is an "!") of the named counter. If NEXT is present, the counter is stepped before the assignment is made. The second kind makes the value of the label be the value of the expression. The third kind makes it be the value of PAGE! for the page that the <text line> begins on.

The first and third kinds of definer both cause a consistency check between all forward-references to this label that mention a counter name and the counter named here. The second kind does not cause a consistency check. CAUTION: There is no consistency check on backward-references. There is no check when a cross-reference specifies a P-value (or C-value) that the label definition also specifies a P-value (or C-value). So if the output is incorrect, check for such inconsistencies yourself.

A label must always be on a command line, or inside curly brackets on a text line. If the /D switch was used to obtain debugging information, then the label will be printed on the right side of the document.

There is a slight ambiguity if your <label definer> involves PAGE (or PAGE!). If you use the first or third kind of definer, the evaluation is deferred until the next text line is output, i.e., until the next paragraph break or overflowed FILL line. Thus, you may think of a PAGE label as labelling the following text line; even if a new page is begun before that line is output, the label value will be the page on which that line appears. On the other hand, if you use the second kind of definer (i.e., an
expression involving PAGE or PAGE!), the value used will be the current one at the time of evaluation -- even if the following text line doesn't get output until the next page.

Note that L:NEXT PAGE will go to a new page and then assign L a value when the first line is output to that page. If the page is blank, the assignment is deferred another page. All this kludgery is to insure that if you say: See ("Page" X) there will really be something to see! Be sure to position the label just before the text line you want them to see.
A Text Response enables you to detect certain conditions in the text, and to call macro-like subroutines without using (...) Presently, four conditions can be responded to:

1. Page Mark or Form Feed
2. Blank Text Line
3. Indented Text Line
4. Presence of a one-to-five character signal

Syntax:
AT <condition> [;] <template>

When the condition is detected, the characters causing it are replaced by:

{ <template> }

which is immediately executed.

To dis-declare a response exit the block in which it was declared, or use a null template:

AT <condition> [;] =>

The various <condition>s available are as follows:

AT PAGEMARK responds to an SOS page mark or a TECO <CR LF FF>. A typical use is:

.AT PAGEMARK cSKIP TO COLUMN 1>

AT NULL responds to a blank text line. There is a predeclared response for this:

.AT NULL <IF FILLING THEN BREAK ELSE SKIP 1>

This is the only reason that blank lines break in FILL mode. You can redeclare it to treat them otherwise. If you want them ignored, write:

.AT NULL <IF NOT FILLING THEN SKIP 1>

10.1
RESPONSES

AT \( N \), where \( N \) is a positive integer \( N \), responds to each text line indented exactly \( N \) spaces. For example, the standard macro \texttt{TABBREAK} declares:

\texttt{.AT 8 \text{IF FILLING THEN BREAK ELSE "}  } \\
\texttt{AT S, where S is a string beginning with a non-alphanumeric character, responds to the occurrence of that string (up to 5 characters long) appearing uninterrupted in a text line. This one \langle \text{condition} \rangle can be followed by a formal parameter list of up to five arguments, e.g.:

\texttt{.AT "\%\%" A "\$" B "\%" C "\$" \langle SEND INDEX \rangle A B-C(BREAK \rangle \rangle }

Each parameter name is followed by a string containing a single character called its delimiter. When the signal (in the example, \%\%) is spotted in a text line, the compiler continues to scan for actual parameters, terminating each scan at the appropriate delimiter. An actual parameter may span several text lines, but each carriage-return is converted to a space if this happens. No parameters may be omitted. None are evaluated -- they are all literal.

10.2 TRANSITION RESPONSES

A transition response permits the specification of an action to be taken at the beginning or end of any counter or at the opening or closing of any area.

Syntax:
\texttt{BEFORE|AFTER \langle \text{counter id} \rangle | \langle \text{area id} \rangle \ [ ]; \langle \text{template} \rangle }

Here is how this response fits into the \texttt{NEXT} algorithm. To execute \texttt{NEXT SECTION}, \texttt{PUB} performs six steps:

1) If \texttt{SECTION} NULL then do \texttt{AFTER SECTION} response (if any)
2) Step \texttt{SECTION} using \texttt{FROM} and \texttt{BY} values
3) Compute \texttt{SECTION!} using \texttt{PRINTING} clause, and assign it to \!
4) Do \texttt{BEFORE SECTION} response (if any)
5) For each sub-counter of \texttt{SECTION}, and each of their sub-counters:
   a) If \texttt{C-value} NULL: \texttt{AFTER} sub-counter response (if any)
   b) Clear sub-counter \texttt{C-value} and \texttt{P-value} to NULL
   c) If it is PAGE, initialize \texttt{PAGE} and \texttt{PAGE!}
6) Re-assign \texttt{SECTION!} to \! (steps 4 and 5 may have changed it)
The END of the block in which any counter is declared (except PAGE) also causes Step 1 to occur.

The most useful transition responses are BEFORE PAGE and AFTER PAGE. The BEFORE PAGE template is executed not immediately after NEXT PAGE is executed, but a little later, when the first line of output is ready to be put on the new page. BEFORE <area id> responses are similarly deferred until the first output is ready for the named area. The AFTER PAGE template is executed just before the page is written out -- it had better not place into a text area that is already full!

One transition response is pre-declared for you:

BEFORE PAGE < STANDARD TITLES >

STANDARD TITLES is a pre-declared macro that prints the headings and footings set up by EVERY HEADING and its friends. You can of course redefine this response.

Every transition response template is automatically a block. It is surrounded by BEGIN 1?03 ... END 1?03 or something of that sort (which you will occasionally spot in error messages). It is usually necessary to declare all the mode settings and formats you want to be in force inside the <template>. Consider the case of BEFORE PAGE. At the time that a page happens to overflow, it is entirely possible that the compiler is in the midst of a FILL block with INDENT 10,20, all control characters off, and the NOFILL PREFACE set to 2. However, the headings and footings should be processed in NOFILL mode with preface 0, no indentation, and several control characters activated. You might declare:

.BEFORE PAGE < NOFILL ; INDENT 0 ; PREFACE 0 ; TURN ON "{<" 
.PLACE HEADING 
{DATE}-AUTOMATIC VETERINARIAN-{SECTION!}

){SUBTITLE!}
.PLACE FOOTING 
{IF ODD PAGE THEN "{*}{PAGE!}
.5

10.2
SECTION 11

FOOTNOTES

Every column of every area has two portions, the calf and the foot. During preparation of the calf, paragraphs can be sent to the foot. A column is considered full when the sum of the lines in the calf and the foot exceeds the height of the area.

You cannot declare Portion FOOT nor do a RECEIVE for it. The compiler automatically declares a FOOT for each area you declare, and does a RECEIVE for it after each line is placed in the calf. Your only foothold is the command "SEND FOOT".

Example:

.COUNT FOOTNOTE INLINE IN PAGE PRINTING "(1)"
.AT "$$" ENTRY "*" = NEXT FOOTNOTE ; ! ;
.SEND FOOT < TURN ON "{* PREFACE 1 SPREAD-1 INDENT 0,0
{!} ENTRY
.BREAK > >

An example of the use of the "$" signal response is:

...have been shown $$Abelwitz has obtained similar results in his experiments with hippopotami.*.
    which will print in the calf:
...have been shown (iv).

    and in the foot:

(iv) Abelwitz has obtained similar results in his experiments with hippopotami.

There is an implicit BEGIN FILL ; ... END around footnotes.

---

*) Thanks are due to Profs. G.I. Wish and O.U. Kidd of Steady State University for this example.
APPENDIX A

THOROUGHLY EXPLAINED EXAMPLES

A.1 SECTIONING MACROS

At the beginning of each section, it is customary to perform the following ritual:

<<1>> Begin a fresh page
<<2>> Count up the section number
<<3>> Print the section number and name
<<4>> Skip a few lines
<<5>> Send the section number, name, and page number to the CONTENTS

See the document PUBMAC.TES[UP,DOC] for a description of the macros STANDARD FRONT and STANDARD BACK, which do all these things for you. If you want to define your own macros, read on.

A macro to perform all the steps listed above (commented for reference) is:

.MACRO SEC(NAME)
<<1>> NEXT PAGE
<<2>> NEXT SECTION
<<3>> ONCE CENTER
\_SECTION {!}_\_ -- \_NAME_\_\_
<<4>> SKIP 3
<<5>> SEND CONTENTS c
{!}\\NAME-\{PAGE!}{33

A sample call on this macro is:

.SEC SPECIAL FEATURES

At the beginning of each subsection, a similar macro can be called. The one below prints the subsection name on a line by itself but prints the subsection number at the beginning of the first line of the first paragraph of the subsection. It makes sure that there are enough lines for all this to be contiguous; if not, it first goes to a new page. Finally, it allows the macro call to specify a label to be used in cross-references to this subsection.

-
Notice that nearly all formatting is accomplished in the SEND template.

Long titles will not be handled by the portion declared above. Instead, FILL mode must be used. The following declarations are approximately what was used to produce the table of contents of this manual.

`.PORTION CONTENTS
.FILL CRBREAK NOJUST
.INDENT 0, 35, 10
.TABS 10, 15, 25
.SKIP 2
.RECEIVE

A.3 ONE-LEVEL INDEXES

There is presently no way to have a phrase indexed automatically every time it occurs. PUB now requires that you mark each phrase to be indexed at every occurrence that you wish to be referenced. The AT <signal characters> response is particularly useful for this purpose.

Choose a character or a pair of characters that will never appear in the text of your manuscript; for example "%". This will be your signal that the next phrase is to be indexed. Then choose a character that will never appear in such a phrase; for example, ">". This will be your end-of-phrase delimiter. Suppose you would like the phrase "Simple variables" to be indexed. Find each occurrence of this phrase in your manuscript and surround it by your chosen characters:

Nothing warms the heart like %<simple variables>. We
THOROUGHLY EXPLAINED EXAMPLES

**.PORTION INDEX**

**.NOFILL**

**.LETTER = PHR = NULL**

**.AT "<" PHRASE ">" PGNO "v" c**

**.IF "PHRASE" = PHR THEN START },#PGNO( END**

**.ELSE START COMMENT NEW PHRASE, GO TO A NEW LINE ;**

**.IF ^LETTER = t"PHRASE"[1] THEN BREAK**

**.ELSE SKIP 1 ;**

**PHRASE##PGNO{**

**.PHR = "PHRASE" ; LETTER = "PHRASE"[1] ;**

**.END } ; COMMENT END SIGNAL RESPONSE ;**

**.RECEIVE "<" ; COMMENT ALPHABETIZE BEFORE READING ;**

RECEIVE will alphabetize the entries. The "AT" signal response will analyze each entry and make a couple of checks before outputting it:

(1) If the phrase is the same as the one just printed, it simply appends the page number of the new entry to the last line printed. For example, if the alphabetized file contained:

```
...<Signs>2.4v<A<Signs>3.7v...
```

then the index would include:

**Signs 2.4, 3.7**

(2) Else, if the first letter of this and the previous phrase are the same, it goes to the next line; else, it skips a line. Thus, between the A's (and a's) and the B's (and b's) there will be a blank line. Then, it prints both the phrase and the page number.

**A.4 TWO-LEVEL INDEXES**

A "two-level index" divides some phrases into two parts: a "generic" part and a "specific" part, e.g.:

```
compilers,
Algol 13, 15
Fortran 18
LISP 22
SAIL 15
computers,
analog 29
digital 18, 64, 88
consoles 16, 25
```

A.4
.PORTION INDEX
.WASL = WASR = WASP = NULL ;
.NOFill ; TABS 6 ;
.AT "$ L "$ R "$ >" PGNO "$ c
. IF *WASL[1] != *"L[1] THEN SKIP 1 ; <<1ST LETTER DIFFERENT>>
. IF WASL != "L" THEN
.  START "NEW GENERIC" BREAK ;
L{IF "R"*NULL THEN START "WITH SPECIFIC PART" },
\R##{   END "WITH SPECIFIC PART" ; )PGNO(
.   END "NEW GENERIC"
. ELSE IF WASR != "R" THEN
.  START "NEW SPECIFIC" BREAK ;
\R##PGNO{ END "NEW SPECIFIC"
. ELSE IF WASP != "PGNO" THEN START },PGNO{ END
. ) ;
.RECEIVE "<>" ;

To create the generated file for PORTION INDEX requires a signal response such as:

.AT "<" SPECIFIC */" GENERIC ">" c
. IF "SPECIFIC"*NULL THEN START }SPECIFIC { END ;
. )SPECIFIC{ ; comment print SPECIFIC and GENERIC in the text ;
. SEND INDEX c )<SPECIFIC/SPECIFIC>{PAGE!}%{ ;
. ) ;

Every phrase to be sent to the index should be bracketed as follows where it appears in the manuscript:

that there are more %<Algol/compilers> than Algol users ... the world's leading expert on %</consoles>.

A.5 KWIC INDEXES

This one is easy. The desired output is:

writers of compilers often balk 16
The XYZ compilers 35
certain Algol compilers 8
compilers I've known 41
IBM computer 72
uses of the computer in government 64

At the end of the manuscript:

A.5
SUBJECT AND COMMAND INDEX
(References are to Page numbers)

| SKIP 32 | BREAKING AT CARRIAGE-RETURNS 15 |
| ! 46    | BREAKING AT TABS 15               |
| <dc> 34 | BY 46                            |
| <e> 33  | C-value 46                        |
| <id> 38 | CALLING A MACRO FROM A COMMAND    |
| <oc> 34 | LINE 25                           |
| <qc> 34 | CALLING A MACRO FROM A TEXT LINE  |
| <s> 35  | 25                                |
| <se> 33 | CAPABILITIES 1                    |
| <v> 31  | carriage-return 15, 64            |
| ABOVE 57| CASE 56                           |
| ADJUST 14| CENTER 16, 19                    |
| AFTER 64| channels 53                       |
| ALIGN 56| CHAR 31, 41                       |
| alphabetize 52 | CHARACTER 57               |
| APART 17| CHARS 31                          |
| APPENDICES 49 | collating sequence 52         |
| APPENDIX 43| COLUMN 31                   |
| AREA DECLARATION 39 | COLUMNS 31, 40      |
| arithmetic 38 | COMMAND AND TEXT LINES 13    |
| ASCII 34, 52 | COMMAND CHARACTER STATEMENT 49 |
| Assignment expressions 33 | command line 13   |
| assignment statement 45 | Command Processing 20, 24|
| AT N 64 | COMMENTS 58                       |
| AT NULL 63 | COMPACT 16                   |
| AT PAGEMARK 63 | COMPILER OUTPUT 4         |
| AT S 64 | COMPUTED TEXT 23                  |
| AUTOMATICALLY DECLARED VARIABLES 31 | conditional expressions 33 |
| BEFORE 64 | CONDITIONAL STATEMENT 45         |
| BEFORE PAGE 65 | CONSTANTS 34               |
| BEGIN 35 | CONTENTS 49, 58, 69               |
| BELOW 57 | CONTINUE 41                       |
| BIBLIOGRAPHY 49 | CONTINUE STATEMENT 55       |
| BLOCK 35 | CONTROL CHARACTER ACTIVATION 21  |
| block structure 35, 45 | control characters 18         |
| BOX FRAME STATEMENT 56 | CONTROL FUNCTIONS 18       |
| BREAK STATEMENT 55 | COUNT 43, 46               |
| BREAKING AT BLANK LINES 15 | COUNTER DECLARATION 43     |

<p>| crown 14 | CROSS-REFERENCES 59 |</p>
<table>
<thead>
<tr>
<th>CRSPACE</th>
<th>GENERAL EXPRESSIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA TYPE</th>
<th>generated file</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>58, 52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATE</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>decimal</th>
<th>GROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>declaration</th>
<th>GROUP SKIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>54</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DECLARATIONS</th>
<th>HEADINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEVICE LDX</th>
<th>HEADINGS AND FOOTINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEVICE LPT</th>
<th>hem</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEVICE MIC</th>
<th>hierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEVICE STATEMENT</th>
<th>HORIZONTAL SPACE COMPACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEVICE TTY</th>
<th>hyphen</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dot in column</th>
<th>IF</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DOUBLE SPACE</th>
<th>ILLEGAL CHARACTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DOUBLE SPACING</th>
<th>imperative</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>edit</th>
<th>IMPERATIVES</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ELSE</th>
<th>INDENT DECLARATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>38</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>END</th>
<th>INDENT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>31, 38, 45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQUATION</th>
<th>INDENT2</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>31, 38, 45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>error messages</th>
<th>INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>49, 58, 52, 70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVERY HEADING</th>
<th>INLINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EXPRESSIONS</th>
<th>INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>38</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FALSE</th>
<th>JUSTIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>14, 18, 59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Justifying</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FILE</th>
<th>JUSTJUST</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FILL MODE</th>
<th>KWIC INDEXES</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>74</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>filling</th>
<th>LABELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8, 31, 63</td>
<td>60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FLUSH LEFT</th>
<th>LABELS AND CROSS-REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FLUSH RIGHT</th>
<th>LDX</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FONT</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FOOTING</th>
<th>LINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FOOTNOTE</th>
<th>LMARG</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>31, 45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FOOTNOTES</th>
<th>local</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>38</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FOOTSEP</th>
<th>LOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Forward Cross-Reference</th>
<th>MACRO CALLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>forward-references</th>
<th>59</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FROM</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td></td>
</tr>
</tbody>
</table>
string 34
STRING SPACE EXHAUSTED 52
Strings 38
Subscript 28
SUBSECTION 43
substring 33
SUPERIMPOSE [n] 16
Superscript 20
switches 3
syntax of PUB commands 5

TAB 19, 41
TABBREAK 15
TABLE 43
table of contents 50
TABS DECLARATION 37
TABSPACE 15
TEMPLATES 26
TEMPORARY AREAS 42
TEXT 39
TEXT AREA 40
TEXT CONVENTIONS 13
text line 13
text processing 23
TEXT RESPONSES 63
THEN 45
THOROUGHLY EXPLAINED EXAMPLES 68
TIME 32
TITLE AREA 40
TITLEPAGE 49
TOPLINE 31
TRANSITION RESPONSES 64
TRIPLE SPACE 39
TRUE 31
TURN OFF 22
TURN ON 21
TUTORIAL FOR BEGINNERS 5
TWO-LEVEL INDEXES 72

Underline 20
UNIMPLEMENTED STATEMENTS 56

VARIABLES 30